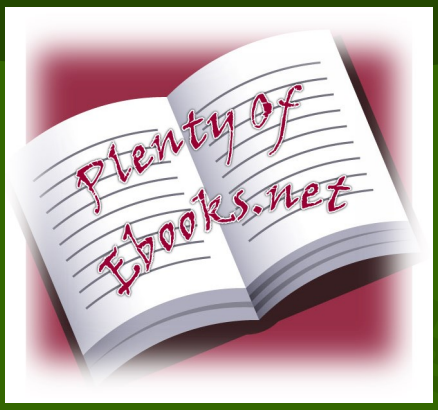


This eBook is downloaded from
www.PlentyofeBooks.net

PlentyofeBooks.net is a blog with an aim of helping people, especially students, who cannot afford to buy some costly books from the market.

For more Free eBooks and educational material visit
www.PlentyofeBooks.net

Uploaded By

amexy98
theBooks

THE EXPERT'S VOICE® IN ORACLE

Expert Oracle RAC 12c

GAIN DEEP EXPERTISE IN MANAGING ORACLE REAL APPLICATION CLUSTERS

Syed Jaffar Hussain, Tariq Farooq, Riyaj Shamsudeen, and Kai Yu

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

CHAPTER 1

Overview of Oracle RAC

by Kai Yu

In today's business world, with the growing importance of the Internet, more and more applications need to be available online all the time. One obvious example is the online store application. Many companies want to keep their online stores open 24x7 on 365 days so that customers from everywhere, in different time zones, can come at any time to browse products and place orders.

High Availability (HA) may also be critical for non-customer-facing applications. It is very common for IT departments to have complex distributed applications that connect to multiple data sources, such as those that extract and summarize sales data from online store applications to reporting systems. A common characteristic of these applications is that any unexpected downtime could mean a huge loss of business revenue and customers. The total loss is sometimes very hard to quantify with a dollar amount. As the key components of these applications, Oracle databases are often key components of a whole storefront ecosystem, so their availability can impact the availability of the entire ecosystem.

The second area is the scalability of applications. As the business grows, transaction volumes can double or triple as compared to what was scoped for the initial capacity. Moreover, for short times, business volumes can be very dynamic; for example, sales volumes for the holiday season can be significantly higher. An Oracle Database should be scalable and flexible enough to easily adapt to business dynamics and able to expand for high workloads and shrink when demand is reduced. Historically, the old Big Iron Unix servers that used to dominate the database server market lacked the flexibility to adapt to these changes. In the last ten years, the industry standard has shifted to x86-64 architecture running on Linux to meet the scalability and flexibility needs of growing applications. Oracle Real Application Clusters (RAC) running on Linux on commodity X86-64 servers is a widely adapted industry-standard solution to achieve high availability and scalability.

This chapter introduces the Oracle RAC technology and discusses how to achieve the high availability and scalability of the Oracle database with Oracle RAC. The following topics will be covered in this chapter:

- Database High Availability and Scalability
- Oracle Real Application Clusters (RAC)
- Achieving the Benefits of Oracle RAC
- Considerations for Deploying Oracle RAC

High Availability and Scalability

This section discusses the database availability and scalability requirements and their various related factors.

What Is High Availability?

As shown in the previous example of the online store application, business urges IT departments to provide solutions to meet the availability requirements of business applications. As the centerpiece of most business applications, database availability is the key to keeping all the applications available.

In most IT organizations, Service Level Agreements (SLAs) are used to define the application availability agreement between business and IT organization. They can be defined as the percentage availability, or the maximum downtime allowed per month or per year. For example, an SLA that specifies 99.999% availability means less than 5.26 minutes downtime allowed annually. Sometimes an SLA also specifies the particular time window allowed for downtime; for example, a back-end office application database can be down between midnight and 4 a.m. the first Saturday of each quarter for scheduled maintenance such as hardware and software upgrades.

Since most high availability solutions require additional hardware and/or software, the cost of these solutions can be high. Companies should determine their HA requirements based on the nature of the applications and the cost structure. For example some back-end office applications such as a human resource application may not need to be online 24x7. For those mission–critical business applications that need to be highly available, an evaluation of the cost of downtime may be calculated too; for example, how much money can be lost due to 1 hour of downtime. Then we can compare the downtime costs with the capital costs and operational expenses associated with the design and implementation of various levels of availability solution. This kind of comparison will help business managers and IT departments come up with realistic SLAs that meet their real business and affordability needs and that their IT team can deliver.

Many business applications consist of multi-tier applications that run on multiple computers in a distributed network. The availability of the business applications depends not only on the infrastructure that supports these multi-tier applications, including the server hardware, storage, network, and OS, but also on each tier of the applications, such as web servers, application servers, and database servers. In this chapter, I will focus mainly on the availability of the database server, which is the database administrator's responsibility.

Database availability also plays a critical role in application availability. We use *downtime* to refer to the periods when a database is unavailable. The downtime can be either unplanned downtime or planned downtime. Unplanned downtime can occur without being prepared by system admin or DBAs—it may be caused by an unexpected event such as hardware or software failure, human error, or even a natural disaster (losing a data center). Most unplanned downtime can be anticipated; for example, when designing a cluster it is best to make the assumption that everything will fail, considering that most of these clusters are commodity clusters and hence have parts which break. The key when designing the availability of the system is to ensure that it has sufficient redundancy built into it, assuming that every component (including the entire site) may fail. Planned downtime is usually associated with scheduled maintenance activities such as system upgrade or migration.

Unplanned downtime of the Oracle database service can be due to data loss or server failure. The data loss may be caused by storage medium failure, data corruption, deletion of data by human error, or even data center failure. Data loss can be a very serious failure as it may turn out to be permanent, or could take a long time to recover from. The solutions to data loss consist of prevention methods and recovery methods. Prevention methods include disk mirroring by RAID **(Redundant Array of Independent Disks)** configurations such as RAID 1 (mirroring only) and RAID 10 (mirroring and striping) in the storage array or with ASM (Automatic Storage Management) diskgroup redundancy setting. Chapter 5 will discuss the details of the RAID configurations and ASM configurations for Oracle Databases. Recovery methods focus on getting the data back through database recovery from the previous database backup or flashback recovery or switching to the standby database through Data Guard failover.

Server failure is usually caused by hardware or software failure. Hardware failure can be physical machine component failure, network or storage connection failure; and software failure can be caused by an OS crash, or Oracle database instance or ASM instance failure. Usually during server failure, data in the database remains intact. After the software or hardware issue is fixed, the database service on the failed server can be resumed after completing database instance recovery and startup. Database service downtime due to server failure can be prevented by providing redundant database servers so that the database service can fail over in case of primary server failure. Network and storage connection failure can be prevented by providing redundant network and storage connections.

Planned downtime for an Oracle database may be scheduled for a system upgrade or migration. The database system upgrade can be a hardware upgrade to servers, network, or storage; or a software upgrade to the OS, or Oracle Database patching and upgrade. The downtime for the upgrade will vary depending on the nature of the upgrade. One way to avoid database downtime for system upgrades is to have a redundant system which can take over the database workloads during the system upgrade without causing a database outage. Migration maintenance is sometimes necessary to relocate the database to a new server, a new storage, or a new OS. Although this kind of migration is less frequent, the potential downtime can be much longer and has a much bigger impact on the business application. Some tools and methods are designed to reduce database migration downtime: for example, Oracle transportable tablespace, Data Guard, Oracle GoldenGate, Quest SharePlex, etc.

In this chapter, I focus on a specific area of Oracle Database HA: server availability. I will discuss how to reduce database service downtime due to server failure and system upgrade with Oracle RAC. For all other solutions to reduce or minimize both unplanned and planned downtime of Oracle Database, we can use the Oracle Maximum Availability Architecture (MAA) as the guideline. Refer to the Oracle MAA architecture page, www.oracle.com/technetwork/database/features/availability/maa-090890.html, for the latest developments.

Database Scalability

In the database world, it is said that one should always start with application database design, SQL query tuning, and database instance tuning, instead of just adding new hardware. This is always true, as with a bad application database design and bad SQL queries, adding additional hardware will not solve the performance problem. On the other hand, however, even some well-tuned databases can run out of system capacity as workloads increase.

In this case, the database performance issue is no longer just a tuning issue. It also becomes a scalability issue. Database scalability is about how to increase the database throughput and reduce database response time, under increasing workloads, by adding more computing, networking, and storage resources.

The three critical system resources for database systems are CPU, memory, and storage. Different types of database workloads may use these resources differently: some may be CPU bound or memory bound, while others may be I/O bound. To scale the database, DBAs first need to identify the major performance bottlenecks or resource contentions with a performance monitoring tool such as Oracle Enterprise Manager or AWR (Automatic Workload Repository) report. If the database is found to be I/O bound, storage needs to be scaled up. In Chapter 5, we discuss how to scale up storage by increasing storage I/O capacity such as IOPs (I/O operations per second) and decrease storage response time with ASM striping and I/O load balancing on disk drives.

If the database is found to be CPU bound or memory bound, server capacity needs to be scaled up. Server scalability can be achieved by one of the following two methods:

- Scale-up or vertical scaling: adding additional CPUs and memory to the existing server.
- Scale-out or horizontal scaling: adding additional server(s) to the database system.

The scale-up method is relatively simple. We just need to add more CPUs and memory to the server. Additional CPUs can be recognized by the OS and the database instance. To use the additional memory, some memory settings may need to be modified in OS kernel, as well as the database instance initialization parameters. This option is more useful with x86 servers as these servers are getting more CPUs cores and memory (up to 80 cores and 4TB memory per server of the newer servers at the time of writing). The HP DL580 and DL980 and Dell R820 and R910 are examples of these powerful X86 servers. For some servers, such as those which are based on Intel's Sandybridge and Northbridge architectures, adding more memory with the older CPUs might not always achieve the same memory performance. One of the biggest issues with this scale-up method is that it can hit its limit when the server has already reached the maximal CPU and memory capacity. In this case, you may have to either replace it with a more powerful server or try the scale-out option.

The scale-out option is to add more server(s) to the database by clustering these servers so that workloads can be distributed between them. In this way, the database can double or triple its CPU and memory resources. Compared to the scale-up method, scale-out is more scalable as you can continue adding more servers for continuously increasing workloads.

One of the factors that will help to determine whether the scale-up or the scale-out option is more appropriate for your environment is the transaction performance requirements. If a lower transaction latency is the goal, the scale-up method may be the option to choose, as reading data from local memory is much faster than reading data from a remote server across the network due to the fact that memory speed is much faster than networking speed, even for the high-speed InfiniBand network. If increasing database transaction throughput is the goal, scale-out is the option to be considered, as it can distribute transaction loads to multiple servers to achieve much higher transaction throughput.

Other factors to be considered include the costs of hardware and software licenses. While the scale-up method may need a high-cost, high-end server to allow vertical scalability, the scale-out method will allow you to use low-cost commodity servers clustered together. Another advantage of the scale-out method is that this solution also confers high availability, which allows database transactions to be failed over to other low-cost servers in the cluster, while the scale-up solution will need another high-cost server to provide a redundant configuration. However, the scale-out method usually needs special licensed software such as Oracle RAC to cluster the applications on multiple nodes. While you may be able to save some hardware costs with the scale-out model, you need to pay for the licencing cost of the cluster software.

The scale-out method takes much more complex technologies to implement. Some of the challenges are how to keep multiple servers working together on a single database while maintaining data consistency among these nodes, and how to synchronize operations on multiple nodes to achieve the best performance. Oracle RAC is designed to tackle these technical challenges and make database servers work together as one single server to achieve maximum scalability of the combined resources of the multiple servers. Oracle RAC's cache fusion technology manages cache coherency across all nodes and provides a single consistent database system image for applications, no matter which nodes of the RAC database the applications are connected to.

Oracle RAC

This section discusses Oracle RAC: its architecture, infrastructure requirements, and main components.

Database Clustering Architecture

To achieve horizontal scalability or scale-out of a database, multiple database servers are grouped together to form a cluster infrastructure. These servers are linked by a private interconnect network and work together as a single virtual server that is capable of handling large application workloads. This cluster can be easily expanded or shrunk by adding or removing servers from the cluster to adapt to the dynamics of the workload. This architecture is not limited by the maximum capacity of a single server, as the vertical scalability (scale-up) method is. There are two types of clustering architecture:

- Shared Nothing Architecture
- Shared Everything Architecture

The shared nothing architecture is built on a group of independent servers with storage attached to each server. Each server carries a portion of the database. The workloads are also divided by this group of servers so that each server carries a predefined workload. Although this architecture can distribute the workloads among multiple servers, the distribution of the workloads and data among the servers is predefined. Adding or removing a single server would require a complete redesign and redeployment of the cluster.

For those applications where each node only needs to access a part of the database, with very careful partitioning of the database and workloads, this shared nothing architecture may work. If the data partition is not completely in sync with the application workload distribution on the server nodes, some nodes may need to access data stored in other nodes. In this case, database performance will suffer. Shared nothing architecture also doesn't work well with a large set of database applications such as OLTP (Online transaction processing), which need to access the entire database; this architecture will require frequent data redistribution across the nodes and will not work well. Shared nothing also doesn't provide high availability. Since each partition is dedicated to a piece of the data and workload which is not duplicated by any other server, each server can be a single point of failure. In case of the failure of any server, the data and workload cannot be failed over to other servers in the cluster.

In the shared everything architecture, each server in the cluster is connected to a shared storage where the database files are stored. It can be either active-passive or active-active. In the active-passive cluster architecture, at any given time, only one server is actively accessing the database files and handling workloads; the second one is passive and in standby. In the case of active server failure, the second server picks up the access to the database files and becomes the active server, and user connections to the database also get failed over to the second server. This active-passive cluster provides only availability, not scalability, as at any given time only one server is handling the workloads.

Examples of this type of cluster database include Microsoft SQL Server Cluster, Oracle Fail Safe, and Oracle RAC One Node. Oracle RAC One Node, introduced in Oracle Database 11.2, allows the single-instance database to be able to fail over to the other node in case of node failure. Since Oracle RAC One Node is based on the same Grid Infrastructure as Oracle RAC Database, it can be converted from one node to the active-active Oracle RAC Database with a couple of srvctl commands. Chapter 14 will discuss the details of Oracle RAC One Node.

In the active-active cluster architecture, all the servers in the cluster can actively access the database files and handle workloads simultaneously. All database workloads are evenly distributed to all the servers. In case of one or more server failures, the database connections and workloads on the failed servers get failed over to the rest of the surviving servers. This active-active architecture implements database server virtualization by providing users with a virtual database service. How many actual physical database servers are behind the virtual database service, and how the workloads get distributed to these physical servers, is transparent to users. To make this architecture scalable, adding or removing physical servers from the cluster is also transparent to users. Oracle RAC is the classic example of the active-active shared everything database architecture.

RAC Architecture

Oracle Real Application Cluster (RAC) is an Oracle Database option, based on a share everything architecture. Oracle RAC clusters multiple servers that then operate as a single system. In this cluster, each server actively accesses the shared database and forms an active-active cluster configuration. Oracle first introduced this active-active cluster database solution, called Oracle Parallel Server (OPS), in Oracle 6.2 on VAX/VMS. This name was used until 2001, when Oracle released Oracle Real Application Clusters (RAC) in Oracle Database 9i. Oracle RAC supersedes OPS with many significant enhancements including Oracle Clusterware and cache fusion technology.

In the Oracle RAC configuration, the database files are stored in shared storage, which every server in the cluster shares access to. As shown in Figure 1-1, the database runs across these servers by having one RAC database instance on a server. A database instance consists of a collection of Oracle-related memory plus a set of database background processes that run on the server. Unlike a single-node database, which is limited to one database instance per database, a RAC database has one or more database instances per database and is also built to add additional database instances easily. You can start with a single node or a small number of nodes as an initial configuration and scale out to more nodes with no interruption to the application. All instances of a database share concurrent access to the database files.

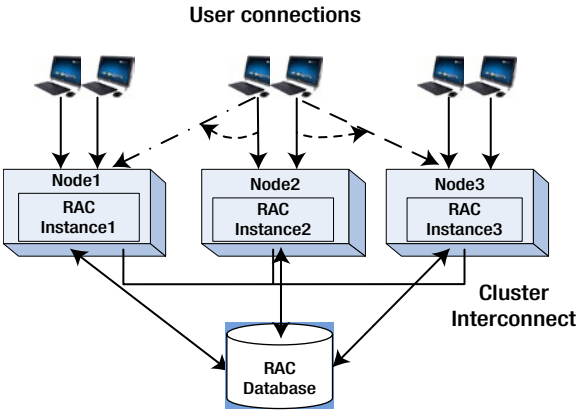


Figure 1-1. *Oracle RAC Database architecture*

Oracle RAC is designed to provide scalability by allowing all the RAC instances to share database workloads. In this way, Oracle RAC Database presents users with a logical database server that groups computing resources such as CPUs and memory from multiple RAC nodes. Most times, with proper configuration using RAC features such as services, Single Client Access Name (SCAN), and database client failover features, changes on the cluster configuration such as adding or removing nodes can be done as transparently to the users as possible. Figure 1-1 illustrates an Oracle RAC configuration where users are connected to the database and can perform database operations through three database instances.

This architecture also provides HA during a failure in the cluster. It can tolerate N-1 node failures, where N is the total number of the nodes. In case of one or more nodes failing, the users connecting to the failed nodes are failed over automatically to the surviving RAC nodes. For example, as shown in Figure 1-1, if node 2 fails, the user connections on instance 2 fail over to instance 1 and instance 3. When user connections fail over to the surviving nodes, RAC ensures load balancing among the nodes of the cluster.

Oracle RAC 12cR1 introduced a new architecture option called Flex Clusters. In this new option, there are two types of cluster nodes: Hub nodes and Leaf nodes. The Hub Nodes are same as the traditional cluster nodes in Oracle RAC 11gR2. All of the Hub Nodes are interconnected with the high-speed interconnect network and have direct access to shared storage. The Leaf Nodes are a new type of node with a lighter-weight stack. They are connected only with the corresponding attached Hub Nodes and they are not connected with each other. These Leaf Nodes are not required to have direct access to shared storage. Instead, they will perform storage I/O through the Hub Nodes that they attach to. The Flex Cluster architecture was introduced to improve RAC scalability. Chapter 4 will discuss the detailed configuration of this new feature in 12c.

Hardware Requirements for RAC

A typical Oracle RAC database requires two or more servers, networking across the servers, and the storage shared by the servers. Although the servers can be SMP Unix servers as well as low-cost commodity x86 servers, it has been an industry trend to move the database server from large SMP Unix machines to low-cost x86-64 servers running on Linux OS, such as Red Hat Enterprise Linux and Oracle Linux.

It is recommended that all the servers in any Oracle RAC cluster should have similar hardware architecture. It is mandatory to have the same OS, with possibly different patches among the servers on the same Oracle RAC. In order to ensure load balancing among the RAC cluster nodes, in 11gR2, server pool management is based on the importance of the server pool and the number of servers associated with the server pool, and there is no way to differentiate between the capacities of the servers. All the servers on the RAC cluster are assumed to have similar (homogeneous) capacity configuration such as CPU counts and total memory, as well as physical networks. If the servers are different in capacity, this will affect resource distribution and session load balancing on the RAC. In Oracle RAC 12c, the policy-based cluster management can manage clusters that consist of heterogeneous servers with different capabilities such as CPU power and memory sizes. With the introduction of server categorization, server pool management has been enhanced to understand the differences between servers in the cluster.

Each server should also have proper local storage for the OS, Oracle Grid Infrastructure software home, and possibly for Oracle RAC software home if you decide to use the local disk to store the RAC Oracle Database binary. Potentially, in the event of a RAC node failure, the workloads on the failed node will be failed over to the working nodes; so each RAC node should reserve some headroom for the computing resources to handle additional database workloads that are failed over from other nodes.

The storage where the RAC database files reside needs to be accessible from all the RAC nodes. Oracle Clusterware also stores two important pieces of Clusterware components—Oracle Cluster Registry (OCR) and voting disk files—in the shared storage. The accessibility of the shared storage by each of the RAC nodes is critical to Clusterware as well as to RAC Database. To ensure the fault tolerance of the storage connections, it is highly recommended to establish redundant network connections between the servers and shared storage. For example, to connect to a Fibre Channel (FC) storage, we need to ensure that each sever on the cluster has dual HBA(**Host Bus Adapter)** cards with redundant fiber links connecting to two fiber channel switches, each of which connects to two

FC storage controllers. On the software side, we need to configure multipathing software to group the multiple I/O paths together so that one I/O path can fail over I/O traffic to another surviving path in case one of the paths should fail. This ensures that at least one I/O path is always available to the disks in the storage.

Figure 1-2 shows a example of configurating two redundant storage network connections to a SAN storage. Depending on the storage network protocols, the storage can be linked with servers using either the FC or iSCSI network. To achieve high I/O bandwidth of the storage connections, some high-speed storage network solutions, such as 16GbE FC and 10gBe iSCSI, have been adapted for the storage network. The detailed configuration of shared storage is discussed in Chapter 5.

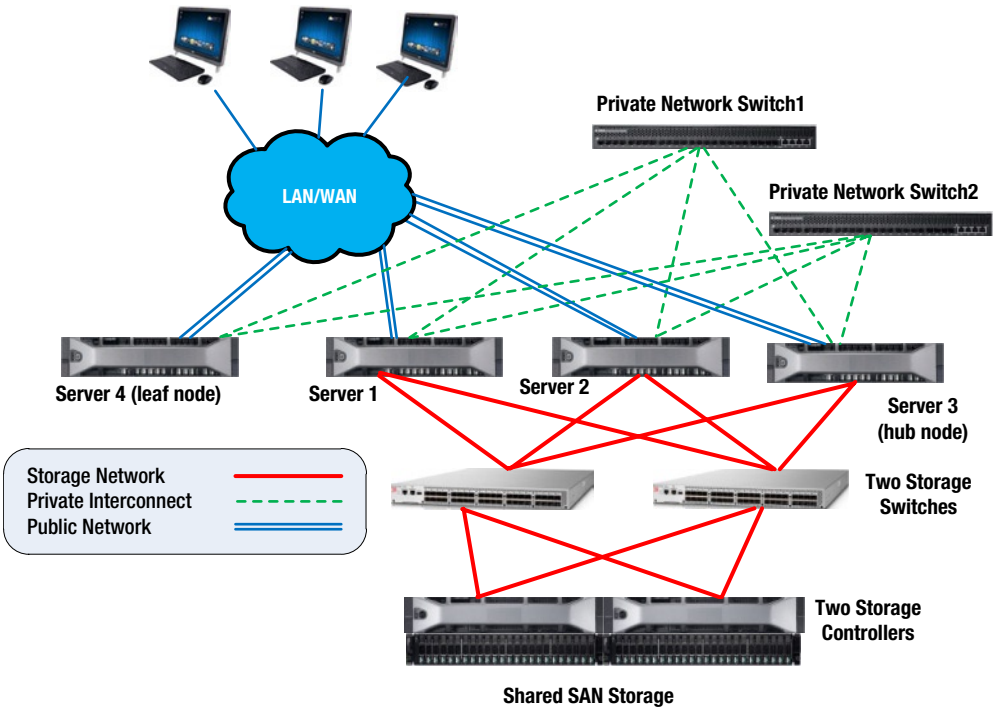


Figure 1-2. *Oracle RAC hardware architecture*

The network configuration for an Oracle RAC configuration includes the public network for users or applications to connect to the database, and the private interconnect network for connecting the RAC nodes in the cluster. Figure 1-2 illustrates these two networks in a two-node RAC database. The private interconnect network is accessible only to the nodes in the cluster. This private interconnect network carries the most important heartbeat communication among the RAC nodes in the cluster. The network is also used by the data block transfer between the RAC instances.

A redundant private interconnect configuration is highly recommended for a production cluster database environment: it should comprise at least two network interface cards (NICs) that are connected to two dedicated physical switches for the private interconnect network. These two switches should not be connected to any other network, including the public network. The physical network cards can be bound into a single logical network using OS network bonding utilities such as Linux Bonding or Microsoft NIC Teaming for HA and load balancing.

Oracle 11.2.0.2 introduced a new option for bonding multiple interconnect networks with an Oracle Redundant Interconnect feature called Cluster High Availability IP (HAIP), which provides HA and bonds the interfaces for aggregation at no extra cost to the Oracle environment. Oracle HAIP can take up to four NICs for the private network. Chapter 9 details some best practices for private network configuration. To increase the scalability of the Oracle RAC database, some advanced network solutions have been introduced. For example, as alternatives, 10g GbE network and InfiniBand network are widely used for the private interconnect, to alleviate the potential performance bottleneck.

In Oracle Clusterware 12cR1, Flex Clusters are introduced as a new option. If you use this option, Leaf Nodes are not required to have direct access to the shared storage, while Hub Nodes are required to have direct access to the shared storage, like the cluster nodes in an 11gR2 cluster. Figure 1-2 also illustrates the Flex Cluster structure where servers 1 to 3 are Hub Nodes that have direct access to storage, while server 4 is a Leaf Node that does not connect to shared storage and relies on a Hub Node to perform I/O operations. In release 12.1, all Leaf Nodes are in the same public and private network as the Hub Nodes.

It is recommended to verify that the hardware and software configuration and settings comply with Oracle RAC and Clusterware requirements, with one of these three verification and audit tools depending on the system:

- For a regular RAC system, use RACCheck RAC Configuration Audit Tool (My Oracle Support [MOS] note ID 1268927.1)
- For an Oracle Exadata system, run Exachk Oracle Exadata Database Machine exachk or HealthCheck (MOS note ID 1070954.1)
- For an Oracle Database Appliance, use ODAchk Oracle Database Appliance (ODA) configuration Audit Tool (MOS note ID: 1485630).

RAC Components

In order to establish an Oracle RAC infrastructure, you need to install the following two Oracle licensed products:

- Oracle Grid Infrastructure: This combines Oracle Clusterware and Oracle ASM. Oracle Clusterware clusters multiple interconnected servers (nodes). Oracle ASM provides the volume manager and database file system that is shared by all cluster nodes.
- Oracle RAC: This coordinates and synchronizes multiple database instances to access the same set of database files and process transactions on the same database.

Figure 1-3 shows the architecture and main components of a two-node Oracle RAC database. The RAC nodes are connected by the private interconnect network that carries the Clusterware heartbeat as well as the data transfer among the RAC nodes. All the RAC nodes are connected to shared storage to allow them to access it. Each RAC node runs Grid Infrastructure, which includes Oracle Clusterware and Oracle ASM. Oracle Clusterware performs cluster management, and Oracle ASM handles shared storage management. Oracle RAC runs above the Grid Infrastructure on each RAC node to enable the coordination of communication and storage I/O among the RAC database instances. In the next two sections, we will discuss the functionality and components of these two Oracle products.

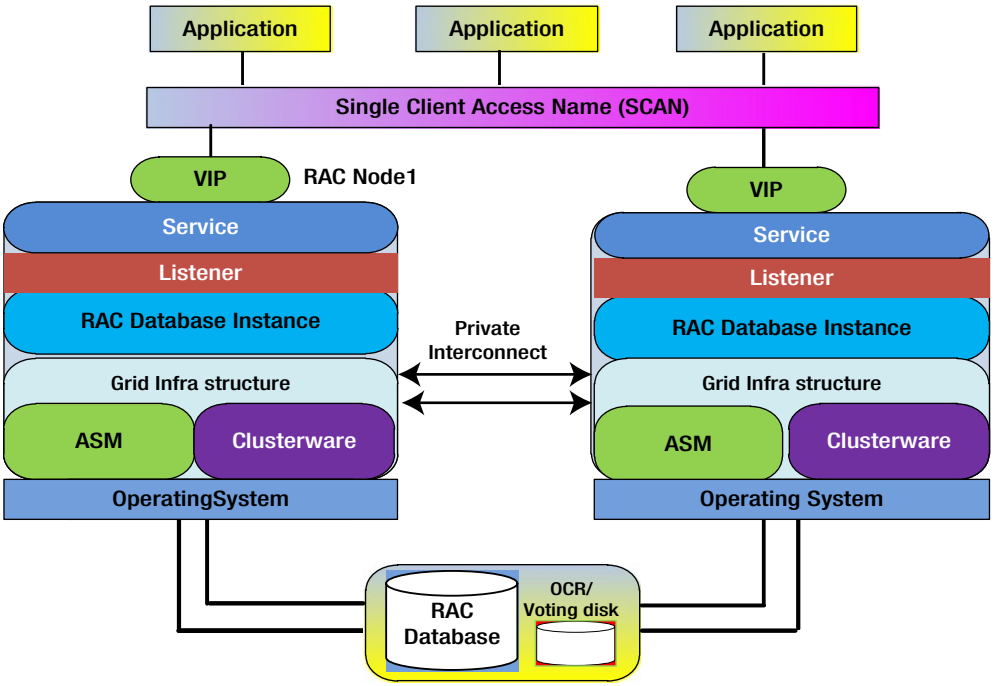


Figure 1-3. *Oracle RAC architecture and components*

Grid Infrastructure: Oracle Clusterware and ASM

Clusterware is a layer of software that is tightly integrated with the OS to provide clustering features to the RAC databases on a set of servers. Before Oracle 9i, Oracle depended on OS vendors or third-party vendors to provide the Clusterware solution. In Oracle 9i, Oracle released its own clusterware on Linux and Windows, and in Oracle 10g Oracle extended its clusterware to other OS. Oracle Clusterware was significantly enhanced in 11g. In 11gR2, Oracle combined Clusterware and Oracle ASM into a single product called Grid Infrastructure. Oracle Clusterware is required software to run the Oracle RAC option, and it must be installed in its own, nonshared Oracle home. Usually we have a dedicated OS user "grid" to own Grid Infrastructure as well as Oracle ASM instance, which is different from the Oracle RAC database owner "oracle."

Oracle Clusterware serves as a foundation for Oracle RAC Database. It provides a set of additional processes running on each cluster server (node) that allow the cluster nodes to communicate with each other so that these cluster nodes can work together as if they were one server serving the database users. This infrastructure is necessary to run Oracle RAC.

During Grid Infrastructure installation, ASM instances, database services, and virtual IP (VIP) services, the Single Client Access Name (SCAN), SCAN listener, Oracle Notification Service (ONS), and the Oracle Net listener are configured and also registered as Clusterware resources and managed with Oracle Clusterware. Then, after you create a RAC database, the database is also registered and managed with Oracle Clusterware. Oracle Clusterware is responsible for starting the database when the clusterware starts and restarting it once if fails.

Oracle Clusterware also tracks the configuration and status of resources it manages, such as RAC databases, ASM instances, database services, listeners, VIP addresses, ASM diskgroups, and application processes. These are known as Cluster Ready Service (CRS) resources. Oracle Clusterware checks the status of these resources at periodic intervals and restarts them a fixed number of times (depending on the type of resource) if they fail. Oracle Clusterware stores

the configuration and status of these CRS resources in OCR in the shared storage so that the Clusterware on each RAC node can access it. The configuration and the status information are used by Oracle Clusterware to manage these resources. You can use the following crsctl command to check the status of these resources:

```
[grid@k2r720n1 ~]$ crsctl stat res –t –

--------------------------------------------------------------------------------
NAME               TARGET    STATE        SERVER                   STATE_DETAILS
--------------------------------------------------------------------------------
Local Resources
--------------------------------------------------------------------------------
ora.ACFSHOME.dg
                   ONLINE    ONLINE       k2r720n1
ora.DATA.dg
                   ONLINE    ONLINE       k2r720n1
ora.LISTENER.lsnr
                   ONLINE    ONLINE       k2r720n1
ora.PCIE_DATA.dg
                   ONLINE    OFFLINE      k2r720n1
ora.VOCR.dg
                   ONLINE    ONLINE       k2r720n1
ora.asm
                   ONLINE    ONLINE       k2r720n1                 Started
ora.gsd
                   OFFLINE   OFFLINE      k2r720n1
ora.net1.network
                   ONLINE    ONLINE       k2r720n1
ora.ons
                   ONLINE    ONLINE       k2r720n1

--------------------------------------------------------------------------------
Cluster Resources
--------------------------------------------------------------------------------
ora.LISTENER_SCAN1.lsnr
      1            ONLINE    ONLINE       k2r720n1
ora.cvu
      1            ONLINE    ONLINE       k2r720n1
ora.k2r720n1.vip
      1            ONLINE    ONLINE       k2r720n1
ora.k2r720n2.vip
      1            ONLINE    INTERMEDIATE k2r720n2
ora.khdb.db
      1            ONLINE    ONLINE       k2r720n1                 Open
      2            ONLINE    ONLINE       k2r720n2                 Open
ora.oc4j
      1            ONLINE    ONLINE       k2r720n1
ora.scan1.vip
      1            ONLINE    ONLINE       k2r720n1
```

You also can use the SRVCTL command to manage each individual resource. For example, to check the RAC database status:

```
grid@k2r720n1 ~]$ srvctl status database -d khdb
Instance khdb1 is running on node k2r720n1
Instance khdb2 is running on node k2r720n2
```

Oracle Clusterware requires shared storage to store its two components: voting disk for node membership and Oracle Clusterware Registry (OCR) for cluster configuration information. The private interconnect network is required between the cluster nodes to carry the network heartbeat; among them, Oracle Clusterware consists of several process components which provide event monitoring, high availability features, process monitoring, and group membership of the cluster. In Chapter 2, we discuss more details of these components, including the process structure of the Clusterware and OCR and voting disks, best practices for managing Clusterware, and related troubleshooting methods.

Another component of the Oracle Grid Infrastructure is Oracle ASM, which is installed at the same time into the same Oracle home directory as Oracle Clusterware. Oracle ASM provides the cluster-aware shared storage and volume manager for RAC database files. It also provides shared storage for OCR and voting disks. Chapter 5 discusses the Oracle ASM architecture and management practices.

Oracle RAC: Cache Fusion

Oracle RAC is an option that you can select during Oracle database software installation. Oracle RAC and Oracle Grid Infrastructure together make it possible to run a multiple-node RAC database. Like the single-node database, each RAC database instance has memory structure such as buffer cache, shared pool, and so on. It uses the buffer cache in a way that is a little different from a single instance. For a single instance, the server process first tries to read the data block from the buffer cache. If the data block is not in the buffer cache, the server process will do the physical I/O to get the database block from the disks.

For a multi-node RAC database instance, the server process reads the data block from an instance's buffer cache, which has the latest copy of the block. This buffer cache can be on the local instance or a remote instance. If it is on a remote instance, the data block needs to be transferred from the remote instance through the high-speed interconnect. If the data block is not in any instance's buffer cache, the server process needs to do the physical read from the disks to the local cache. The instance updates the data block in the buffer cache and then the DBwriter writes the updated dirty blocks to the disk in a batch during the checkpoint or when the instance needs to get free buffer cache slots.

However, in Oracle RAC, multiple database instances are actively performing read and write operations on the same database, and these instances can access the same piece of data at the same time. To provide cache coherency among all the cluster nodes, the writer operation is serialized across all the cluster nodes so that at any moment, for any piece of data, there is only one writer. This is because if each instance acted on its own for the read and update operations on its own buffer cache, and the dirty block wrote to the disk without coordination and proper management among the RAC instances, these instances might access and modify the same data blocks independently and end up by overwriting each others' updates, which would cause data corruption.

In Oracle RAC, this coordination relies on communication among RAC instances using the high-speed interconnect. This interconnect is based on a redundant private network which is dedicated to communication between the RAC nodes. Oracle Clusterware manages and monitors this private interconnect using the cluster heartbeat between the RAC nodes to detect possible communication problems.

If any RAC node fails to get the heartbeat response from another RAC node within a predefined time threshold (by default 30 seconds), Oracle Clusterware determines that there is a problem on the interconnect between these two RAC nodes, and therefore the coordination between the RAC instances on these two RAC nodes may fail and a possible split-brain condition may occur in the cluster. As a result, Clusterware will trigger a node eviction event to reboot one of the RAC nodes, thus preventing the RAC instance from doing any independent disk I/O without coordinating with another RAC instance on another RAC node. This methodology is called I/O fencing.

Oracle uses an algorithm called STONITH (Shoot The Other Node In The Head), which allows the healthy nodes to kill the sick node by letting the sick node reboot itself. Since 11.2.0.2 with the introduction of reboot-less node eviction, in some cases the node reboot may be avoided by just shutting down and restarting the Clusterware. While Oracle Clusterware guarantees interconnect communication among the RAC nodes, Oracle RAC provides coordination and synchronization and data exchanging between the RAC instances using the interconnect.

In the Oracle RAC environment, all the instances of a RAC database appear to access a common global buffer cache where the query on each instance can get the up-to-date copy of a data block, also called the "master copy," even though the block has been recently updated by another RAC instance. This is called global cache coherency. In this global cache, since resources such as data blocks are shared by the database process within a RAC instance and across all RAC instances, coordination of access to the resources is needed across all instances. Coordination of access to these resources within a RAC instance is done with latches and locks, which are the same as those in a single-instance database. Oracle cache fusion technology is responsible for coordination and synchronization of access to these shared resources between RAC instances to achieve global cache coherency:

1. Access to shared resources between instances is coordinated and protected by the global locks between the instances.

2. Although the actual buffer cache of each instance still remains separate, each RAC instance can get the master copy of the data block from another instance's cache by transferring the data block from the other cache through the private interconnect.

Oracle Cache Fusion has gone through several major enhancements in various versions of Oracle Database. Before the Cache Fusion technology was introduced in Oracle 8.1.5, the shared disk was used to synchronize the updates—one instance needs to write the updated data block to the storage immediately after the block is updated in the buffer cache so that the other instance can read the latest version of the data block from the shared disk.

In Oracle 8.1.5, Cache Fusion I was introduced to allow the Consistent Read version of the data block to be transferred across the interconnect. Oracle 9i introduced Cache Fusion II to dramatically reduce latency for the write-write operations. With Cache Fusion II, if instance A needs to update a data block which happens to be owned by instance B, instance A requests the block through the Global Cache Service (GCS), instance B gets notification from the GCS and releases the ownership of the block and sends the block to instance A through the interconnect. This process avoids the disk write operation of instance B and disk read operation of instance A, which were required prior to Oracle 9i. This was called a disk ping and was highly inefficient for this multiple instance's write operation.

Since the introduction of Cache Fusion II , in Oracle RAC Database, coordination and synchronization between the RAC database instances have been achieved by two RAC services: the Global Cache Service (GCS) and Global Enqueue Service (GES) along with a central repository called the Global Resource Directory (GRD). These two services are the integrated part of Oracle RAC, and they also rely on the clusterware and private interconnects for communications between RAC instances. Both GES and GCS coordinate access to shared resources by RAC instances. GES manages enqueue resources such as the global locks between the RAC instances, and the GCS controls global access to data block resources to implement global cache coherency.

Let's look at how these three components work together to implement global cache coherency and coordination of access to resources in the RAC across all the RAC instances.

In Oracle RAC, multiple database instances share access to resources such as data blocks in the buffer cache and the enqueue. Access to these shared resources between RAC instances needs to be coordinated to avoid conflict. In order to coordinate and manage shared access to these resources, information such as data block ID, which RAC instance holds the current version of this data block, and the lock mode in which this data block is held by each instance is recorded in a special place called the Global Resource Directory (GRD). This information is used and maintained by GCS and GES for global cache coherency and coordination of access to resources such as data blocks and locks.

The GRD tracks the mastership of the resources, and the contents of the GRD are distributed across all the RAC instances, with the amount being equally divided across the RAC instances using a mod function when all the nodes of the cluster are homogeneous. The RAC instance that holds the GRD entry for a resource is the master instance of

the resource. Initially, each resource is assigned to its master instance using a hashing algorithm. The master instance can be changed when the cluster is reconfigured when adding or removing an instance from the cluster. This process is referred as the "reconfiguration."

In addition to reconfiguration, the resource can be remastered through Dynamic Resource Mastering (DRM). DRM can be triggered by resource affinity or an instance crash. Resource affinity links the instance and resources based on the usage pattern of the resource on the instance. If a resource is accessed more frequently from another instance, the resource can be remastered on another instance. The master instance is a critical component of global cache coherency. In the event of failure of one or more instances, the remaining instances will reconstruct the GRD. This ensures that the global GRD is kept as long as one instance of the RAC database is still available.

The GCS is one of the services of RAC that implement Oracle RAC cache fusion. In the Oracle RAC environment, a data block in an instance may be requested and shared by another instance. The GCS is responsible for the management of this data block sharing between RAC instances. It coordinates access to the database block by RAC instances, using the status information of the data blocks recorded in the entry of the GRD. The GCS is responsible for data block transfers between RAC instances.

The GES manages the global enqueue resources much as the GCS manages the data block resource. The enqueue resources managed by GES include library cache locks, dictionary cache locks, transaction locks, table locks, etc.

Figure 1-4 shows a case in which an instance requests a data block transfer from another instance.

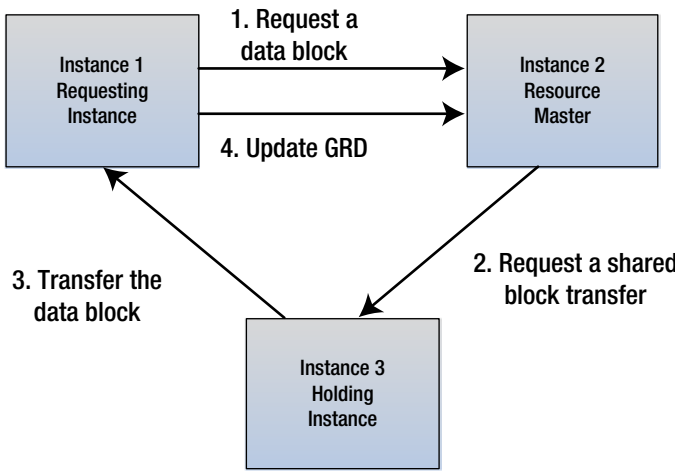


Figure 1-4. *Obtaining a data block from another instance*

Instance 1 needs access to a data block. It first identifies the resource master instance of the block, which is instance 2, and sends a request to instance 2 through GCS.

From the entry of the GRD for the block in resource master instance 2, the GCS gets the lock status of the data block and identifies that the holding instance is instance 3, which holds the latest copy of the data block; then GCS requests instance 3, the shared resource of the data block, and the block transfer to instance 1.

Instance 3 transfers the copy of the block to instance 1 via the private interconnects.

After receiving the copy of the block, instance 1 sends a message to the GCS about receiving the block, and the GCS records the block transfer information in GRD.

RAC Background Processes

Each RAC database instance is a superset of a single-node instance. It has the same set of background processes and the same memory structure, such as the **System Global Area** (SGA) and the Program Global Area (PGA). As well as these, RAC instances also have the additional processes and memory structure that are dedicated to the GCS processes, GES, and the global cache directory. These processes are as follows:

- LMS: Lock Manager Server process
- LMON: Lock Monitor processes
- LMD: Lock Monitor daemon process
- LCK: Lock process
- DIAG: Diagnostic daemon

1. LMS process: The Lock Manager Server is the Global Cache Service (GCS) process. This process is responsible for transferring the data blocks between the RAC instances for cache fusion requests. For a Consistent Read request, the LMS process will roll back the block and create the Consistent Read image of the block and then transfer the block to the requesting instance through the high-speed interconnect. It is recommended that the number of the LMS processes is less than or equal to the number of physical CPUs. Here the physical CPUs are the "CPU cores"; for example, for a server with two sockets that has four cores, the number of the physical CPU is 8. By default, the number of LMS processes is based on the number of the CPUs on the server. This number may be too high as one LMS process may be sufficient for up to four CPUs. There are a few ways to control the number of the LMS processes. You can modify the values for the init.ora parameter CPU_COUNT, which will also indirectly control the number of LMS processes that will be started during the Oracle RAC Database instance startup. The number of LMS processes is directly controlled by the init.ora parameter GCS_SERVER_PROCESSES. For a single CPU server, only one LMS is started. If you are consolidating multiple small databases on a cluster environment, you may want to reduce the number of LMS processes per instance, as there may be multiple instances of RAC databases on a single RAC node. Refer to the Oracle support note [ID 1439551.1] "Oracle (RAC) Database Consolidation Guidelines for Environments Using Mixed Database Versions" for detailed guidelines for setting LMS processes for multiple databases of RAC.
2. LMON process: The Lock Monitor process is responsible for managing the Global Enqueue Service (GES). It is also responsible for reconfiguration of lock resources when an instance joins or leaves the cluster and responsible for dynamic lock remastering.
3. LMD process: The Lock Monitor daemon process is the Global Enqueue Service (GES). The LMD process manages the incoming remote lock requests from other instances in the cluster.
4. LCK process: The Lock process manages non–cache fusion resource requests, such as row cache and library cache requests. Only one LCK process (lck0) per instance.
5. DIAG process: The Diagnostic daemon process is responsible for all the diagnostic work in a RAC instance.

In addition, Oracle 11gR2 introduced a few new processes for RAC. These processes are as follows:

ACMS: Atomic Controlfile to Memory Server

GTXn: Global Transaction process

LMHB: LM heartbeat monitor (monitors LMON, LMD, LMSn processes)

PING: Interconnect latency measurement process

RMS0: RAC management server

RSMN: Remote Slave Monitor

The following command shows these five background processes on a RAC node. This example shows that both the khdb1 instance and the ASM1 instance have a set of background processes. The Grid user owns the background processes for the ASM instance and the Oracle user owns the background processes for the RAC database instance 'khdb1'. If you have run multiple RAC databases on the RAC node, you will see multiple sets of the background processes. The process-naming convention is 'ora_<process>_<instance_name>', for example 'ora_lms2_khdb1' and 'asm_lms0_+ASM1'.

```
$ ps -ef | grep -v grep | grep 'lmon\|lms\|lck\|lmd\|diag\|acms\|gtx\|lmhb\|ping\|rms\|rsm'
grid      6448     1  0 Nov08 ?        00:13:48 asm_diag_+ASM1
grid      6450     1  0 Nov08 ?        00:01:43 asm_ping_+ASM1
grid      6455     1  0 Nov08 ?        00:34:52 asm_lmon_+ASM1
grid      6457     1  0 Nov08 ?        00:26:20 asm_lmd0_+ASM1
grid      6459     1  0 Nov08 ?        00:58:00 asm_lms0_+ASM1
grid      6463     1  0 Nov08 ?        00:01:17 asm_lmhb_+ASM1
grid      6483     1  0 Nov08 ?        00:02:03 asm_lck0_+ASM1
oracle   21797     1  0 Nov19 ?        00:07:53 ora_diag_khdb1
oracle   21801     1  0 Nov19 ?        00:00:57 ora_ping_khdb1
oracle   21803     1  0 Nov19 ?        00:00:42 ora_acms_khdb1
oracle   21807     1  0 Nov19 ?        00:48:39 ora_lmon_khdb1
oracle   21809     1  0 Nov19 ?        00:16:50 ora_lmd0_khdb1
oracle   21811     1  0 Nov19 ?        01:22:25 ora_lms0_khdb1
oracle   21815     1  0 Nov19 ?        01:22:11 ora_lms1_khdb1
oracle   21819     1  0 Nov19 ?        01:22:49 ora_lms2_khdb1
oracle   21823     1  0 Nov19 ?        00:00:41 ora_rms0_khdb1
oracle   21825     1  0 Nov19 ?        00:00:55 ora_lmhb_khdb1
oracle   21865     1  0 Nov19 ?        00:04:36 ora_lck0_khdb1
oracle   21867     1  0 Nov19 ?        00:00:48 ora_rsmn_khdb1
oracle   21903     1  0 Nov19 ?        00:00:45 ora_gtx0_khdb1
```

Besides these processes dedicated to Oracle RAC, a RAC instance also has other background processes which it has in common with a single node database instance. On Linux or Unix, you can see all the background processes of a RAC instance by using a simple OS command, for example: $ ps -ef|grep 'khdb1'

Or run the following query in SQL*Plus:

```
sqlplus> select NAME, DESCRIPTION  from  v$bgprocess where PADDR != '00'
```

Achieving the Benefits of Oracle RAC

In the last few sections we have examined the architecture of Oracle RAC and its two major components: Oracle Clusterware and Oracle RAC Database. In this section, we discuss how Oracle RAC technology achieves HA and scalability of Oracle Database.

High AvailabilityAgainst Unplanned Downtime

The Oracle RAC solution prevents unplanned downtime of the database service due to server hardware failure or software failure. In the Oracle RAC environment, Oracle Clusterware and Oracle RAC work together to allow the Oracle Database to run across multiple clustered servers. In the event of a database instance failure, no matter whether the failure is caused by server hardware failure or an OS or Oracle Database software crash, this clusterware provides the high availabilityand redundancy to protect the database service by failing over the user connections on the failed instance to other database instances.

Both Oracle Clusterware and Oracle RAC contribute to this high availability database configuration. Oracle Clusterware includes the High Availability (HA) service stack which provides the infrastructure to manage the Oracle Database as a resource in the cluster environment. With this service, Oracle Clusterware is responsible for restarting the database resource every time a database instance fails or after a RAC node restarts. In the Oracle RAC Database environment, the Oracle Database along with other resources such as the virtual IP (VIP) are managed and protected by Oracle Clusterware. In case of a node failure, Oracle Clusterware fails over these resources such as VIP to the surviving nodes so that applications can detect the node failure quickly without waiting for a TCP/IP timeout. Then, the application sessions can be failed over to the surviving nodes with connection pool and Transparent Application Failover (TAF).

If a database instance fails while a session on the instance is in the middle of a DML operation such as inserting, updating, or deleting, the DML transaction will be rolled back and the session will be reconnected to a surviving node. The DML of the transaction would then need to be started over. Another great feature of the clusterware is the Oracle Notification Services (ONS). ONS is responsible for publishing the Up and Down events on which the Oracle Fast Application Notification (FAN) and Fast Connect Failover (FCF) rely to provide users with fast connection failover to the surviving instance during a database instance failure.

Oracle RAC database software is cluster-aware. It allows Oracle RAC instances to detect an instance failure. Once an instance failure is detected, the RAC instances communicate with each other and reconfigure the cluster accordingly. The instance failure event triggers the reconfiguration of instance resources. During the instances' startup, these instance resources were distributed across all the instances using a hashing algorithm. When an instance is lost, the reconfiguration reassigns the new master instance for those resources that used the failed instance as the master instance. This reconfiguration ensures that the RAC cache fusion survives the instance failure. The reconfiguration is also needed when an instance rejoins the cluster once the failed server is back online, as this allows further redistribution of the mastership with the newly joined instance. But this reconfiguration process that occurs when adding a new instance takes less work than the one that occurs with a leaving instance, as when an instance is leaving the cluster, those suspected resources need to be replayed and the masterships need to be re-established.

DRM is different from reconfiguration. **DRM** is a feature of Global Cluster Service that changes the master instance of a resource based on resource affinity. When the instance is running on an affinity-based configuration, DRM remasters the resource to another instance if the resource is accessed more often from another node. Therefore, DRM occurs when the instance has a higher affinity to some resources than to others, whereas reconfiguration occurs when an instance leaves or joins the cluster.

In the Oracle 12c Flex Cluster configuration, a Leaf node connects to the cluster through a Hub node. The failure of the Hub Node or the failure of network between the Hub node and the Leaf nodes results in the node eviction of the associated Leaf nodes. In Oracle RAC 12cR1, since there is no user database session connecting to any Leaf Nodes, the failure of a Leaf Node will not directly cause user connection failure. The failure of the Hub Node is handled in essentially the same way as the failover mechanism of a cluster node in 11gR2.

RAC resource mastering is performed only on the Hub node instances, not on a Leaf node. This ensures that the failure of a Leaf node does not require remastering and also ensures that masters have affinity to the Hub node instances.

Oracle RAC and Oracle Clusterware also work together to allow application connections to perform seamless failover from the failed instance to the surviving instance. The applications can use these technologies to implement smooth failover for their database operations such as query or transactions. These technologies include:

1. Transparent Application Failover (TAF)
2. Fast Connect Failover (FCF)
3. Better Business continuity & HA using Oracle 12c Application Continuity (AC)

Transparent Application Failover (TAF)

Transparent Application Failover (TAF) is a feature that helps database connection sessions fail over to a surviving database instance during an instance failure. This is a client-side failover. With this feature, you can specify how to fail over the session and re-establish the session on another instance, and how the query of the original database connection continues after the connection gets relocated to the new database instance. It should be mentioned that only a query operation such as a select statement gets replayed after the connection is relocated to the new database instance. However, active transaction operations such as DML statements will not be failed over and replayed, as TAF can't preserve these active transactions. During an instance failure, these transactions will be failed and rolled back, and the application will receive an error message about the transaction failure.

The configuration of TAF is done through the tnsname.ora file on the client side without a need for any application code change.

```
KHDB_Sales =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = kr720n-scan)(PORT = 1521))
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = khdb_sales.dblab.com)
    (FAILOVER_MODE =
    (TYPE=session)
    (METHOD=basic)
    (RETRIES=10)
    (DELAY=10)
  ) ))
```

The failover mode is specified by the TYPE parameter, with three possible values: "session" for the session mode; "select" for the select mode; "none" for deactivating the failover. The session mode is a basic configuration. In this mode, TAF just reconnects the session to another instance and the new session has to start over again. In the select mode, TAF reconnects the session to another instance and allows the query to reuse the open cursor from the previous session. You can deactivate the failover if you put "none" or just skip the FAILOVER_MODE clause.

The options for the failover method are "basic" or "preconnect." Using the basic failover method, TAF re-establishes the connection to another instance only after the instance failure. In the preconnect method, the application preconnects a session to a backup instance. This will speed up failover and thus avoid the huge sudden reconnection storm that may happen to the surviving instance in the basic failover method. This is especially serious in a two-node RAC, where all the connections on the failed instance will need to be reconnected to the only surviving instance during an instance failure.

TAF is relatively easy to configure; however, this feature requires the OCI (Oracle Call Interface) library and doesn't work with a JDBC thin driver, which is widely used in many Java applications. Another disadvantage is that TAF mainly works with the session that runs the SELECT query statement. If the session fails in the middle of executing a DML or DDL or a PL/SQL program such as a stored procedure, a function, or a package, it will receive the ORA-25408 error, and the database will roll back the transaction. The application needs to reissue the statement after failover. These disadvantages lead us on to a discussion of another alternative called the Fast Connect Failover (FCF).

Fast Connect Failover (FCF)

Fast Connect Failover (FCF) provides a better way to fail over and recover the database connection transparently during an instance failure. The database clients are registered with Fast Application Notification (FAN), a RAC HA framework that publishes Up and Down events for the cluster reconfiguration. The database clients are notified of these Up and Down events published by FAN and react to the events accordingly. When an instance fails, FCF allows all database clients that connect to the failed instance to be quickly notified about the failed instance by receiving the Down event. Then, these database clients will stop and clean up the connections and immediately establish new connections to the surviving instance. FCF is supported by JDBC and OCI clients, Oracle Universal Connection Pool(UCP), and Oracle Data Providers for .Net. Oracle FCF is more flexible than TAF.

Connect to the RAC Database with VIPs

In the configuration of a database connection using tnsnames.ora and Java thin client driver, a VIP (instead of the database hostname (IP)) must be used for the hostname to avoid the TCP timeout issue, as shown in the following example.

```
KHDB =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = kr720n1-vip)(PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = kr720n2-vip)(PORT = 1521))
    (CONNECT_DATA =
      (SERVICE_NAME = khdb.dblab.com)
       )
    )
   )
```

This is because if the host is not accessible, then the user that connects this host with the hostname has to wait for the TCP/IP timeout to determine the host connection failure. Moreover, this timeout can range from a few seconds to a few minutes. In the worst case, therefore, the client may have to wait for a few minutes to determine if the host is actually down. During these few minutes, the database instance may have already been down and the database connection may be frozen, but the client does not know about the down event and will not fail over until the TCP/IP timeout is completed.

The solution to this problem is to connect the database using the VIP in the connection configuration to eliminate this timeout issue. The VIP is a CRS resource managed by Oracle Clusterware. When the host fails, the CRS automatically relocates the VIP to a surviving node, which avoids waiting for the TCP/IP timeout. However, after the relocation, since the listener on the new node listens only to the native VIP on the node, not the VIP relocated from the other node, this relocated VIP will not have a listener to listen to any database request on this VIP. Any connection on this VIP will receive the ORA-12541 no listener error. After receiving the error, the client will try the next address to connect to the database. In the preceding example of KHDB connection string, when node 1 k2r720n1 fails, the kr720n1-vip fails over to node 2 k2r720n2. Any connection using kr720n1-vip will receive the ORA-12541 no listener error and TAF will switch the connection to the next entry on the address list to connect to node 2's VIP: kr720n2-vip. This switch is immediate without waiting for the TCP/IP timeout.

Application Continuity (AC) of Oracle 12c

In pre–Oracle 12c Database, depending on the applications, application errors may occur during a database instance outage despite the successful commit of the transaction at the moment of failure. Such errors may leave applications in doubt, and users may receive errors or need to log in again or resubmit requests, etc. These problems are due to the lack of a way for applications to know the outcome of a failed transaction, the inability to migrate the workload that is affected by the planned or unplanned outage, and also the need to repair the workload.

To ensure that applications are minimally impacted in the event of node failure or instance failure, Oracle 12c introduces a new solution called Application Continuity (AC). This new feature masks recoverable outages from end-users and applications by replaying the database request at another Oracle RAC instance (for RAC) or another database for the standby database. This feature is designed to preserve the commit outcome and ensure application continuity for both unplanned and planned downtime.

High Availability Against Planned Downtime

Oracle RAC helps achieve database HA by reducing database service downtime due to the scheduled maintenance of the database infrastructure. Scheduled maintenance work may include server hardware upgrades or maintenance, server OS upgrade, and Oracle software upgrades. Depending on the task, these maintenance jobs may require bringing down the database instance or OS, or the server hardware itself. With Oracle RAC, maintenance work can be done in rolling fashion without the need to bring down the entire database service. Let's see how to perform rolling-fashion maintenance for different types of maintenance tasks.

Hardware maintenance of the database server may be needed during the lifetime of the server, for example upgrading or replacing hardware components such as CPUs, memory, and network cards. Although server downtime is required for this kind of maintenance, with Oracle RAC the database connections on the database instance of the impacted server can be relocated to other instances on the cluster. The maintenance can be done in rolling fashion by the following steps:

1. Relocate the database connections to the other instance;
2. Shut down the entire software stack in this order:
    a) Database instance
    b) ASM and Clusterware
    c) Operating system
    d) Server hardware
3. Perform the server hardware upgrade;
4. Restart the software stack in the reverse order
5. Repeat steps 1 to 4 for all other servers in the cluster.

Since the database connections are relocated to the other instance during hardware maintenance, the database service outage is eliminated. This rolling-fashion system maintenance enables system upgrades without database service downtime.

A similar rolling-upgrade method applies to OS upgrades, as well as other utility upgrades such as firmware, BIOS, network driver, and storage utility upgrades. Follow steps 1 to 5 except for the server hardware shutdown at step 2, and perform the OS or utility upgrade instead of the hardware upgrade at step 3.

You can also perform a rolling upgrade of Oracle Clusterware and ASM to avoid Clusterware and ASM downtime. In Oracle RAC 12.1, you can use Oracle Universal Installer (OUI) and Oracle Clusterware to perform a rolling upgrade to apply a patchset release of Oracle Clusterware. This allows you to shut down and patch RAC instances one or more at a time while keeping other RAC instances available online. You can also upgrade and patch clustered Oracle ASM instances in rolling fashion. This feature allows the clustered ASM environment to continue to function while

one or more ASM instances run different software releases and you are doing the rolling upgrade of the Oracle ASM environment. Many of these rolling-upgrade features were available in releases prior to RAC 12c, but they are easier to do with the GUI in Oracle 12cR1.

In order to apply the rolling upgrade for Oracle RAC software, the Oracle RAC home must be on a local file system on each RAC node in the cluster, not in a shared file system. There are several types of patch for an Oracle RAC database: interim patch, bundle patch, patch set upgrades (PSU), critical patch update (CPU), and diagnostic patch. Before you apply a RAC database patch, check the readme to determine whether or not that patch is certified for the rolling upgrade. You can also use the Opatch utility to check if the patch is a rolling patch:

```
$ opatch query –all <Patch_location> | grep rolling
```

If the patch is not a rolling patch, it will show the result "Patch is a rolling patch: false"; otherwise, it will show "Patch is a rolling patch: true."

You can use the OPatch utility to apply individual patches, not the patchset release to the RAC software. If the upgrade can be performed using the rolling fashion, follow these steps to perform the rolling upgrade:

1. Shut down the instance on one RAC node
2. Shut down the CRS stack on this RAC node
3. Apply the patch to the RAC home on that RAC node
4. Start the CRS stack on the RAC node
5. Start the RAC instance on the RAC node
6. Repeat steps 1 to 4 on each of the other RAC nodes in the cluster

There is a special type of interim patch or diagnostic patch. These patches contain a single shared library, and do not require shutting down the instance or relinking the Oracle binary. These patches are called online patches or hot patches. To determine whether a patch is an online patch, check if there is an online directory under the patch and if the README file has specified this patch to be online patchable. You can use the Opatch tool to apply an online patch without shutting down the Oracle instance that you are patching. For example, Patch 10188727 is an online patch, as shown in the patch directory:

```
$ cd <PATCH_TOP>/10188727
$ ls
etc/ files/ online/ README.txt
```

You also can query if the patch is an online patch by going to the patch direcory and running the following command:

```
$ opatch query -all online
```

If the patch is an online patch, you should see something like this in the result for this command: "Patch is an online patch: true." You should not confuse this result with the query result of a rolling patch result, `"Patch is a rolling patch: true."`

You should be aware that very few patches are online patches. Usually, online patches are used when a patch needs to be applied urgently before the database can be shut down. It is highly recommended that at the next database downtime the all-online patches should be rolled back and replaced with offline version of the patches. Refer to MOS note ID 761111.1 for all the best practices when using online patches.

For those patches that are not certified for the rolling upgrade, if you have a physical standby configuration for the database, you can use the Oracle Data Guard SQL apply feature and Oracle 11g Transient Logical standby feature

to implement the rolling database upgrade between the primary database and standby database and thus reduce database upgrade downtime. In this case, the database can be either a RAC or a non-RAC single-node database. Refer to MOS note ID 949322.1 for a detailed configuration of this method.

Oracle RAC One Node to Achieve HA

Oracle RAC One Node is a single-node RAC database. It provides an alternative way to protect the database against both unplanned and planned downtime. Unlike Oracle RAC Database with multiple database instances to provide an active-active cluster database solution, Oracle RAC One Node database is an active-passive cluster database. At any given time, only one database instance is running on one node of the cluster for the database. The database instance will be failed over to another node in the cluster in case of failure of the RAC node. This database instance can also be relocated to another node. This relocation is called online migration, as there is no database service downtime during the relocation. This online migration eliminates the planned downtime of maintenance.

Oracle RAC One Node is based on the same Grid Infrastructure as Oracle RAC Database. Oracle Clusterware in the Grid Infrastructure provides failover protection for Oracle RAC One Node. Since Oracle RAC One Node runs only one database instance, you can scale up the database server only by adding more CPU and memory resources to the server instead of scaling out by running multiple database instances. If the workloads expand beyond the capacity of a single server, you can easily upgrade the RAC One Node database to a fully functional multi-node Oracle RAC. Compared with Oracle RAC, Oracle RAC One Node has a significant advantage in its software license cost. Chapter 14 discusses Oracle RAC One Node technology in detail.

RAC Scalability

Oracle RAC offers an architecture that can potentially increase database capacity by scaling out the database across multiple server nodes. In this architecture, the multi-node cluster combines the CPU and memory computing resources of multiple RAC nodes to handle the workloads. Oracle cache fusion makes this scale-out solution possible by coordinating shared access to the database from multiple database instances. This coordination is managed through communication over the high-speed interconnect. One of the key components of the coordination is GCS data block transfer between database instances. Heavy access to the same data block by multiple database instances leads to high traffic of the data transfer over the interconnect. This can potentially cause interconnect congestion, which easily becomes a database performance bottleneck. The following considerations may help maintain RAC database scalability:

1. Segregate workloads to different RAC nodes to reduce the demand for sharing the same data block or data object by multiple RAC nodes. This segregation can also be done through application affinity or instance affinity. For example, for the Oracle E-Business application, we can assign each application module to a specific RAC instance so that all applications that access the same set of tables are from the same instance.
2. Reduce potential block transfers between instances. One way is to use a big cache value and NOORDER option for the sequence creation in a RAC database. This will ensure that each instance caches a separate range of sequence numbers and the sequence numbers are assigned out of order by the different instances. When these sequence numbers are used as the index key values, different key values of the index are inserted depending on the RAC instance in which the sequence number is generated. This creates instance affinity to the index Leaf blocks, and helps reduce pinging of the index Leaf blocks between instances.
3. Reduce the interconnect network latency by using a high bandwidth, high-speed network such as InfiniBand or 10GB Ethernet.

4. Constantly monitor the interconnect traffic and RAC cluster wait events. You can either monitor these cluster wait events on the Clusterware cache coherence page of Oracle Enterprise Manager, or check if there are any of the GCS-related wait events shown on the Top 5 Timed Events of AWR report.

Load Balancing Among RAC Instances

Another important feature related to RAC scalability is designed to distribute the workloads among all RAC instances for optimizing performance. This workload distribution occurs when a client connects to the RAC database for the first time or when the client connection is failed over from a failed instance to a surviving instance, in which case the load balancing works together with the failover feature. Oracle provides two kinds of load balancing: client-side load balancing and server-side load balancing.

Client-side load balancing is enabled by setting LOAD_BALANCE=yes in the client tnsnames.ora file:

```
KHDB =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = kr720n1-vip)(PORT = 1521))
  (ADDRESS = (PROTOCOL = TCP)(HOST = kr720n2-vip)(PORT = 1521))
 (LOAD_BALANCE =  yes)
  (CONNECT_DATA =
   (SERVICE_NAME = khdb.dblab.com)
   )
  )
 )
```

With LOAD_BALANCE enabled, Oracle Net chooses an address to connect based on load balancing characteristics from pmon rather than on sequential order. This order ensures the even distribution of the number of user sessions connecting to each database instance. In Oracle 11gR2, the list of addresses has been replaced with one SCAN entry. If you define SCAN with three IP address in the corporate DNS (Domain Name Service), client-side load balancing is moved to the DNS level among the three IPs for the SCAN. Chapter 9 gives more details about the 11gR2 SCAN configuration. Some old-version Oracle clients such as pre-11gR2 clients (11gR1, 10gR2, or older) may not be able to get the benefits of SCAN, as these clients will not be able to handle the three IPs of SCAN; instead, they may just connect to the first one. If the one that the client connects to fails, the client connection fails. Therefore, it may be better to use the old way by listing three VIP addresses of the SCAN IPs on the tnames.ora files.

Unlike the selection of the RAC node for the incoming user connection by client-side load balancing, in server-side load balancing the least-loaded RAC node is selected. Then, by using information from the Load Balancing Advisory, the best RAC instance that is currently provided to the service is selected for the user to connect. There is no need for any code change in the application side for server-side load balancing. However, the initialization parameter remote_listener needs to be set to enable listener connection load balancing. In 11gR2, the remote_listener is set to SCAN:PORT, as shown in the following example:

```
SQL> show parameter _listener

NAME                 TYPE    VALUE
-------------------- ------- ------------------------------
local_listener       string  (DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(
                             (PROTOCOL=TCP)(HOST=172.16.9.171)
                             (PORT=1521))))
remote_listener      string  kr720n-scan:1521
```

The remote_listener parameter is set by defult if you use DBCA to create the RAC database.

Flex Cluster to Increase Scalability

Oracle RAC 12c introduces Oracle Flex Cluster to improve cluster scalability. Before Oracle 12c, all the nodes of an Oracle RAC cluster were tightly coupled. This architecture is difficult to scale and manage as the number of nodes in the cluster increases beyond a certain level. One of the issues with this architecture is the number of interconnect links between the cluster nodes. In this architecture, each node is connected to every other node in the cluster. For an N-node cluster, the number of interconnect links is N *(N-1)/2; for a 100-node cluster, this number reaches 4,950. The Oracle 12c Flex Cluster reduces the number of network links between nodes by allowing loosely coupled Leaf Nodes and requiring a tightly coupled cluster only among a smaller number of Hub Nodes. In this Flex Cluster architecture, the Leaf Nodes connect only to the Hub Nodes that they are attached to, and there is no interconnect among the Leaf Nodes. This architecture significantly reduces the number of connections among the cluster nodes and makes the cluster more scalable and manageable.

Consolidating Database Services with Oracle RAC

In the traditional corporate computing model, one infrastructure is usually built for one application, with little or no resource sharing among applications. Not only does this model result in low efficiency and poor utilization of resources, it also makes it very difficult to reassign resources to adapt to the rapid pace of business change. Many systems have to preallocate large amounts of system resources in their capacity planning to cover peak demand and future growth. Today's IT departments, under increasing pressure to provide low-cost, flexible computing services, have to adapt to the idea that multiple applications services can be consolidated to share a pool of the computing resources: servers, networks, and storage. Grid computing originated from the concept of the electricity utility grid, and the recent Private Cloud is also based on this idea. These models have the following characteristics:

1. Consolidation: Many applications with different workloads are consolidated in the same infrastructure.
2. Dynamic resource sharing: All resources in the infrastructure are shared and can be dynamically reassigned or redistributed to applications services as needed.
3. High Availability: Applications within the shared infrastructure can be failed over or migrated across physical resources. This provides a virtualized infrastructure service that is independent of the physical hardware and also protects applications against unplanned system outages and planned system maintenance.
4. Scalability: Allows adding more physical resources to scale the infrastructure.

Consolidation and resource sharing dramatically increase resource utilization, and reduce hardware and system costs. There are cost savings both in capital expenses and operating expenses, as you not only need to purchase less hardware and software, but you can spend less on management and ongoing support costs. Consolidation and resource sharing also confer the benefits of high availability, flexibility, and scalability on application services.

Oracle RAC, along with Oracle Clusterware and ASM, provides the key technologies to implement this shared resource infrastructure for database consolidation.

- Oracle Clusterware and ASM provide infrastructure which consists of a pool of servers, along with storage and a network for database consolidation.
- Oracle Clusterware and RAC provide high availability and scalability for all databases that share this infrastructure.
- Oracle RAC features such as Instance Caging, Database Resource Manager, and Quality of Service enable the efficient use of shared resources by databases.
- The enhancement introduced by Oracle 12c Clusterware, like the policy-based approach for Clusterware management, allows for dynamic resource reallocation and prioritization of various applications' workloads consolidated in the shared cluster infrastructure.

Figure 1-5 shows an example of such a shared resources Grid Infrastructure based on a 16-node Oracle 11gR2 RAC that consolidates more than 100 Oracle E-Business suite databases running on various versions of Oracle Database from 10gR2 to 11gR2.

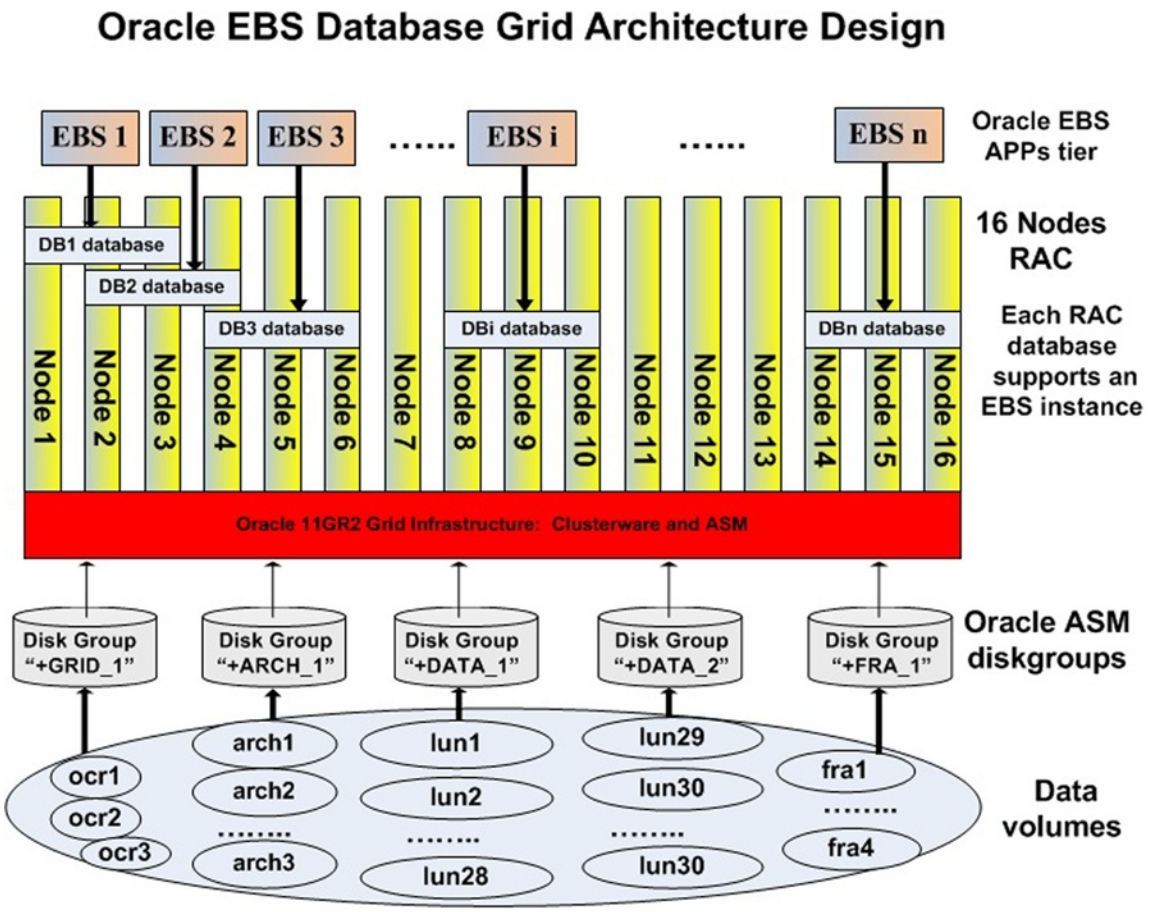


Figure 1-5. *The shared infrastructure for database consolidation*

- Each database may run on one to three RAC nodes with the capability to expand to more nodes.
- Each database instance can be failed over or relocate to other nodes.
- Each RAC node may run multiple RAC database instances from different databases using different versions of Oracle RAC Database binaries.
- Multiple versions (10g2-11gR2) of Oracle RAC Database are based on a single Oracle 11gR2 Grid Infrastructure.
- It is required to ping CRS on each of 16 nodes for pre-11gR2 databases

MOS note ID 1439551.1 "Oracle (RAC) Database Consolidation Guidelines for Environments Using Mixed Database Versions" discusses some principles and guidelines for how to consolidate multiple versions of Oracle Database on the same Oracle RAC and Oracle Grid Infrastructure 11g release 2 or higher. The discussions include considerations for including pre-11g Release 2 databases and how to set up the number of the LMS processes and

how to set the CPU count to manage CPU resources for those pre-11g R2 Database instances. The related whitepaper on the Database consolidation topic is Oracle whitepaper "Best Practices for Database Consolidation in Private Clouds," which can be found at the following URL: www.oracle.com/technetwork/database/focus-areas/database-cloud/database-cons-best-practices-1561461.pdf.

As one of the most important new features introduced in Oracle 12c, the pluggable database provides a better solution to consolidate multiple database services. In Chapter 4, we explain in detail how the pluggable database feature works in the Oracle RAC 12c environment and how to implement the multitenancy of database services by consolidating multiple pluggable databases into a single container database running on Oracle RAC 12c.

Considerations for Deploying RAC

As we have shown in this chapter, RAC embodies a great technology solution for achieving HA and scalability of Oracle database services. However, this solution itself has a complex hardware and software technology stack. Before an IT organization decides to adapt Oracle RAC technology for its database architecture, it should be aware of the advantages and potential disadvantages of the Oracle RAC technology, and its implications for the organization's business goals. This will help to justify the adoption of Oracle RAC for the business. The next section highlights some related considerations that will help you decide whether or not Oracle RAC should be used as the database architecture.

Cost of Ownership

One of the possible reasons that IT departments are looking at Oracle RAC is to reduce the cost of ownership of the database infrastructure. This cost saving is relative, depending on what you are comparing. The cost of Oracle RAC implementation includes three parts: hardware infrastructure cost, Oracle software cost, and management cost. The hardware stack consists of multiple servers, redundant networks, and shared storage. The software cost mainly includes Oracle RAC license and the Oracle Database license. For Oracle Database Enterprise Edtion, the Oracle RAC license is separate from Oracle Database license, while for Oracle Database standard edition, the Oracle Database license already includes the Oracle RAC license which you don't have to pay for separately.

One of the limitations of Oracle Standard Edition is that the total number of CPU sockets of all the servers in the cluster can not go beyond 4. A **CPU socket** is a connection that allows a computer processor to be connected to a motherboard. A CPU socket can have multilple CPU cores. For example, a Dell R820 server has four CPU sockets while a Dell R720 server has two sockets. Since each socket can have 8 cores, an R820 server can have up to 4 * 8 = 32 CPU cores, and a Dell R720 server can have up to 16 CPU cores. Using Oracle Standard Edition with a maximum capacity of 4 CPU sockets, you can make a two-node Oracle RAC cluster with Dell R720 servers, but only a one-node cluster with a Dell R820 server. For Oracle Enterprise Edition, the RAC license can be based on the total number of processors. This is based on the total cores of the servers in the cluster. For example, for a two-node Oracle RAC configuration using Dell R720s with 8 core CPU sockets, the total number of the CPU cores can be 2 * 2 * 8 = 32.

Management staff cost is related to the cost of training and attracting individuals with the skills needed (system admins, network admins, and DBAs) to manage it. The hardware and software costs include the initial purchase cost as well as the ongoing support cost. Although this cost is higher than a simple database solution like a single-node MS SQL server, the RAC solution is cheaper than typical complex mission-critical databases running on big SMP servers, as Oracle RAC is mainly implemented on Linux and industry-standard low-cost commodity hardware. In the last decade, these industry-standard servers running Linux have become much cheaper, and offer a powerful and reliable solution widely accepted for enterprise systems.

Another cost-saving factor is that Oracle RAC can be implemented as a shared resource pool to consolidate many databases. This can significantly reduce the costs of hardware, software, and management by reducing the number of systems. In the Oracle E-Business database consolidation example mentioned in the last section, 100 databases were consolidated onto a 16-node RAC. The number of database servers was reduced from 100 or more to 16. The reduction led to huge savings in hardware, software, and management. As already mentioned, long-term operating costs are also cut by reducing the need for support, maintenance, and even powering and cooling 100 systems in a data center for the entire life cycle of the environment. For full details of this example, refer to my technical presentation at Oracle OpenWorld: http://kyuoracleblog.files.wordpress.com/2012/09/ebs_dbs_on_11gr2_grid_oow2011_session8945.pdf.

High Availability Considerations

Oracle RAC provides HA of the database service by reducing unplanned and planned downtime caused by server failure. But RAC itself doesn't protect the database against other failures, such as storage failure, data corruption, network failure, human operation error, or even data center failure. To provide complete protection against these failures, additional measures need to be taken. Oracle MAA (Maximal Availability Architecture) lists the guidelines and related Oracle technologies needed to protect databases against those failures.

During the deployment of RAC, it is critical to follow HA practices to ensure the stability of the RAC. The most important hardware components that Oracle RAC relies on are the private network and shared storage. The private network should be based on a redundant network with two dedicated switches. Chapter 9 discusses the RAC network in detail. The shared storage access should be based on multiple I/O paths, and the storage disk drives should be set up with a RAID configuration and Oracle ASM disk mirroring to ensure redundancy. Chapter 5 discusses storage best practices in detail.

In theory, Oracle RAC protects the database service against failure of up to N-1 servers (where N is the total number of servers). In reality, if all of the N-1 servers fail, the workloads of the entire clusterware will be on the only surviving node, and the performance will definitely suffer unless each server leaves N-1/N headroom. For example, for a four-node RAC, leaving 3/4 (75%) headroom would not be realistic. A realistic approach is to ensure that each server in the cluster can handle the failed-over workload in case of single server failure. This requires each server to leave only 1/N headroom. And the bigger N is, the less headroom is needed. The worst case is a two-node RAC, where each server needs to reserve 1/2 (50%) headroom. For a four-node RAC, only 1/4 = 25% headroom is needed.

■ **Note** CPU headroom is the CPU resource that we have to leave unused in case of server failure. The less headroom, the better resource utilization on each node.

Scalability Considerations

Oracle RAC provides database scalability. With the addition of each extra RAC node, the cluster is expected to increase database performance capability: handling larger workloads or more concurrent users, performing more TPS (transactions per second) for OLTP, or reducing the average transaction/query response time. However, many RAC databases may not show linear scalability when adding more RAC nodes. This is because there are many other factors that are related to database scalability:

1. Poor database design and poorly tuned SQL queries can lead to very costly query plans that may kill database throughput and significantly increase query response time. Poorly tuned queries will run just as badly (or even worse) in RAC compared to a single-node database.

2. There may be quite costly performance overhead caused by Oracle cache fusion, and excessive wait time on data blocks transferring on interconnects between RAC nodes during query executions and database transactions. These wait events are called cluster wait events. Cache fusion overhead and cluster wait events may increase when multiple RAC instances access the same data blocks more frequently. A higher number of RAC nodes also contributes to cluster waits and slows down the interconnect. The number of RAC nodes is limited by the bandwidth of the interconnect network, which is less of an issue with the introduction of high-speed networks such as InfiniBand and 10-40GB Ethernet.

3. In some database environments with I/O-intensive workloads, most performance bottlenecks are on storage I/O with lower CPU utilization. Such environments will not scale well just by adding more RAC nodes. Therefore, it is important to understand the workload characteristics and potential performance bottlenecks before we opt for the scalable solution. Storage performance capacity is measured in IOPS (I/O operations per second) for OLTP workloads and throughput MB/second for DSS workloads. Since the speed of hard disks is limited by physics (drive seek time and latency), they tend to impose an upper limit on IOPS and throughput. One way to scale storage performance is to add more disk drives and stripe the data files with either RAID or Oracle ASM striping. Another option is to move frequently accessed data (hot data) to solid disk drives (SSDs). SSDs provide much higher IOPS, especially for the random small I/O operations which dominate OLTP workloads, as SSDs have no moving parts and hence no mechanical delays. Using SSDs is a very viable option to scale storage IOPS performance for OLTP workloads. For DSS workloads, one option is based on the building block concept. Each building block is composed of a RAC node plus additional storage and network based on the balance between CPU processing power and storage throughput for a DSS/Data warehouse–type workload. Scalability is based on the building block instead of just a server.

RAC or Not

When IT organizations need to decide whether to deploy an Oracle RAC as their database architecture, IT architects and DBAs need to make decisions based on many factors.

1. The High availability SLA: How much database downtime is acceptable for both unplanned and planned downtime? Without RAC, planned downtime for hardware and software maintenance may vary from a few minutes to as much as a few hours. And the unplanned time for hardware and software problems can also vary from minutes to hours. If a database server is completely lost, it will take a longer time to rebuild it, although some downtimes, like the complete loss of the server, may occur only rarely, Are these potential downtimes acceptable to the business according to the SLA? If not, can downtime prevention justify the cost of the RAC deployment? And furthermore, a loss of the entire database system including the storage may take hours or days to recover from the backup. Does this justify a Disaster Recovery (DR) solution which consists of a completely duplicated system in another data center? Some mission-critical databases may be equipped with the combination of RAC and Data Guard DR solution to protect the database from server failure as well as storage and site failure. However, the cost and technical complexity need to be justified by business needs.

2. Scalability Requirement: What kind of workloads are there and where is the performance bottleneck: CPU intensive and/or storage I/O intensive? If there is no need to scale out CPU/memory resources or if we can scale up by adding additional CPUs or memory, Oracle RAC One Node (instead of multiple RAC RAC) may be a good way of providing HA without the need to pay for an RAC license. RAC One Node also has the flexibility to be easily upgraded to a full RAC solution any time there is a need to scale out to multi-node RAC in the future.

3. Database Consolidation Requirements: If the organization has a business requirement to consolidate many database services together, a multi-node RAC can offer significant advantages in terms of cost of ownership wjile providing HA and scalability to all the databases.

4. If the organization decides to deploy the RAC solution, it should fulfil the hardware requirements and follow configuration and management best practices to ensure that the RAC provides all the potential advantages.

5. Many companies have carried out successful migration of their mission-critical databases from big SMP Unix machines to multi-node Oracle RAC clusters based on lower-cost industry-standard commodity X86-64 servers running Linux. In the last decade, these servers have advanced significantly in terms of reliability as well as processing power. Practical experience has shown that the new architecture can provide a highly available and scalable infrastructure for enterprise-level applications.

Summary

In this chaper, you received an overview of the architecture and components of Oracle RAC. I explained how Oracle RAC provides high availability and scalability database solutions. I also demonstrated how to use Oracle RAC to build a Grid computing infrasturecture to consolidate many database services. In conclusion, I reviewed what factors IT departments should consider to justify an Oracle RAC deployment.

In this introductory chapter, it is worthwhile listing some invaluable MOS notes that introduce some very useful tools and utilties for Oracle RAC. The tools and best practices mentioned in these MOS notes can be very beneficial for the study of Oracle RAC technology.

RACcheck—A RAC Configuration Audit Tool (MOS doc ID 1268927.1) is about a RACcheck tool that can be used to audit various important configuration settings for Oracle RAC, Clusterware, ASM, and the Grid Infrastructure environment.

RAC and Oracle Clusterware Best Practices and Starter Kit (Platform Independent) (MOS doc ID 810394.1). This document provides generic and platform-independent best practices for implementing, upgrading, and maintaining an Oracle RAC system. It also lists links to platform-specific documents.

ODAchk—- Oracle Database ApplianceODA Configuration Audit Tool (MOS doc ID 1485630.1). This document introduces the ODAchk tool, which automates the assessment of ODA systems for known configuration problems and best practices.

TFA Collector—The Preferred Tool for Automatic or ADHOC Diagnostic Gathering Across All Cluster Nodes (MOS doc ID 1513912.1). This document introduces a new tool called TFA Collector (aka TFA) a diagnostic collection utility for Oracle Clusterware/Grid Infrastructure and RAC systems.

CHAPTER 2

Clusterware Stack Management and Troubleshooting

by Syed Jaffar Hussain, Kai Yu

In Chapter 1, we mentioned that the Oracle RAC cluster database environment requires cluster manager software ("Clusterware") that is tightly integrated with the operating system (OS) to provide the cluster management functions that enable the Oracle database in the cluster environment.

Oracle Clusterware was originally introduced in Oracle 9i on Linux with the original name Oracle Clusterware Management Service. Cluster Ready Service (CRS) as a generic cluster manager was introduced in Oracle 10.1 for all platforms and was renamed to today's name, Oracle Clusterware, in Oracle 10.2. Since Oracle 10g, Oracle Clusterware has been the required component for Oracle RAC. On Linux and Windows systems, Oracle Clusterware is the only clusterware we need to run Oracle RAC, while on Unix, Oracle Clusterware can be combined with third-party clusterware such as Sun Cluster and Veritas Cluster Manager.

Oracle Clusterware combines a group of servers into a cluster environment by enabling communication between the servers so that they work together as a single logical server. Oracle Clusterware serves as the foundation of the Oracle RAC database by managing its resources. These resources include Oracle ASM instances, database instances, Oracle databases, virtual IPs (VIPs), the Single Client Access Name (SCAN), SCAN listeners, Oracle Notification Service (ONS), and the Oracle Net listener. Oracle Clusterware is responsible for startup and failover for the resources. Because Oracle Clusterware plays such a key role in the high availability and scalability of the RAC database, the system administrator and the database administrator should pay careful attention to its configuration and management.

This chapter describes the architecture and complex technical stack of Oracle Clusterware and explains how those components work. The chapter also describes configuration best practices and explains how to manage and troubleshoot the clusterware stack. The chapter assumes the latest version of Oracle Clusterware 12cR1.

The following topics will be covered in this chapter:

- Oracle Clusterware 12cR1 and its components
- Clusterware startup sequence
- Clusterware management
- Troubleshooting cluster stack startup failure
- CRS logs and directory structure
- RACcheck, diagcollection.sh, and oratop
- Debugging and tracing CRS components
- RAC database hang analysis

Clusterware 12cR1 and Its Components

Before Oracle 11gR2, Oracle Clusterware was a distinct product installed in a home directory separate from Oracle ASM and Oracle RAC database. Like Oracle 11gR2, in a standard 12cR1 cluster, Oracle Clusterware and Oracle ASM are combined into a product called Grid Infrastructure and installed together as parts of the Grid Infrastructure to a single home directory. In Unix or Linux environments, some part of the Grid Infrastructure installation is owned by the root user and the rest is owned by special user grid other than the owner of the Oracle database software oracle. The grid user also owns the Oracle ASM instance.

Only one version of Oracle Clusterware can be active at a time in the cluster, no matter how many different versions of Oracle Clusterware are installed on the cluster. The clusterware version has to be the same as the Oracle Database version or higher. Oracle 12cR1 Clusterware supports all the RAC Database versions ranging from 10gR1 to 12cR1. ASM is always the same version as Oracle Clusterware and can support Oracle Database versions ranging from 10gR1 to 12cR1.

Oracle 12cR1 introduced Oracle Flex Cluster and Flex ASM. The architecture of Oracle Clusterware and Oracle ASM is different from the standard 12cR1 cluster. We will discuss Oracle Flex Cluster and Flex ASM in Chapter 5. This chapter will focus on the standard 12cR1 cluster.

Storage Components of Oracle Clusterware

Oracle Clusterware consists of a storage structure and a set of processes running on each cluster node. The storage structure consists of two pieces of shared storage: the Oracle Cluster Registry (`OCR`) and voting disk (VD) plus two local files, the Oracle Local Registry (`OLR`) and the Grid Plug and Play (`GPnP`) profile.

`OCR` is used to store the cluster configuration details. It stores the information about the resources that Oracle Clusterware controls. The resources include the Oracle RAC database and instances, listeners, and virtual IPs (`VIPs`) such as SCAN VIPs and local VIPs.

The voting disk (VD) stores the cluster membership information. Oracle Clusterware uses the VD to determine which nodes are members of a cluster. Oracle Cluster Synchronization Service daemon (`OCSSD`) on each cluster node updates the VD with the current status of the node every second. The VD is used to determine which RAC nodes are still in the cluster should the interconnect heartbeat between the RAC nodes fail.

Both OCR and VD have to be stored in a shared storage that is accessible to all the servers in the cluster. They can be stored in raw devices for 10g Clusterware or in block devices in 11gR1 Clusterware. With 11g R2 and 12cR1 they should be stored in an ASM disk group or a cluster file system for a freshly installed configuration. They are allowed to be kept in raw devices and block devices if the Clusterware was just being upgraded from 10g or 11gR1 to 11gR2; however, it is recommended that they should be migrated to an ASM disk group or a cluster file system soon after the upgrade. If you want to upgrade your Clusterware and Database stored in raw devices or block devices to Oracle Clusterware 12c and Oracle Database 12c, you must move the database and OCR/VDs to ASM first before you do the upgrade, as Oracle 12c no longer supports the use of raw device or block storage. To avoid single-point-of failure, Oracle recommends that you should have multiple OCRs, and you can have up to five OCRs. Also, you should have at least three VDs, always keeping an odd number of the VDs. On Linux, the `/etc/oracle/ocr.loc` file records the OCR location:

```
$ cat  /etc/oracle/ocr.loc
ocrconfig_loc=+VOCR
local_only=FALSE
```

In addition, you can use the following command to find the VD location:

```
$ ./crsctl query css votedisk
```

The Oracle ASM disk group is the recommended primary storage option for OCR and VD. Chapter 5 includes a detailed discussion of storing OCR and VDs in an ASM disk group.

Two files of Oracle Clusterware (OLR) and GPnP profile are stored in the grid home of the local file system of each RAC node. OLR is the OCR's local version, and it stores the metadata for the local node and is managed by the Oracle High Availability Services daemon (OHASD). OLR stores less information than OCR, but OLR can provide this metadata directly from the local storage without the need to access the OCR stored in an ASM disk group. One OLR is configured for each node, and the default location is in $GIHOME/cdata/<hostname>.olr. The location is also recorded in /etc/oracle/olr.loc, or you can check it through the ocrcheck command:

```
$ cat /etc/oracle/olr.loc
olrconfig_loc=/u01/app/12.1.0/grid/cdata/knewracn1.olr
crs_home=/u01/app/12.1.0/grid

$ ocrcheck -local -config
Oracle Local Registry configuration is :
  Device/File Name         : /u01/app/12.1.0/grid/cdata/knewracn1.olr
```

The GPnP profile records a lot of important information about the cluster, such as the network profile and the VD. The information stored in the GPnP profile is used when adding a node to a cluster. Figure 2-1 shows an example of the GPnP profile. This file default is stored in $GRID_HOME/gpnp/<hostname>/profiles/peer/profile.xml.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <gpnp:GPnP-Profile Version="1.0" xmlns="http://www.grid-pnp.org/2005/11/gpnp-profile" xmlns:gpnp="http://www.grid-
  pnp.org/2005/11/gpnp-profile" xmlns:orcl="http://www.oracle.com/gpnp/2005/11/gpnp-profile"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.grid-pnp.org/2005/11/gpnp-profile
  gpnp-profile.xsd" ProfileSequence="7" ClusterUId="0a73b0dbe0fcff2bff5b7aed352821b4" ClusterName="knewrac" PALocation="">
  - <gpnp:Network-Profile>
    - <gpnp:HostNetwork id="gen" HostName="*">
        <gpnp:Network id="net1" IP="172.16.0.0" Adapter="eth0" Use="public" />
        <gpnp:Network id="net2" IP="192.168.9.0" Adapter="eth1" Use="asm,cluster_interconnect" />
      </gpnp:HostNetwork>
    </gpnp:Network-Profile>
    <orcl:CSS-Profile id="css" DiscoveryString="+asm" LeaseDuration="400" />
    <orcl:ASM-Profile id="asm" DiscoveryString="" SPFile="+DATA1/knewrac/ASMPARAMETERFILE/registry.253.807834851"
      Mode="remote" />
    <orcl:BC-BigCluster id="bc" DiscoveryVIP="172.16.150.9" />
  - <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    - <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
      - <ds:Reference URI="">
        - <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
          - <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
              <InclusiveNamespaces xmlns="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="gpnp orcl xsi" />
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
          <ds:DigestValue>GtFWI6t3M9kHIyEr4G6SiuXMAv4=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
```

Figure 2-1. *GPnP profile*

Clusterware Software Stack

Beginning with Oracle 11gR2, Oracle redesigned Oracle Clusterware into two software stacks: the High Availability Service stack and CRS stack. Each of these stacks consists of several background processes. The processes of these two stacks facilitate the Clusterware. Figure 2-2 shows the processes of the two stacks of Oracle 12cR1 Clusterware.

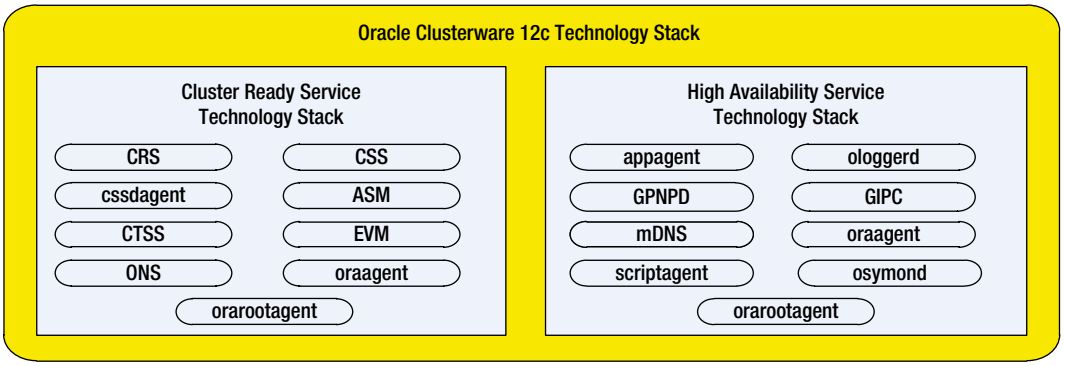


Figure 2-2. *Oracle Clusterware 12cR1 stack*

High Availability Cluster Service Stack

The High Availability Cluster Service stack is the lower stack of the Oracle Clusterware. It is based on the Oracle High Availability Service (`OHAS`) daemon. The OAHS is responsible for starting all other clusterware processes. In the next section, we will discuss the details of the clusterware sequences.

OHAS uses and maintains the information in OLR. The High Availability Cluster Service stack consists of the following daemons and services:

GPnP daemon (`GPnPD`): This daemon accesses and maintains the GPnP profile and ensures that all the nodes have the current profile. When OCR is stored in an ASM diskgroup, during the initial startup of the clusterware, OCR is not available as the ASM is not available; the GPnP profile contains enough information to start the Clusterware.

Oracle Grid Naming Service (`GNS`): This process provides the name resolutions with the cluster. With 12cR1, GNS can be used for multiple clusters in contrast to the single-cluster version.

Grid Interprocess Communication (`GIPC`): This daemon supports Grid Infrastructure communication by enabling Redundant Interconnect Usage.

Multicast Domain Name Service (`mDNS`): This daemon works with GNS to perform name resolution.

This stack also includes the System Monitor Service daemon (`osysmond`) and Cluster Logger Service daemon (`ologgerd`).

The CRS Stack

The CRS stack is an upper-level stack of the Oracle Clusterware which requires the support of the services of the lower High Availability Cluster Service stack. The CRS stack includes the following daemons and services:

CRS: This service is primarily responsible for managing high availability operations. The CRS daemon (CRSD) manages the cluster resource's start, stop monitor, and failover operations. CRS maintains the configuration information in OCR. If the cluster has an Oracle RAC database, the resources managed by CRS include the Oracle database and its instances, listener, ASM instance, VIPs, and so on. This service runs as the `crs.bin` process on Linux/Unix and `OracleOHService` on Windows.

CSS: This service manages and monitors the node membership in the cluster and updates the node status information in VD. This service runs as the `ocssd.bin` process on Linux/Unix and `OracleOHService (ocssd.exe)` on Windows.

CSS Agent: This process monitors, starts, and stops the CSS. This service runs as the `cssdagent` process on Linux/Unix and `cssdagent.exe` on Windows.

CSS Monitor: This process works with the `cssdagent` process to provide the I/O fencing to ensure data integrity by rebooting the RAC node in case there is an issue with the `ocssd.bin` process, a CPU starvation, or an OS locked up. This service runs as `cssdmonitor` on Linux/Unix or `cssdmonitor.exe` on Windows. Both `cssdagent` and `cssdmonitor` are the new features started in 11gR2 that replace the previous Oracle Process Monitor daemon (`oprocd`) in 11gR1.

Cluster Time Synchronization Service (CTSS): A new daemon process introduced with 11gR2, which handles the time synchronization among all the nodes in the cluster. You can use the OS's Network Time Protocol (NTP) service to synchronize the time. Or, if you disable NTP service, CTSS will provide the time synchronization service. This service runs as the `octssd.bin` process on Linux/Unix or `octssd.exe` on Windows.

Event Management (EVM): This background process publishes events to all the members of the cluster. On Linux/Unix, the process name is `evmd.bin`, and on Windows, it is `evmd.exe`.

ONS: This is the publish and subscribe service that communicates Fast Application Notification (FAN) events. This service is the `ons` process on Linux/Unix and `ons.exe on Windows.`

Oracle ASM: Provides the volume manager and shared storage management for Oracle Clusterware and Oracle Database.

Clusterware agent processes: Oracle Agent (`oraagent`) and Oracle Root Agent (`orarootagent`). The `oraagent` agent is responsible for managing all Oracle-owned `ohasd` resources. The `orarootagent` is the agent responsible for managing all root-owned `ohasd` resources.

Clusterware Startup Sequence

Oracle Clusterware is started up automatically when the RAC node starts. This startup process runs through several levels. Figure 2-3 shows the multiple-level startup sequences to start the entire Grid Infrastructure stack plus the resources that Clusterware manages.

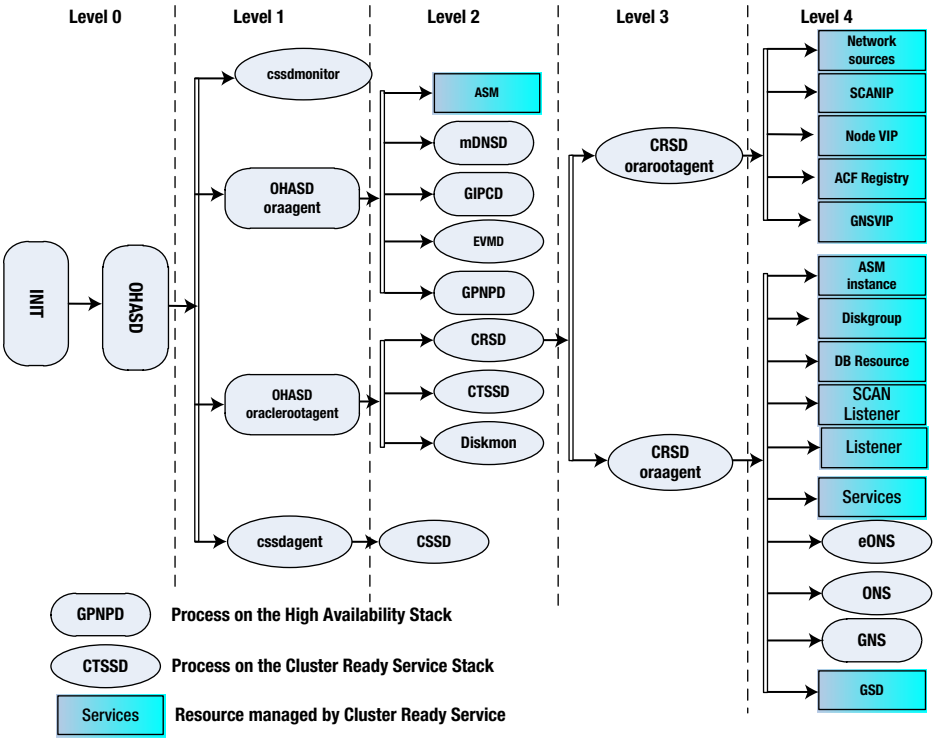


Figure 2-3. *Startup sequence of 12cR1 Clusterware processes*

Level 0: The OS automatically starts Clusterware through the OS's `init` process. The `init` process spawns only one `init.ohasd`, which in turn starts the OHASD process. This is configured in the `/etc/inittab` file:

```
$cat /etc/inittab|grep init.d | grep -v grep
h1:35:respawn:/etc/init.d/init.ohasd run >/dev/null 2>&1 </dev/null
Oracle Linux 6.x and Red Hat Linux 6.x have deprecated inittab. init.ohasd is configured in startup
in /etc/init/oracle-ohasd.conf:
$ cat /etc/init/oracle-ohasd.conf
......

start on runlevel [35]
stop  on runlevel [!35]
respawn
exec /etc/init.d/init.ohasd run >/dev/null 2>&1 </dev/null
This starts up "init.ohasd run", which in turn starts up the ohasd.bin background process:
$ ps -ef | grep ohasd | grep -v grep
root      4056     1  1 Feb19 ?        01:54:34 /u01/app/12.1.0/grid/bin/ohasd.bin reboot
root     22715    1  0 Feb19 ?        00:00:00 /bin/sh /etc/init.d/init.ohasd run
```

Once OHASD is started on Level 0, OHASD is responsible for starting the rest of the Clusterware and the resources that Clusterware manages directly or indirectly through Levels 1-4. The following discussion shows the four levels of cluster startup sequence shown in the preceding Figure 2-3.

Level 1: OHASD directly spawns four agent processes:

- `cssdmonitor`: CSS Monitor
- `OHASD orarootagent`: High Availability Service stack Oracle root agent
- `OHASD oraagent`: High Availability Service stack Oracle agent
- `cssdagent`: CSS Agent

Level 2: On this level, `OHASD oraagent` spawns five processes:

- `mDNSD`: mDNS daemon process
- `GIPCD`: Grid Interprocess Communication
- `GPnPD`: GPnP profile daemon
- `EVMD`: Event Monitor daemon
- `ASM`: Resource for monitoring ASM instances

Then, `OHASD oraclerootagent` spawns the following processes:

- `CRSD`: CRS daemon
- `CTSSD`: CTSS daemon
- `Diskmon`: Disk Monitor daemon (Exadata Storage Server storage)
- `ACFS`: (ASM Cluster File System) Drivers

Next, the `cssdagent` starts the `CSSD` (`CSS` daemon) process.

Level 3: The CRSD spawns two CRSD agents: `CRSD orarootagent` and `CRSD oracleagent`.
Level 4: On this level, the `CRSD orarootagent` is responsible for starting the following resources:

- Network resource: for the public network
- SCAN VIPs
- Node VIPs: VIPs for each node
- ACFS Registry
- GNS VIP: VIP for GNS if you use the GNS option

Then, the `CRSD orarootagent` is responsible for starting the rest of the resources as follows:

- ASM Resource: ASM Instance(s) resource
- Diskgroup: Used for managing/monitoring ASM diskgroups.
- DB Resource: Used for monitoring and managing the DB and instances
- SCAN listener: Listener for SCAN listening on SCAN VIP
- SCAN VIP: Single Client Access Name VIP
- Listener: Node listener listening on the Node VIP
- Services: Database services
- ONS
- eONS: Enhanced ONS
- GSD: For 9i backward compatibility
- GNS (optional): performs name resolution

ASM and Clusterware: Which One is Started First?

If you have used Oracle RAC 10g and 11gR1, you might remember that the Oracle Clusterware stack has to be up before the ASM instance starts on the node. Because 11gR2, OCR, and VD also can be stored in ASM, the million-dollar question in everyone's mind is, "Which one is started first?" This section will answer that interesting question.

The Clusterware startup sequence that we just discussed gives the solution: ASM is a part of the CRS of the Clusterware and it is started at Level 3 after the high availability stack is started and before CRSD is started. Then, the question is, "How does the Clusterware get the stored cluster configuration and the clusterware membership information, which are normally stored in OCR and VD, respectively, without starting an ASM instance?" The answer is that during the startup of the high availability stack, the Oracle Clusterware gets the clusterware configuration from OLR and the GPnP profile instead of from OCR. Because these two components are stored in the $GRID_HOME in the local disk, the ASM instance and ASM diskgroup are not needed for the startup of the high availability stack. Oracle Clusterware also doesn't rely on an ASM instance to access the VD. The location of the VD file is in the ASM disk header. We can see the location information with the following command:

```
$  kfed read /dev/dm-8 | grep -E 'vfstart|vfend'
kfdhdb.vfstart:                      352 ; 0x0ec: 0x00000160
kfdhdb.vfend:                        384 ; 0x0f0: 0x00000180
```

The `kfdhdb.vfstart` is the begin AU offset of the VD file, and the `kfdhdb.vfend` indicates the end AU offset of the VD file. Oracle Clusterware uses the values of `kfdhdb.vfstart` and `kfdhdb.vfend` to locate the VD file.

In this example, /dev/dm-8 is the disk for the ASM disk group VOCR which stores the VD file, as shown with running the following command:

```
$ crsctl query css votedisk
##  STATE    File Universal Id               File Name Disk group
--  -----    -----------------               --------- ---------
 1. ONLINE   7141f13f99734febbf94c73148c35a85 (/dev/dm-8) [VOCR]
                                              Located 1 VD(s).
```

Clusterware Management

The Grid Infrastructure Universal Installer takes care of the installation and configuration of the Oracle Clusterware and the ASM instance. After this installation, the Clusterware and ASM get restarted automatically every time when the server starts. Most times, this entire stack works well without need for a lot of manual intervention. However, as the most important infrastructure for Oracle RAC, this stack does need some proper management and ongoing maintenance work. Oracle Clusterware provides several tools, utilities, and log files for a Clusterware admin to perform management, troubleshooting, and diagnostic work. This section will discuss tools and Clusterware management, and the next few sections will discuss Clusterware troubleshooting and diagnosis.

Clusterware Management Tools and Utilities

Oracle provides a set of tools and utilities that can be used for Oracle Grid Infrastructure management. The most commonly used tool is the Clusterware control utility `crsctl`, which is a command-line tool for managing Oracle Clusterware. Oracle Clusterware 11gR2 has added to `crsctl` the cluster-aware commands that allow you to perform CRS check, start, and stop operations of the clusterware from any node. Use `crsctl -help` to print all the command Help with `crsctl`.

```
$ crsctl -help
Usage: crsctl add       - add a resource, type, or other entity
crsctl backup    - back up voting disk for CSS
crsctl check     - check a service, resource, or other entity
crsctlconfig    - output autostart configuration
crsctl debug     - obtain or modify debug state
crsctl delete    - delete a resource, type, or other entity
crsctl disable   - disable autostart
crsctl discover  - discover DHCP server
crsctl enable    - enable autostart
crsctleval      - evaluate operations on resource or other entity without performing them
crsctl get       - get an entity value
crsctlgetperm   - get entity permissions
crsctllsmodules - list debug modules
crsctl modify    - modify a resource, type, or other entity
crsctl query     - query service state
crsctl pin       - pin the nodes in the nodelist
crsctl relocate  - relocate a resource, server, or other entity
crsctl replace   - replace the location of voting files
crsctl release   - release a DHCP lease
crsctl request   - request a DHCP lease or an action entrypoint
crsctlsetperm   - set entity permissions
crsctl set       - set an entity value
```

```
crsctl start     - start a resource, server, or other entity
crsctl status    - get status of a resource or other entity
crsctl stop      - stop a resource, server, or other entity
crsctl unpin     - unpin the nodes in the nodelist
crsctl unset     - unset a entity value, restoring its default
```

You can get the detailed syntax of a specific command, such as `crsctl status -help`. Starting with 11gR2, `crsctl` commands are used to replace a few deprecated `crs_*` commands, such as `crs_start, crs_stat,` and `crs_stop`. In the following sections, we discuss the management tasks in correlation with the corresponding `crsctl` commands.

Another set of command-line tools are based on the `srvctl` utility. These commands are used to manage the Oracle resources managed by the Clusterware.

A `srvctl` command consists of four parts:

```
$ srvctl <command> <object> [<options>]
```

The command part specifies the operation of this command. The object part specifies the resource where this operation will be executed. You can get Help with the detailed syntax of the `srvctl` by running the `srvctl` Help command. For detailed Help on each command and object and its options for use, run the following commands:

```
$ srvctl <command> -h or
$ srvctl <command> <object> -h
```

There are also other utilities:

- `oifcfg` is a command-line tool that can be used to configure network interfaces.
- `ocrconfig` is a command-line tool that can be used to administer the OCR and OLR.
- `ocrcheck` is the OCR Check tool to check the state of the OCR.
- `ocrdump` is the Oracle Clusterware Registry Dump tool that can be used to dump the contents of OCR.
- Oracle Enterprise Manager Database Control 11g and Enterprise Manager Grid control 11g and 12c can be used to manage the Oracle Clusterware environment.

Start Up and Stop Clusterware

As we discussed in the previous section, through the OS `init` process, Oracle Clusterware is automatically started up when the OS starts. The clusterware can also be manually started and stopped by using the `crsctl` utility.

The `crsctl` utility provides the commands to start up the Oracle Clusterware manually:

Start the Clusterware stack on all servers in the cluster or on one or more named server in the cluster:

```
$ crsctl start cluster [-all | - n server1[,..]]
```

For example:

```
$crsctl start cluster –all
$ crsctl start cluster –n k2r720n1
```

Start the Oracle High Availability Services daemon (OHASD) and the Clusterware service stack together on the local server only:

```
$crsctl start crs
```

Both of these two crsctl startup commands require the root privilege on Linux/Unix to run. The 'crsctl start crs' command will fail if OHASD is already started.

The crsctl utility also provides similar commands to stop the Oracle Clusterware manually. It also requires root privilege on Linux/Unix to stop the clusterware manually.

The following command stops the clusterware stack on the local node, or all nodes, or specified local or remote nodes. Without the [-f] option, this command stops the resources gracefully, and with the [-f] option, the command forces the Oracle Clusterware stack to stop, along with the resources that Oracle Clusteware manages.

```
$ crsctl stop cluster [-all | -n server_name [...]] [-f]
```

The following command stops the Oracle High Availability service on the local server. Use the [-f] option to force any resources to stop, as well as to stop the Oracle High Availability service:

```
$ crsctl stop crs [-f]
```

Managing Oracle Clusterware

You can use the following command to check the cluster status:

```
$ crsctl check cluster {-all}

CRS-4537: Cluster Ready Services is online
CRS-4529: Cluster Synchronization Services is online
CRS-4533: Event Manager is online
```

Check the CRS status with the following command:

```
$ crsctl check crs

CRS-4638: Oracle High Availability Services is online
CRS-4537: Cluster Ready Services is online
CRS-4529: Cluster Synchronization Services is online
CRS-4533: Event Manager is online
```

Check the OHASD status:

```
$GRID_HOME/bin/crsctl check has

CRS-4638: Oracle High Availability Services is online
```

Check the current status of all the resources using the following command. It replaces the crs_stat -t command on 11gR1 and earlier.

```
[grid@knewracn1 ~]$ crsctl status  resource -t
--------------------------------------------------------------------------------
Name           Target  State        Server                   State details
--------------------------------------------------------------------------------
Local Resources
--------------------------------------------------------------------------------
ora.ASMNET1LSNR_ASM.lsnr
               ONLINE  ONLINE       knewracn1                STABLE
               ONLINE  ONLINE       knewracn2                STABLE
               ONLINE  ONLINE       knewracn4                STABLE
ora.DATA1.dg
               ONLINE  ONLINE       knewracn1                STABLE
               ONLINE  ONLINE       knewracn2                STABLE
               ONLINE  ONLINE       knewracn4                STABLE
ora.LISTENER.lsnr
               ONLINE  ONLINE       knewracn1                STABLE
               ONLINE  ONLINE       knewracn2                STABLE
               ONLINE  ONLINE       knewracn4                STABLE
ora.LISTENER_LEAF.lsnr
               OFFLINE OFFLINE      knewracn5                STABLE
               OFFLINE OFFLINE      knewracn6                STABLE
               OFFLINE OFFLINE      knewracn7                STABLE
               OFFLINE OFFLINE      knewracn8                STABLE
ora.net1.network
               ONLINE  ONLINE       knewracn1                STABLE
               ONLINE  ONLINE       knewracn2                STABLE
               ONLINE  ONLINE       knewracn4                STABLE
ora.ons
               ONLINE  ONLINE       knewracn1                STABLE
               ONLINE  ONLINE       knewracn2                STABLE
               ONLINE  ONLINE       knewracn4                STABLE
ora.proxy_advm
               ONLINE  ONLINE       knewracn1                STABLE
               ONLINE  ONLINE       knewracn2                STABLE
               ONLINE  ONLINE       knewracn4                STABLE
--------------------------------------------------------------------------------
Cluster Resources
--------------------------------------------------------------------------------
ora.LISTENER_SCAN1.lsnr
      1        ONLINE  ONLINE       knewracn2                STABLE
ora.LISTENER_SCAN2.lsnr
      1        ONLINE  ONLINE       knewracn4                STABLE
ora.LISTENER_SCAN3.lsnr
      1        ONLINE  ONLINE       knewracn1                STABLE
ora.MGMTLSNR
      1        ONLINE  ONLINE       knewracn1                169.254.199.3 192.16.

8.9.41,STABLE
ora.asm
      1        ONLINE  ONLINE       knewracn1                STABLE
      2        ONLINE  ONLINE       knewracn2                STABLE
```

```
      3        ONLINE  ONLINE       knewracn4                STABLE
ora.cvu
      1        ONLINE  ONLINE       knewracn1                STABLE
ora.gns
      1        ONLINE  ONLINE       knewracn1                STABLE
ora.gns.vip
      1        ONLINE  ONLINE       knewracn1                STABLE
ora.knewdb.db
      1        ONLINE  ONLINE       knewracn2                Open,STABLE
      2        ONLINE  ONLINE       knewracn4                Open,STABLE
      3        ONLINE  ONLINE       knewracn1                Open,STABLE
ora.knewracn1.vip
      1        ONLINE  ONLINE       knewracn1                STABLE
ora.knewracn2.vip
      1        ONLINE  ONLINE       knewracn2                STABLE
ora.knewracn4.vip
      1        ONLINE  ONLINE       knewracn4                STABLE
ora.mgmtdb
      1        ONLINE  ONLINE       knewracn1                Open,STABLE
ora.oc4j
      1        ONLINE  ONLINE       knewracn1                STABLE
ora.scan1.vip
      1        ONLINE  ONLINE       knewracn2                STABLE
ora.scan2.vip
      1        ONLINE  ONLINE       knewracn4                STABLE
ora.scan3.vip
      1        ONLINE  ONLINE       knewracn1                STABLE
--------------------------------------------------------------------------------
```

These commands can be executed by root user, grid (GI owner), and Oracle (RAC owner). You also can disable or enable all the CRSDs:

```
$GRID_HOME/bin/crsctl disable crs
$GRID_HOME/bin/crsctl enable crs
```

Managing OCR and the Voting Disk

Oracle provides three tools to manage OCR: `ocrconfig`, `ocrdump`, and `ocrcheck`. The `ocrcheck` command lists the OCR and its mirrors.

The following example lists the OCR location in the +VOCR diskgroup and its mirror in the +DATA1 diskgroup. In 11gR2 and 12cR1, OCR can have up to five mirrored copies. Each mirrored copy can be an ASM diskgroup or a cluster file system:

```
 $ ocrcheck
Status of Oracle Cluster Registry is as follows:
         Version                  :          3
         Total space (kbytes)     :     262120
         Used space (kbytes)      :       3192
         Available space (kbytes) :     258928
         ID                       : 1707636078
         Device/File Name         :      +VOCR
```

```
              Device/File integrity check succeeded
                                        Device/File Name         :      +DATA1/

 Device/File integrity check succeeded
                                     Device/File not configured
                                     Device/File not configured
                                     Device/File not configured

         Cluster registry integrity check succeeded

    Logical corruption check bypassed due to non-privileged user
You also can use the ocrconfig command to add/delete/replace OCR files, and you can add another
mirror of OCR in +DATA2:
                            $GRID_HOME/bin/ocrconfig -add +DATA2

Or remove the OCR copy from +DATA1 :
$GRID_HOME/bin/ocrconfig -delete +DATA1
```

The `ocrdump` command can be used to dump the contents of the OCR to a .txt or .xml file. It can be executed only by the root user, and the default file name is OCRDUMPFILE:

```
$ ./ocrdump
$ ls -l OCRDUMPFILE
-rw------- 1 root root 212551 Dec 28 20:21 OCRDUMPFILE
```

The OCR is backed up automatically every four hours on at least one of the nodes in the cluster. The backups are stored in the `$GRID_HOME/cdata/<cluster_name>` directory. To show the backup information, use the `ocrconfig -showbackup` command:

```
$GRID_HOME/bin/ocrconfig -showbackup

knewracn1     2013/03/02 07:01:37     /u01/app/12.1.0/grid/cdata/knewrac/backup00.ocr
knewracn1     2013/03/02 03:01:33     /u01/app/12.1.0/grid/cdata/knewrac/backup01.ocr

knewracn1     2013/03/01 23:01:32     /u01/app/12.1.0/grid/cdata/knewrac/backup02.ocr
knewracn1     2013/03/01 03:01:21     /u01/app/12.1.0/grid/cdata/knewrac/day.ocr
knewracn1     2013/02/20 02:58:55     /u01/app/12.1.0/grid/cdata/knewrac/week.ocr
knewracn1     2013/02/19 23:15:34     /u01/app/12.1.0/grid/cdata/knewrac/backup_20130219_231534.ocr
knewracn1     2013/02/19 23:05:26     /u01/app/12.1.0/grid/cdata/knewrac/backup_20130219_230526.ocr
.....
```

The steps to restore OCR from a backup file are as follows:

1. Identify the backup by using the `ocrconfig -showbackup` command.
2. Stop the clusterware on all the cluster nodes.
3. Perform the restore with the restore command:
   `ocrconfig -restore file_name`
4. Restart the crs and do an OCR integrity check by using `cluvfy comp ocr.`

You can use the following command to check the VD location:

```
$ crsctl query css votedisk
##  STATE    File Universal Id               File Name Disk group
--  -----    -----------------               --------- ---------
 1. ONLINE   7141f13f99734febbf94c73148c35a85 (/dev/dm-8) [VOCR]
                                  Located 1 voting disk(s).
To move the VD to another location, you can use the following crsctl command:
                                   $GRID_HOME/bin/crscrsctl replace votedisk +DATA3
```

Managing CRS Resources

The `srvctl` utility can be used to manage the resources that the Clusterware manages. The resources include database, instance, service, nodeapps, vip, asm, diskgroup, listener, scan, scan listener, serer pool, server, oc4j, home, file system, and gns. This managed resource is specified in the `<object>` part of the command. The command also specifies the management operation on the resource specified in the `<action>` part of the command. The operations include `enable`, `disable`, `start`, `stop`, `relocate`, `status`, `add`, `remove`, `modify`, `getenv`, `setenv`, `unsetenv`, `config`, `convert`, and `upgrade`.

```
srvctl <action> <object> [<options>]
```

Here are a few examples of `SRVCTL` commands.
Check the SCAN configuration of the cluster:

```
$srvctl config scan
SCAN name: knewrac-scan.kcloud.dblab.com, Network: 1
Subnet IPv4: 172.16.0.0/255.255.0.0/eth0
Subnet IPv6:
SCAN 0 IPv4 VIP: -/scan1-vip/172.16.150.40
SCAN name: knewrac-scan.kcloud.dblab.com, Network: 1
Subnet IPv4: 172.16.0.0/255.255.0.0/eth0
Subnet IPv6:
SCAN 1 IPv4 VIP: -/scan2-vip/172.16.150.83
SCAN name: knewrac-scan.kcloud.dblab.com, Network: 1
Subnet IPv4: 172.16.0.0/255.255.0.0/eth0
Subnet IPv6:
SCAN 2 IPv4 VIP: -/scan3-vip/172.16.150.28
```

Check the node VIP status on `knewracn1`:

```
$ srvctl status vip  -n knewracn1
VIP 172.16.150.37 is enabled
VIP 172.16.150.37 is running on node: knewracn1
```

Check the node apps on `knewracn1`:

```
$ srvctl status nodeapps  -n knewracn1
VIP 172.16.150.37 is enabled
VIP 172.16.150.37 is running on node: knewracn1
Network is enabled
Network is running on node: knewracn1
ONS is enabled
ONS daemon is running on node: knewracn1
```

Adding and Removing Cluster Nodes

The flexibility of Oracle Clusterware is exhibited through its ability to scale up and scale down the existing cluster online by adding and removing nodes in conformity with the demands of the business. This section will outline the procedure to add and remove nodes from the existing cluster.

Adding a Node

Assume that you have a two-node cluster environment and want to bring in an additional node (named rac3) to scale up the existing cluster environment, and that the node that is going to be part of the cluster meets all prerequisites essential to begin the procedure to add a node.

Adding a new node to the existing cluster typically consists of the following stages:

- Cloning Grid Infrastructure Home (cluster/ASM)
- Cluster configuration
- Cloning RDBMS home

When the new node is ready with all necessary prerequisites to become part of the existing cluster, such as storage, network, OS, and patches, use the following step-by-step procedure to add the node:

From the first node of the cluster, execute the following command to initiate integrity verification checks for the cluster and on the node that is going to be part of the cluster:

```
$ cluvfy stage –pre nodeadd –n rac3 –fixup -verbose
```

When no verification check failures are reported, use the following example to launch the procedure to add the node, assuming that the Dynamic Host Configuration Protocol (DHCP) and Grid Naming Service (GNS) are not configured in the current environment:

```
$ $GRID_HOME/oui/bin/addNode.sh –silent "CLUSTER_NEW_NODES={rac3}"
              "CLUSTER_NEW_VIRTUAL_HOSTNAMES={rac3_vip}"
```

Use the following example when adding to the Flex Cluster setup:

```
$ $GRID_HOME/oui/bin/addNode.sh –silent "CLUSTER_NEW_NODES={rac3}"
 "CLUSTER_NEW_VIRTUAL_HOSTNAMES={rac3-vip}" "CLUSTER_NEW_NODE_ROLES={hub}"
```

Execute the `root.sh` script as the root user when prompted on the node that is joining the cluster. The script will initialize cluster configuration and start up the cluster stack on the new node.

After successfully completing the procedure to add a new node, perform post node add verification checks from any cluster node using the following example:

```
$ cluvfy stage –post nodeadd –n rac3
$ crsctl check cluster –all –- verify the cluster health from all nodes
$ olsnodes –n  -- to list all existing nodes in a cluster
```

After a successful node addition, execute the following from `$ORACLE_HOME` to clone the Oracle RDBMS software over the new node to complete the node addition procedure:

```
$ORACLE_HOME/oui/bin/addNode.sh "CLUSTER_NEW_NODES={rac3}"
```

When prompted, execute the `root.sh` script as the root user on the new node.

Once a new node is successfully added to the cluster, run through the following post-addnode command:

```
$ ./cluvfy stage –post addnode –n rac3 -verbose
```

Removing a Node

Assume that you have a three-node cluster environment and want to delete the rac3 node from the existing cluster. Ensure the node that is going to be dropped has no databases, instances, or other services running. If any do exist, either drop them or just move them over to other nodes in the cluster. The following steps outline a procedure to remove a node from the existing cluster:

The node that is going to be removed shouldn't be pinned. If so, unpin the node prior to starting the procedure. The following examples demonstrate how to identify if a node is pinned and how to unpin the node:

```
$ olsnodes –n –s -t
```

You will get the following typical output if the nodes are pinned in the cluster:

```
rac1        1       Active  Pinned
rac2        2       Active  Pinned
rac3        3       Active  Pinned
```

Ensure that the cluster stack is up and running on node rac3. If the cluster is inactive on the node, you first need to bring the cluster up on the node and commence the procedure to delete the node.

Execute the following command as the root user from any node if the node that is going to be removed is pinned:

```
$ crsctl unpin css –n rac3
```

Run the following command as the root user on the node that is going to be removed:

```
$GRID_HOME/deinstall/deinstall –local
```

■ **Note** The `–local` argument must be specified to remove the local node; otherwise, the cluster will be deinstalled from every node of the cluster.

Run the following command as the root user from an active node in a cluster:

```
$crsctl delete node –n rac3
```

From any active node, execute the following command to update the Oracle inventory for GI and RDBMS homes across all nodes:

```
$GRID_HOME/oui/bin/runInstaller –updateNodeList ORACLE_HOME=$GRID_HOME
cluster_nodes={rac1,rac2} CRS=TRUE -silent

$GRID_HOME/oui/bin/runInstaller –updateNodeList ORACLE_HOME=$ORACLE_HOME
cluster_nodes={rac1,rac2} CRS=TRUE  –silent
```

When you specify the `-silent` option, the installer runs in silent mode and therefore doesn't display any interactive screens. In other words, it will run in non-interactive mode.

From any active node, verify the post-node deletion:

```
$cluvfy stage -post nodedel -n rac3 -verbose
$olsnodes -n -s -t
```

Clean up the following directories manually on the node that was just dropped:

```
/etc/oraInst.loc, /etc/oratab, /etc/oracle/ /tmp/.oracle, /opt/ORCLmap
```

Also, the filesystem where cluster and RDBMS software was installed.

Troubleshooting common Clusterware Stack Start-Up Failures

Various factors could contribute to the inability of the cluster stack to come up automatically after a node eviction, failure, reboot, or when cluster startup initiated manually. This section will focus and cover some of the key facts and guidelines that will help with troubleshooting common causes for cluster stack startup failures. Though the symptoms discussed here are not exhaustive or complete, the key points explained in this section indeed provide a better perspective to diagnose various cluster daemon processes common start-up failures and other issues.

Just imagine: a node failure or cluster manual shutdown, and subsequent cluster startup doesn't start the Clusterware as expected. Upon verifying the cluster or CRS health status, one of the following error messages have been encountered by the DBA:

```
$GRID_HOME/bin/crsctl check cluster

CRS-4639: Could not contact Oracle High Availability Services
CRS-4639: Could not contact Oracle High Availability Services
CRS-4000: Command Check failed, or completed with errors
OR

CRS-4535: Cannot communicate with Cluster Ready Services
CRS-4530: Communications failure contacting Cluster Synchronization Services daemon
CRS-4534: Cannot communicate with Event Manager
```

ohasd start up failures – This section will explain and provide most significant information to diagnose common issues of Oracle High Availability Services (OHAS) daemon process startup failures and provide workarounds for the following issues:

```
CRS-4639: Could not contact Oracle High Availability Services

    OR

CRS-4124: Oracle High Availability Services startup failed
CRS-4000: Command Start failed, or completed with errors
```

First, review the Clusterware alert and ohasd.log files to identify the root cause for the daemon startup failures.

Verify the existence of the `ohasd` pointer, as follows, in the OS-specific file:

```
/etc/init, /etc/inittab h1:3:respawn:/sbin/init.d/init.ohasd run >/dev/null 2>&1 </dev/null
```

This pointer should have been added automatically upon cluster installation and upgrade. In case no pointer is found, add the preceding entry toward the end of the file and as the root user start the cluster manually or initiate the `inittab` to start up these things automatically.

If the `ohasd` pointer exists, the next thing to check is the cluster high availability daemon auto start configuration. Use the following command as the root user to confirm the auto startup configuration:

```
$ GRID_HOME/bin/crsctl config has  -- High Availability Service
$ GRID_HOME/bin/crsctl config crs   -- Cluster Ready Service
```

Optionally, you can also verify the files under the `/var/opt/oracle/scls_scr/hostname/root or /etc/oracle/scls_scr/hostname/root` location to identify whether the auto config is enabled or disabled.

As the root user, enable the auto start and bring up the cluster manually on the local node when the auto startup is not configured. Use the following examples to enable has/crs auto-start:

```
$ CRS-4621: Oracle High Availability Services autostart is disabled.

Example:
$ GRID_HOME/bin/crsctl enable has    – turns on auto startup option of ohasd
$ GRID_HOME/bin/crsctl enable crs    - turns on auto startup option of crs

$ GRID_HOME/bin/crsctl start has – initiate OHASD daemon startup
$ GRID_HOME/bin/crsctl start crs – initiate CRS daemon startup
```

Despite the preceding, if the `ohasd` daemon process doesn't start and the problem persists, then you need to examine the component-specific trace files to troubleshoot and identify the root cause. Follow these guidelines:

Verify the existence of the `ohasd` daemon process on the OS. From the command-line prompt, execute the following:

```
ps -ef |grep init.ohasd
```

Examine OS platform–specific log files to identify any errors (refer to the operating system logs section later in this chapter for more details).

Refer the `ohasd.log` trace file under the `$GRID_HOME/log/hostname/ohasd` location, as this file contains useful information about the symptoms.

Address any OLR issues that are being reported in the trace file. If OLR corruption or inaccessibility is reported, repair or resolve the issue by taking appropriate action. In case of a restore, restore it from a previous valid backup using the `$ocrconfig  -local –restore $backup_location/backup_filename.olr` command.

Verify Grid Infrastructure directory ownership and permission using OS level commands.

Additionally, remove the cluster startup socket files from the `/var/tmp/.oracle, /usr/tmp/.oracle, /tmp/.oracle` directory and start up the cluster manually. The existence of the directory is subject to operating system dependency.

CSSD startup issues – In case the CSSD process fails to start up or is reported to be unhealthy, the following guidelines help in identifying the root cause of the issue:

```
Error : CRS-4530: Communications failure contacting Cluster Synchronization Services daemon:
```

Review the Clusterware alert.log and ocssd.log file to identify the root cause of the issue.

Verify the CSSD process on the OS:

```
ps –ef |grep cssd.bin
```

Examine the `alert_hostname.log` and `ocssd.log` logs to identify the possible causes that are preventing the CSSD process from starting.

Ensure that the node can access the VDs. Run the `crsctl query css votedisk` command to verify accessibility. If the node doesn't access the VD files for any reason, check for disk permission and ownership and for logical corruptions. Also, take the appropriate action to resolve the issues by either resetting the ownership and permission or by restoring the corrupted OCR file.

If any heartbeat (network|disk) problems are reported in the logs mentioned earlier, verify the private interconnect connectivity and other network-related settings on the node.

If the VD files are placed on ASM, ensure that the ASM instance is up. In case the ASM instance is not up, refer to the ASM instance alert.log to identify the instance's startup issues.

Use the following command to verify `asm`, `cluster_connection`, `cssd`, and other cluster resource status:

```
$ crsctl stat res -init -t

--------------------------------------------------------------------------------
Name           Target  State        Server                   State details
--------------------------------------------------------------------------------
Cluster Resources
--------------------------------------------------------------------------------
ora.asm
      1        ONLINE  ONLINE       rac1                     Started,STABLE
ora.cluster_interconnect.haip
      1        ONLINE  OFFLINE      rac1                     STABLE
ora.crsd
      1        ONLINE  OFFLINE      rac1                     STABLE
ora.cssd
      1        ONLINE  OFFLINE      rac1                     STABLE
ora.cssdmonitor
      1        ONLINE  UNKNOWN      rac1                     STABLE
ora.ctssd
      1        ONLINE  ONLINE       rac1                     ACTIVE:0,STABLE
ora.diskmon
      1        OFFLINE OFFLINE                               STABLE
ora.drivers.acfs
      1        ONLINE  ONLINE       rac1                     STABLE
ora.evmd
      1        ONLINE  ONLINE       rac1                     STABLE
ora.gipcd
      1        ONLINE  ONLINE       rac1                     STABLE
ora.gpnpd
      1        ONLINE  ONLINE       rac1                     STABLE
ora.mdnsd
      1        ONLINE  ONLINE       rac1                     STABLE
ora.storage
      1        ONLINE  ONLINE       rac1                     STABLE
```

If you find the `ora.cluster_interconnect.hiap` resource is `OFFLINE`, you might need to verify the interconnect connectivity and check the network settings on the node. Also, you can try to startup the offline resource manually using the following command:

```
$GRID_HOME/bin/crsctl start res ora.cluster_interconnect.haip –init
```

Bring up the offline cssd daemon manually using the following command:

```
$GRID_HOME/bin/crsctl start res ora.cssd –init
```

The following output will be displayed on your screen:

```
CRS-2679: Attempting to clean 'ora.cssdmonitor' on 'rac1'
CRS-2681: Clean of 'ora.cssdmonitor' on 'rac1' succeeded
CRS-2672: Attempting to start 'ora.cssdmonitor' on 'rac1'
CRS-2676: Start of 'ora.cssdmonitor' on 'rac1' succeeded
CRS-2672: Attempting to start 'ora.cssd' on 'rac1'
CRS-2676: Start of 'ora.cssd' on 'rac1' succeeded
CRS-2672: Attempting to start 'ora.cluster_interconnect.haip' on 'rac1'
CRS-2672: Attempting to start 'ora.crsd' on 'rac1'
CRS-2676: Start of 'ora.cluster_interconnect.haip' on 'rac1' succeeded
CRS-2676: Start of 'ora.crsd' on 'rac1' succeeded
```

CRSD startup issues – When the CRSD–related startup and other issues are being reported, the following guidelines provide assistance to troubleshoot the root cause of the problem:

```
CRS-4535: Cannot communicate with CRS:
```

Verify the CRSD process on the OS:

```
ps –ef |grep crsd.bin
```

Examine the `crsd.log` to look for any possible causes that prevent the CRSD from starting.

Ensure that the node can access the OCR files; run the `'ocrcheck'` command to verify. If the node can't access the OCR files, check the following:

Check the OCR disk permission and ownership.

If OCR is placed on the ASM diskgroup, ensure that the ASM instance is up and that the appropriate diskgroup is mounted.

Repair any OCR-related issue encountered, if needed.

Use the following command to ensure that the CRSD daemon process is ONLINE:

```
$ crsctl stat res -init -t

--------------------------------------------------------------------------------
NAME           TARGET  STATE        SERVER                   STATE_DETAILS
--------------------------------------------------------------------------------
Cluster Resources
--------------------------------------------------------------------------------
ora.crsd
      1        ONLINE  OFFLINE       rac1
```

You can also start the individual daemon manually using the following command:

```
$GRID_HOME/bin/crsctl start res ora.cssd –init
```

In case the Grid Infrastructure malfunctions or its resources are reported as being unhealthy, you need to ensure the following:

- Sufficient free space must be available under the `$GRID_HOME` and `$ORACLE_HOME` filesystem for the cluster to write the events in the respective logs for each component.
- Ensure enough system resource availability, in terms of CPU and memory.
- Start up any individual resource that is OFFLINE.

Clusterware Exclusive Mode

Beginning with 11gR2 (11.2.0.2), the cluster stack can be invoked in an exclusive mode to carry out a few exclusive cluster maintenance tasks, such as restoring OCR and VDs, troubleshooting `root.sh` issues, and so on. To start the cluster in this mode on any particular node, the cluster stack must not be active on other nodes in the cluster. When a cluster is started in exclusive mode, no VD and networks are required. Use the following command as the root user to bring the cluster in exclusive mode:

```
$crsctl start crs –excl {-nocrs} {-nowait}
```

With the `–nocrs` argument, Oracle Clusterware will be started without the CRSD process, and with the `–nowait` argument, Clusterware start doesn't depend on Oracle High Availability Service (`ohasd`) daemon start.

Troubleshooting OCR Issues

In the event of any OCR issues, such as logical corruption, missing permissions and ownership, integrity, and loss of mirror copy, the following troubleshooting and workaround methods are extremely helpful for identifying the root cause as well as resolving the issues:

Verify the OCR file integrity using the following cluster utilities:

```
$ocrcheck                              - verifies OCR integrity & logical corruption
$ocrcheck –config                      - lists OCR disk location and names
$ocrcheck –local –config               - lists LOR name and location
$cluvfy comp ocr -n all –verbose       – verifies integrity from all nodes
$cluvfy comp ocr -n rac1 –verbose      - verifies integrity on the local node
```

With the `ocrdump` utility, you can dump either the entire contents or just a section from the OCR file into a text file. The following commands achieve that:

```
$ ocrdump <filename.txt>
-- to obtains a detailed output, run the command as the root user
```

With the preceding command issued, OCR contents will be dumped into a text file, and if the output filename is not mentioned, a file named `OCRDUMPFILE` will be generated in the local directory.

```
$ ocrdump –stdout –keyname SYSTEM.css {-xml}
```

The preceding command lists `css` section–specific contents from the current OCR file, and the contents will be displayed on the prompt if the output is not diverted into any file.

```
$ ocrdump –backupfile <filename and location>
-- will dump specific backup contents.
```

Diagnose, Debug, Trace Clusterware and RAC Issues

When the default debugging information generated by the Oracle Clusterware processes based on their default trace level settings doesn't provide enough clues to reach a conclusion to a problem, it is necessary to increase the default trace levels of specific components and their subcomponents to get comprehensive information about the problem. The default tracing levels of Clusterware components is set to value 2, which is sufficient in most cases.

In the following sections, we will demonstrate how to modify, enable, and disable the debugging tracing levels of various cluster components and their subcomponents using the cluster commands.

To understand and list various cluster attributes and their default settings under a specific Clusterware component, use the following example command:

```
$ crsctl stat res ora.crsd -init -t –f
```

The output from the preceding example helps you to find the default settings for all arguments of a specific component, like stop/start dependencies, logging/trace level, auto-start, failure settings, time, and so on.

Debugging Clusterware Components and Resources

Oracle lets you dynamically modify and disable the default tracing levels of any of the cluster daemon (`CRSD, CSSD, EVMD`) processes and their subcomponents. The `crsctl set {log|trace}` command allows modification of the default debug setting dynamically. The trace levels range from 1 to 5, whereas the value 0 turns off the tracing option. Higher trace levels generate additional diagnostic information about the component.

The following example lists the default log settings for all modules of a component; the command must be executed as the root user to avoid an Insufficient User Privileges error:

```
$ crsctl get log {css|crs|evm} ALL
```

The following output fetches the default trace levels of various subcomponents of CSSD:

```
Get CSSD Module: BCCM  Log Level: 2
Get CSSD Module: CLSF  Log Level: 0
Get CSSD Module: CLSINET  Log Level: 0
Get CSSD Module: CSSD  Log Level: 2
Get CSSD Module: GIPCBCCM  Log Level: 2
Get CSSD Module: GIPCCM  Log Level: 2
Get CSSD Module: GIPCGM  Log Level: 2
Get CSSD Module: GIPCNM  Log Level: 2
Get CSSD Module: GPnP  Log Level: 1
Get CSSD Module: OLR  Log Level: 0
Get CSSD Module: SKGFD  Log Level: 0
```

To list all components underneath of a module, use the following example as the root user:

```
$ crsctl lsmodules                              -- displays the list of modules
$ crsctl lsmodules {css|crs|evm}                -- displays the sub-components of a module
```

To set a non-default tracing level, use the following syntax as the root user:

```
Syntax:
$ crsctl set log {module}  "component_name=debug_level"
$ crsctl set log res "resourcename=debug_level"
Example:
$ crsctl set log crs crsmain=3
$ crsctl set log crs crsmain=3,crsevt=4
--- let you set different log levels to multiple modules

$ crsctl set log crs all=5
$ crsctl set log res ora.rondb.db:5
```

If the node is evicting due to some mysterious network heartbeat (NHB) issues and the default information is not sufficient to diagnose the cause, you can increase the CSSD tracing level to a higher number. To troubleshoot NHB-related issues, you can set the log level to 3 as the root user, as shown in the following example:

```
$ crsctl set log css ocssd=4
```

The following examples disable the tracing:

```
$ crsctl set log crs crsmain=0
$ crsctl set log res ora.rondb.db:0
$ crsctl set log res ora.crs:0 -init
```

The `-init` flag must be specified while modifying the debug mode of a key cluster daemon process. To list the current logging and tracing levels of a particular component and its subcomponents, use the following example:

```
$crsctl stat res ora.crsd -init -f |grep LEVEL
```

Tracing levels also can be set by specifying the following environmental variables on the local node (however, you need to restart the cluster on the local node to enforce the logging/tracing changes):

```
$ export ORA_CRSDEBUG_ALL=1   --sets debugging level 1 to all modules
$ export ORA_CRSDDEBUG_CRS=2  --sets debugging level 2 to CRS module
```

OS Level Tracing

You should also be able to use the OS-specific tracing utility (`gdb, pstack, truss, strace`, and so on) to dump the debug information of an OS process. The following exercise demonstrates the procedure:

Identify the process ID that you want to set the OS level tracing; for example:

```
$ps -ef |grep oraagent.bin
```

Attach the process with the OS-specific debug utility; for example, on the HP-UX platform:

```
$pstack /u00/app/12.1.0/grid/bin/orarootagent.bin 4558
```

You can then provide the information to Oracle Support or consult your OS admin team to help you identify any issues that were raised from the OS perspective.

Diagnose cluvfy Failures

The `cluvfy:runCluvfy` utility can be used to accomplish pre-component and post-component verification checks, including OS, network, storage, overall system readiness, and clusterware best practices. When the utility fails to execute for no apparent reason, and in addition the `-verbose` argument doesn't yield sufficient diagnostic information about the issue, enable the debugging mode for the utility and re-execute the command to acquire adequate information about the problem. The following example demonstrates enabling debugging mode:

```
$ export SRVM_TRACE=true
```

Rerun the failed command after setting the preceding environmental variable. A detailed output file will be generated under the `$GRID_HOME/cv/log` location, which can be used to diagnose the real cause. When debug settings are modified, the details are recorded in the OCR file and the changes will be affected on that node only.

In addition, when Java-based Oracle tools (such as `srvctl, dbca, dbua, cluvfy`, and `netca`) fail for unknown reasons, the preceding setting will also help to generate additional diagnostic information that can be used to troubleshoot the issues.

```
Example:
$srvctl status database -d
```

■ **Note** When the basic information from the CRS logs doesn't provide sufficient feedback to conclude the root cause of any cluster or RAC database issue, setting different levels of trace mode might produce useful, additional information to resolve the problem. However, the scale of the debug mode level will have an impact on the overall cluster performance and also potentially generate a huge amount of information in the respective log files. On top of that, it is highly advised to seek the advice of Oracle Support prior to tampering with the default settings of cluster components.

Grid Infrastructure Component Directory Structure

Each component in Grid Infrastructure maintains a separate log file and records sufficient information under normal and critical circumstances. The information written in the log files will surely assist in diagnosing and troubleshooting Clusterware components or cluster health-related problems. Exploring the appropriate information from these log files, the DBA can diagnose the root cause to troubleshoot frequent node evictions or any fatal Clusterware problems, in addition to Clusterware installation and upgrade difficulties. In this section, we explain some of the important CRS logs that can be examined when various Clusterware issues occur.

`alert<HOSTNAME>.log`: Similar to a typical database alert log file, Oracle Clusterware manages an alert log file under the `$GRID_HOME/log/$hostname` location and posts messages whenever important events take place, such as when a cluster daemon process starts, when a process aborts or fails to start a cluster resource, or when node eviction occurs. It also logs information about node eviction occurrences and logs when a voting, OCR disk becomes inaccessible on the node.

Whenever Clusterware confronts any serious issue, this should be the very first file to be examined by the DBA seeking additional information about the problem. The error message also points to a trace file location where more detailed information will be available to troubleshoot the issue.

Following are a few sample messages extracted from the alert log file, which explain the nature of the event, like node eviction, CSSD termination, and the inability to auto start the cluster:

```
[ohasd(10937)]CRS-1301:Oracle High Availability Service started on node rac1.
 [/u00/app/12.1.0/grid/bin/oraagent.bin(11137)]CRS-5815:Agent
'/u00/app/12.1.0/grid/bin/oraagent_oracle' could not find any base type
 entry points for type 'ora.daemon.type'. Details at (:CRSAGF00108:) {0:1:2} in
/u00/app/12.1.0/grid/log/rac1/agent/ohasd/oraagent_oracle/oraagent_oracle.log.
[cssd(11168)]CRS-1713:CSSD daemon is started in exclusive mode
[cssd(11168)]CRS-1605:CSSD voting file is online: /dev/rdsk/oracle/vote/ln1/ora_vote_002; details in
/u00/app/12.1.0/grid/log/rac1/cssd/ocssd.log.

[cssd(11052)]CRS-1656:The CSS daemon is terminating due to a fatal error; Details at (:CSSSC00012:)
in /u00/app/12.1.0/grid/log/rac1/cssd/ocssd.log
 [cssd(3586)]CRS-1608:This node was evicted by node 1, rac1; details at (:CSSNM00005:) in
/u00/app/12.1.0/grid/log/rac2/cssd/ocssd.log.
```

`ocssd.log`: Cluster Synchronization Service daemon (CSSD) is undoubtedly one of the most critical components of Clusterware, whose primary functionality includes node monitoring, group service management, lock services, and cluster heartbeats. The process maintains a log file named `ocssd.log` under the `$GIRD_HOME/log/<hostname>/cssd` location and writes all important event messages in the log file. This is one of the busiest CRS log files and is continuously being written; when the debug level of the process is set too high, the file tends to get more detailed information about the underlying issue. Before the node eviction happens on the node, it writes the warning message in the log file. If a situation such as node eviction, VD access issues, or inability of the Clusterware to start up on the local node is raised, it is strongly recommended that you examine the file to find out the reasons.

Following are a few sample entries of the log file:

2012-11-30 10:10:49.989: [CSSD][21]clssnmvDiskKillCheck: not evicted, file /dev/rdsk/c0t5d4 flags 0x00000000, kill block unique 0, my unique 1351280164

```
ocssd.l04:2012-10-26 22:17:26.750: [CSSD][6]clssnmvDiskVerify: discovered a potential voting file
ocssd.l04:2012-10-26 22:36:12.436: [CSSD][1]clssnmvDiskAvailabilityChange: voting file
/dev/rdsk/c0t5d4 now online
ocssd.l04:2012-10-26 22:36:10.440: [CSSD][1]clssnmReadDiscoveryProfile: voting file discovery
string(/dev/rdsk/c0t5d5,/dev/rdsk/c0t5d4)
2012-12-01 09:54:10.091: [CSSD][30]clssnmSendingThread: sending status msg to all nodes
ocssd.l01:2012-12-01 10:24:57.116: [CSSD][1]clssnmInitNodeDB: Initializing with OCR id 1484043234
```

- [cssd(7335)]CRS-1612:Network communication with node rac2 (02) missing for 50% of timeout interval. Removal of this node from cluster in 14.397 seconds2013-03-15 17:02:44.964
- [cssd(7335)]CRS-1611:Network communication with node rac2 (02) missing for 75% of timeout interval. Removal of this node from cluster in 7.317 seconds2013-03-15 17:02:50.024
- [cssd(7335)]CRS-1610:Network communication with node rac2 (02) missing for 90% of timeout interval. Removal of this node from cluster in

Oracle certainly doesn't recommend removing the log file manually for any reason, as it is governed by Oracle automatically. Upon reaching a size of 50 MB, the file will be automatically archived as `cssd.l01` in the same location as part of the predefined rotation policy, and a fresh log file (`cssd.log`) will be generated. There will be ten archived copies kept for future reference in the same directory as part of the built-in log rotation policy and ten-times-ten file retention formula.

When the log file is removed before it reaches 50 MB, unlike the database alter.log file, Clusterware will not generate a new log file instantly until the removed file reaches 50 MB. This is because despite the removal of the file from the OS, the CSS process will be still writing the messages to the file until the file becomes a candidate for the rotation policy. When the removed file reaches a size of 50 MB, a new log file will appear or be generated and will be available in this way. However, the previous messages won't be able to recall in this context.

When the CSSD fails to start up, or it is reported to be unhealthy, this file can be referred to ascertain the root cause of the problem.

`crsd.log`: CRSD is another critical component of Clusterware whose primary functionality includes resource monitoring, resource failover, and managing OCR. The process maintains a log file named `crsd.log` under the `$GIRD_HOME/log/<hostname>/crsd` location and writes all important event messages in the log file. Whenever a cluster or non-cluster resource stops or starts, or a failover action is performed, or any resource-related warning message or communication error occurs, the relevant information is written to the file. In case you face issues like failure of starting resources, the DBA can examine the file to get relevant information that could assist in a resolution of the issue.

Deleting the log file manually is not recommended, as it is governed by Oracle and archived automatically. The file will be archived as `crsd.l01` under the same location on reaching a size of 10 MB, and a fresh log file (`crsd.log`) will be generated. There will be ten archived copies kept for future reference in the same directory.

When the CRSD fails to start up, or is unhealthy, refer to this file to find out the root cause of the problem.

`ohasd.log`: The Oracle High Availability Service daemon (`OHASD`), a new cluster stack, was first introduced with 11gR2 to manage and control the other cluster stack. The primary responsibilities include managing OLR; `starting, stopping, and verifying` cluster health status on the local and remote nodes; and also supporting cluster-wide commands. The process maintains a log file named `crsd.log` under the `$GIRD_HOME/log/<hostname>/ohasd` location and writes all important event messages in the log file. Examine the file when you face issues running root.sh script, such as when the `ohasd` process fails to start up or in case of OLR corruption.

Oracle certainly doesn't encourage deleting the log file for any reason, as it is governed by Oracle automatically. The file will be archived as `ohasd.l01` under the same location on reaching a size of 10 MB, and a fresh log file (`ohasd.log`) will be generated. Like `crsd.log` and `crs.log`, there will be ten archived copies kept for future reference in the same directory.

```
2013-04-17 11:32:47.096: [ default][1] OHASD Daemon Starting. Command string :reboot
2013-04-17 11:32:47.125: [ default][1] Initializing OLR
2013-04-17 11:32:47.255: [  OCRRAW][1]proprioo: for disk 0
(/u00/app/12.1.0/grid_1/cdata/rac2.olr), id match (1), total id sets,

        need recover (0), my votes (0), total votes (0), commit_lsn (3118), lsn (3118)

        2013-04-17 11:32:47.368: [ default][1] Loading debug levels . . .

        2013-04-17 11:32:47.803: [  clsdmt][13]Creating PID [6401] file for home
        /u00/app/12.1.0/grid_1 host usdbp10 bin ohasd to /u00/app/12.1.0/grid_1/ohasd/init/
```

Upon successful execution of `ocrdump, ocrconfig, olsnodes, oifcfg,` and `ocrcheck` commands, a log file will be generated under the `$GRID_HOME/log/<hostname>/client` location. For EVM daemon (EVMD) process–relevant details, look at the `evmd.log` file under the `$GRID_HOME/log<hostname>/evmd` location. Cluster Health Monitor Services (`CHM`) and logger services are maintained under the `$GRID_HOME/log/<hostname>/crfmond, crflogd` directories.

Figure 2-4 depicts the hierarchy of the Clusterware component directory structure.

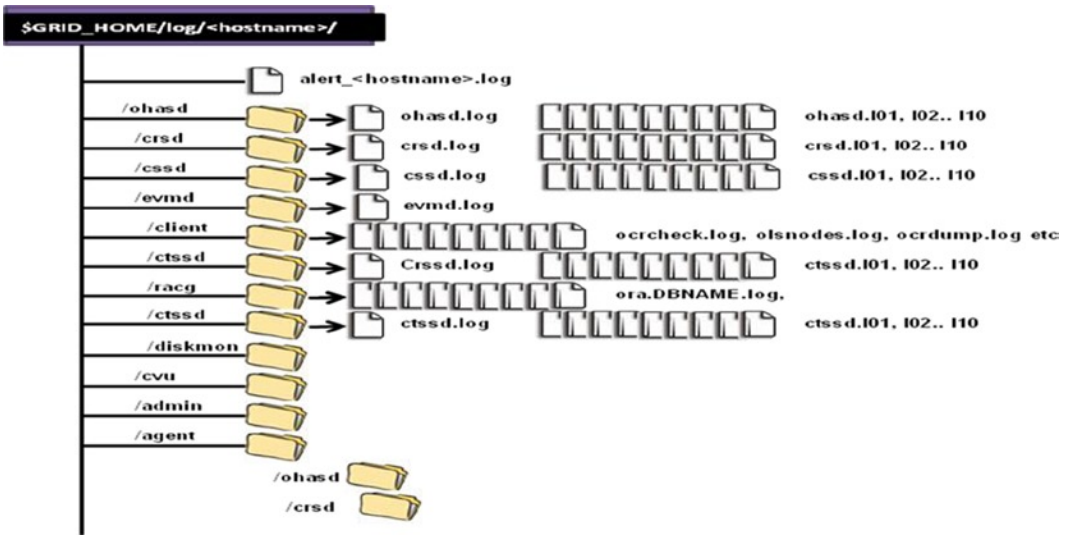


Figure 2-4. *Unified Clusterware log directory hierarchy*

`Operating system (OS) logs`: Referring to the OS-specific log file will be hugely helpful in identifying Clusterware startup and shutdown issues. Different platforms maintain logs at different locations, as shown in the following example:

```
HPUX - /var/adm/syslog/syslog.log
AIX - /bin/errpt –a
Linux - /var/log/messages
Windows  - Refer .TXT log files under Application/System log using Windows Event Viewer
Solaris - /var/adm/messages
```

■ **Caution** Oracle Clusterware generates and maintains 0-sized socket files in the hidden `'./oracle'` directory under the location `/etc` or `/var/tmp` (according to the platform). Removing these files as part of regular log cleanup or unintentionally removing them might lead to a cluster hung situation.

■ **Note** It is mandatory to maintain sufficient free space under the file system on which grid and RDBSM software are installed to prevent Clusterware issues; in addition, Oracle suggests not to remove the logs manually.

Oracle Clusterware Troubleshooting - Tools and Utilities

Managing and troubleshooting various issues related to Clusterware and its components are two of the key responsibilities of any Oracle DBA. Oracle provides a variety of tools and utilities in this context that the DBA can use to monitor Clusterware health and also diagnose and troubleshoot any serious Clusterware issues. Some of the key tools and utilities that Oracle provides are as follows: `CHM, diagcollection.sh`,`ProcWatcher`, `RACcheck`, `oratop`, `OSWatcher Black Box Analyzer (OSWbba)`, `The Light on-board monitor (LTOM)`,`Hang File Generator (HANGFG)`.

In the following sections, we will cover some of the uses of these very important tools and describe their advantages.

Cluster Health Check with CVU

Starting with 11.2.0.3, the cluster verification utility (`cluvfy`) is capable of carrying out the post-Clusterware and Database installation health checks. With the new `–healthcheck` argument, cluster and database components best practices, mandatory requirements, deviation, and proper functionality can be verified.

The following example collects detailed information about best-practice recommendations for Clusterware in an HTML file named `cvucheckreport_<timestamp>.htm`:

```
$./cluvfy comp healthcheck –collect cluster –bestpractice -html
```

When no further arguments are attached with the `healthcheck` parameter, the Clusterware and Database checks are carried out. Use the following example to perform the health checks on the cluster and database because no `–html` argument was specified; the output will be stored in a text file:

```
$./cluvfy comp healthcheck
```

The cluvfy utility supports the following arguments:

```
-collect cluster|database
             -bestpractices|-mandatory|-deviations
             -save -savedir  --to save the output under a particular location
             -html               -- output will be written in an HTML file
```

Real-Time RAC Database Monitoring - oratop

The oratop utility, which is currently restricted to the Linux operating system, resembles an OS-specific top-like utility, providing near–real-time resource monitoring capability for a RAC and single-instance database from 11.2.0.3 onward. It is a very lightweight monitoring utility that utilizes very minimal resources, 0.20% memory and <1% CPU, on the server. With this utility, you can monitor a RAC database, a stand-alone database, and local as well as remote databases.

Invoke the oratop Utility

Download the oratop.zip file from My Oracle Support (MOS) at https://support.oracle.com/epmos/faces/MosIndex.jspx?_afrLoop=463945118311568&_afrWindowMode=0&_adf.ctrl-state=19ctrm4ozz_4. Unzip the file and set the appropriate permission to the oratop file, which is chmod 755 oratop on the Linux platform. Ensure the following database init parameters: timed_statistics set to TRUE and the statistics_level set to TYPICAL. Also, the following environmental settings need to be set on the local node before invoking the utility:

```
$ ORACLE_UNQNAME=<dbname>
$ ORACLE_SID=<instance_name1>
$ ORACLE_HOME=<db_home>
$ export LD_LIBRARY_PATH=$ORACLE_HOME/lib
$ export PATH=$ORACLE_HOME/bin:$PATH
```

The following example runs the utility and sets the interval to every ten seconds to refresh the window (default is every three seconds):

```
$ratop -i 10

$oratop -t <tns_name_for_remote_db> -- to monitor remote database.
```

Input the database user name and password credentials when prompted. When no credentials are entered, it will use the default user SYSTEM with MANAGER as the default password to connect to the database. If you are using a non-system database user, ensure that the user has read permission on some of the dictionary dynamic views, such as v_$SESSION, v_vSYSMETRIC, v_$INSTANCE, v_ $PROCESS, v_$SYSTEM_EVENT, and so on.

Figure 2-5 shows the output window of the oratop utility.

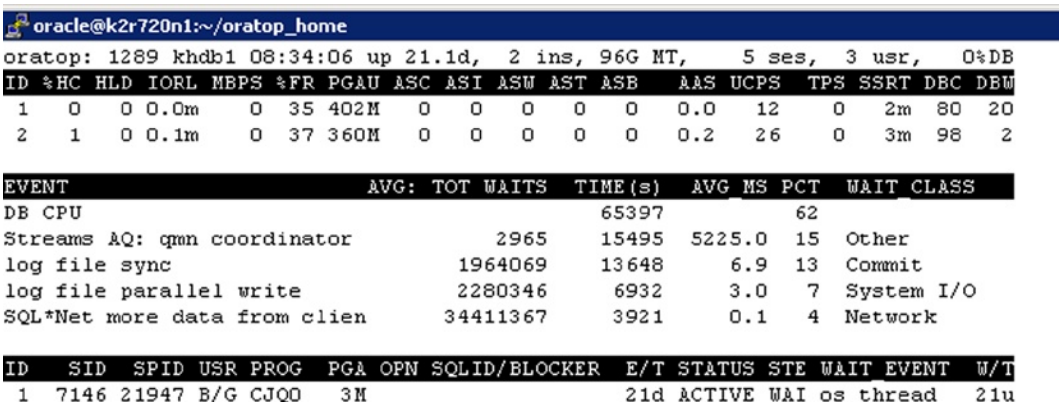


Figure 2-5. *oratop output screen shot*

The granular statistics that appear in the window help to identify database performance contention and bottlenecks. The live window guidelines are categorized into three major sections: 1) top five events (similar to the AWR/ASH report), 2) top Oracle sessions on the server in terms of high I/O and memory and 3) DB load (also provides blocking session details, etc.). Press `q` or `Q` to quit from the utility and press `Cont+C` to abort.

■ **Note** The tool is available for downloading only through MOS, which requires additional support licensing.

RAC Configuration Audit Tool - RACcheck

`RACcheck` is a tool that performs audits on various important configuration settings and provides a comprehensive HTML-based assessment report on the overall health check status of the RAC environment.

The tool is currently certified on the majority of operating systems that can be used in interactive and non-interactive modes and also supports multiple databases at a single run. It can be run across all nodes, on a subset of cluster nodes, or on a local node. When the tool is invoked, it carries out the health checks on various components, such as cluster-wide, CRS, Grid, RDBMS, ASM, general database initialization parameters, OS kernel settings, and OS packages. The most suitable time for performing health checks with this tool is immediately after deploying a new RAC environment, before and after planned system maintenances, prior to major upgrades, and quarterly.

With its Upgrade Readiness Assessment Module ability, it will simplify and enhance system upgrade readiness reliability. Apart from regular upgrade prerequisite verifications, the module lets you perform automatic prerequisite verification checks for patches, best practices, and configuration. This will be of great assistance before planning any major cluster upgrades.

Invoke the RACcheck Tool

Download the raccheck.zip file from MOS, unzip it, and set the appropriate permission to the `raccheck` file, which is `chmod 755 raccheck` on Unix platforms. To invoke the tool in interactive mode, use the following example at the command prompt as the Oracle software owner and provide the following input when prompted:

```
$./raccheck
```

To perform RAC upgrade readiness verification checks, use the following example and response with your inputs when prompted with questions:

```
$./raccheck –u –o pre
```

Following are the supported arguments with the tool:

```
$ ./raccheck -h

Usage : ./raccheck [-abvhpfmsuSo:c:rt:]
        -a      All (Perform best practice check and recommended patch check)
        -b      Best Practice check only. No recommended patch check
        -h      Show usage
        -v      Show version
        -p      Patch check only
-m      exclude checks for Maximum Availability Architecture        -u      Run raccheck to check
pre-upgrade or post-upgrade best
practices.-o pre or -o post is mandatory with -u option like ./raccheck -u -o pre
-f      Run Offline.Checks will be performed on data already        -o      Argument to an option.
if -o is followed by
              v,V,Verbose,VERBOSE or Verbose, it will print checks which
                passs on the screen
                if -o option is not specified,it will print only failures on
 screen. for eg: raccheck -a -o v
-r      To include High availability best practices also in regular
 healthcheck eg ./racchekck -r(not applicable for exachk)
        -c      Pass specific module or component to check best practice
 for.
```

The assessment report provides a better picture of the RAC environment and includes an overall system health check rating (out of 100), Oracle Maximum Availability Architecture (MAA) best practices, bug fixes, and patch recommendations.

■ **Note** The tool is available for download only through MOS, which requires additional support licensing. Executing the tool when the systems are heavily loaded is not recommended. It is recommended to test the tool in a non-production environment first, as it doesn't come by default with Oracle software.

Cluster Diagnostic Collection Tool - diagcollection.sh

Every time you run you into a few serious Clusterware issues and confront node eviction, you typically look at various CRS-level and OS-level logs to gather the required information to comprehend the root cause of the problem. Because Clusterware manages a huge number of logs and trace files, it will sometimes be cumbersome to review many logs from each cluster node. The `diagcollection.sh` tool, located under `GRID_HOME/bin`, is capable of gathering the required diagnostic information referring to various important sources, such as CRS logs, trace and core files, OCR data, and OS logs.

With the diagnostic collection tool, you have the flexibility to collect diagnostic information at different levels, such as cluster, Oracle RDBMS home, Oracle base, and Core analysis. The gathered information from various resources will then be embedded into a few zip files. You therefore need to upload these files to Oracle Support for further analysis to resolve the problem.

The following example will collect the $GRID_HOME diagnostic information:

```
./diagcollection.sh --collect --crs $GRID_HOME
```

The following CRS diagnostic archives will be created in the local directory:

```
crsData_usdbt43_20121204_1103.tar.gz -> logs, traces, and cores from CRS home.
```

■ **Note** Core files will be packaged only with the `--core` option.

```
ocrData_usdbt43_20121204_1103.tar.gz -> ocrdump, ocrcheck etc
coreData_usdbt43_20121204_1103.tar.gz -> contents of CRS core files in text format
osData_usdbt43_20121204_1103.tar.gz -> logs from operating system
Collecting crs data
log/usdbt43/cssd/ocssd.log: file changed size

Collecting OCR data
Collecting information from core files
Collecting OS logs
```

After data collection is complete, the following files will be created in the local directory:

```
crsData_$hostname_20121204_1103.tar.gz
ocrData_$hostname _20121204_1103.tar.gz
coreData_$hostname _20121204_1103.tar.gz
osData_$hostname _20121204_1103.tar.gz
```

The following example will assist you in getting the supported parameters list that can be used with the tool (output is trimmed):

```
./diagcollection.sh -help
    --collect
             [--crs] For collecting crs diag information
             [--adr] For collecting diag information for ADR; specify ADR location
             [--chmos] For collecting Cluster Health Monitor (OS) data
             [--all] Default.For collecting all diag information. <<<>>>
             [--core] Unix only. Package core files with CRS data
             [--afterdate] Unix only. Collects archives from the specified date.
             [--aftertime] Supported with -adr option. Collects archives after the specified
             [--beforetime] Supported with -adr option. Collects archives before the specified
             [--crshome] Argument that specifies the CRS Home location
             [--incidenttime] Collects Cluster Health Monitor (OS) data from the specified
             [--incidentduration] Collects Cluster Health Monitor (OS) data for the duration

             NOTE:
             1. You can also do the following
                ./diagcollection.pl --collect --crs --crshome <CRS Home>

     --clean        cleans up the diagnosability
                    information gathered by this script
```

```
     --coreanalyze  Unix only. Extracts information from core files
                    and stores it in a text file
        Use the --clean argument with the script to clean up previously generated files.
```

■ **Note** Ensure that enough free space is available at the location where the files are being generated. Furthermore, depending upon the level used to collect the information, the script might take a considerable amount of time to complete the job. Hence, keep an eye on resource consumption on the node. The tool must be executed as root user.

CHM

The Oracle CHM tool is designed to detect and analyze OS–and cluster resource–related degradations and failures. Formerly known as Instantaneous Problem Detector for Clusters or IPD/OS, this tool tracks the OS resource consumption on each RAC node, process, and device level and also connects and analyzes the cluster-wide data. This tool stores real-time operating metrics in the CHM repository and also reports an alert when certain metrics pass the resource utilization thresholds. This tool can be used to replay the historical data to trace back what was happening at the time of failure. This can be very useful for the root cause analysis of many issues that occur in the cluster such as node eviction.

For Oracle Clusterware 10.2 to 11.2.0.1, the CHM/OS tool is a standalone tool that you need to download and install separately. Starting with Oracle Grid Infrastructure 11.2.02, the CHM/OS tool is fully integrated with the Oracle Grid Infrastructure. In this section we focus on this integrated version of the CHM/OS.

The CHM tool is installed to the Oracle Grid Infrastructure home and is activated by default in Grid Infrastructure 11.2.0.2 and later for Linux and Solaris and 11.2.0.3 and later for AIX and Windows. CHM consists of two services: `osysmond` and `ologgerd`. `osysmond` runs on every node of the cluster to monitor and collect the OS metrics and send the data to the cluster logger services. `ologgerd` receives the information from all the nodes and stores the information in the CHM Repository. `ologgerd` runs in one node as the master service and in another node as a standby if the cluster has more than one node. If the master cluster logger service fails, the standby takes over as the master service and selects a new node for standby. The following example shows the two processes, `osysmond.bin` and `ologgerd`:

```
$ ps -ef | grep  -E 'osysmond|ologgerd' | grep -v grep
root      3595     1  0 Nov14 ?        01:40:51 /u01/app/11.2.0/grid/bin/ologgerd -m k2r720n1 -r -d
/u01/app/11.2.0/grid/crf/db/k2r720n2
root      6192     1  3 Nov08 ?        1-20:17:45 /u01/app/11.2.0/grid/bin/osysmond.bin
```

The preceding `ologgerd` daemon uses `'-d /u01/app/11.2.0/grid/crf/db/k2r720n2',` which is the directory where the CHM repository resides. The CHM repository is a Berkeley DB-based database stored as *.bdb files in the directory. This directory requires 1GB of disk space per node in the cluster.

```
$ pwd
/u01/app/11.2.0/grid/crf/db/k2r720n2
$ ls *.bdb
crfalert.bdb  crfclust.bdb  crfconn.bdb  crfcpu.bdb  crfhosts.bdb  crfloclts.bdb  crfts.bdb
repdhosts.bdb
```

Oracle Clusterware 12cR1 has enhanced the CHM by providing a highly available server monitor service and also support for the Flex Cluster architecture. The CHM in Oracle Clusterware 12cR1 consists of three components:

- `osysmond`
- `ologgerd`
- Oracle Grid Infrastructure Management Repository

The System Monitor Service process (osysmon) runs on every node of the cluster. The System Monitor Service monitors the OS and cluster resource–related degradation and failure and collects the real-time OS metric data and sends these data to the cluster logger service.

Instead of running on every cluster node as in Oracle Clusterware 11gR2, there is only one cluster logger service per every 32 nodes in Oracle Clusterware 12cR1. For high availability, this service will be restarted in another node if this service fails.

On the node that runs both osysmon and ologgerd:

```
grid@knewracn1]$ ps -ef | grep  -E 'osysmond|ologgerd' | grep -v grep
root      4408     1  3 Feb19 ?        08:40:32 /u01/app/12.1.0/grid/bin/osysmond.bin
root      4506     1  1 Feb19 ?        02:43:25 /u01/app/12.1.0/grid/bin/ologgerd -M -d
/u01/app/12.1.0/grid/crf/db/knewracn1
```

On other nodes that run only osysmon:

```
[grid@knewracn2 product]$ ps -ef | grep  -E 'osysmond|ologgerd' | grep -v grep
root      7995     1  1 Feb19 ?        03:26:27 /u01/app/12.1.0/grid/bin/osysmond.bin
```

In Oracle Clusterware 12cR1, all the metrics data that the cluster logger service receives are stored in the central Oracle Grid Infrastructure Management Repository (the CHM repository), which is a new feature in 12c Clusterware. The repository is configured during the installation or upgrade to Oracle Clusterware by selecting the "Configure Grid Infrastructure Management Repository" option in Oracle Universal Installer (OUI), as shown in Figure 2-6.

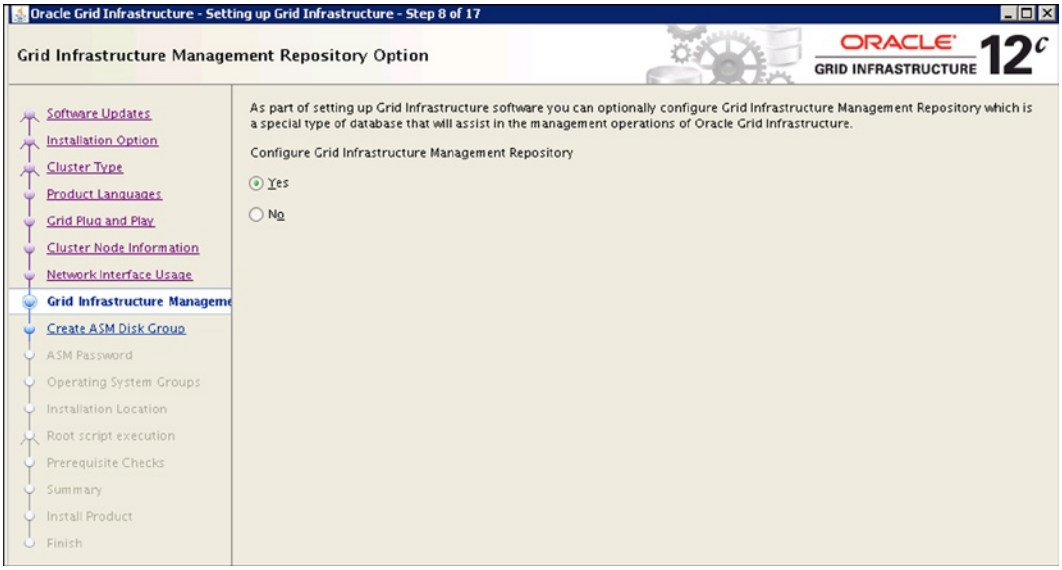


Figure 2-6. *Configure Grid Infrasturecture Management Repository in OUI*

This repository is an Oracle database. Only one node runs this repository in a cluster. If the cluster is a Flex Cluster, this node must be a hub node. Chapter 4 will discuss the architecture of Oracle Flex Clusters and different types of cluster nodes in a Flex Cluster.

To reduce the private network traffic, the repository database (MGMTDB) and the cluster logger service process (osysmon) can be located to run on the same node as shown here:

```
$ ps -ef | grep -v grep | grep pmon | grep MGMTDB
grid     31832     1  0 Feb20 ?        00:04:06 mdb_pmon_-MGMTDB

$ ps -ef | grep -v grep | grep  'osysmon'
root      2434     1  1 Feb 20 ?        00:04:49 /u01/app/12.1.0/grid/bin/osysmond.bin
```

This repository database runs under the owner of the Grid Infrastructure, which is the "grid" user in this example. The database files of the CHM repository database are located in the same diskgroup as the OCR and VD. In order to store the Grid Infrastructure repository, the size requirement of this diskgroup has been increased from the size for the OCR and VD. The actual size and the retention policy can be managed with the oclumon tool. The oclumon tool provides a command interface to query the CHM repository and perform various administrative tasks of the CHM repository. The actual size and the retention policy can be managed with the oclumon tool.

For example, we can get the repository information such as size, repository path, the node for the cluster logger service, and all the nodes that the statistics are collected from using a command like this:

```
$ oclumon manage -get repsize reppath alllogger -details

CHM Repository Path = +DATA1/_MGMTDB/DATAFILE/sysmgmtdata.260.807876429
CHM Repository Size = 38940
Logger = knewracn1
Nodes = knewracn1,knewracn2,knewracn4,knewracn7,knewracn5,knewracn8,knewracn6
```

The CHM admin directory $GRID_HOME/crf/admin has crf(hostname).ora, which records the information about the CHM repository:

```
cat /u01/app/12.1.0/grid/crf/admincrfknewracn1.ora
BDBLOC=default
PINNEDPROCS=osysmond.bin,ologgerd,ocssd.bin,cssdmonitor,cssdagent,mdb_pmon_-MGMTDB,kswapd0
MASTER=knewracn1
MYNAME=knewracn1
CLUSTERNAME=knewrac
USERNAME=grid
CRFHOME=/u01/app/12.1.0/grid
knewracn1 5=127.0.0.1 0
knewracn1 1=127.0.0.1 0
knewracn1 0=192.168.9.41 61020
MASTERPUB=172.16.9.41
DEAD=
knewracn1 2=192.168.9.41 61021
knewracn2 5=127.0.0.1 0
knewracn2 1=127.0.0.1 0
knewracn2 0=192.168.9.42 61020
ACTIVE=knewracn1,knewracn2,knewracn4
HOSTS=knewracn1,knewracn2,knewracn4
knewracn5 5=127.0.0.1 0
knewracn5 1=127.0.0.1 0
knewracn4 5=127.0.0.1 0
knewracn4 1=127.0.0.1 0
```

```
knewracn4 0=192.168.9.44 61020
knewracn8 5=127.0.0.1 0
knewracn8 1=127.0.0.1 0
knewracn7 5=127.0.0.1 0
knewracn7 1=127.0.0.1 0
knewracn6 5=127.0.0.1 0
knewracn6 1=127.0.0.1 0
```

You can collect CHM data on any node by running the `diagcollection.pl` utility on that node as a privileged user root. The steps are as follows:

First, find the cluster node where the cluster logger service is running:

```
$/u01/app/12.1.0/grid /bin/oclumon manage -get master

Master = knewracn1
```

Log in to the cluster node that runs the cluster logger service node as a privileged user (in other words, the root user) and run the `diagcollection.pl` utility. This utility collects all the available data stored in the CHM Repository. You can also specify the specific time and duration to collect the data:

```
[root@knewracn1 ~]# /u01/app/12.1.0/grid/bin/diagcollection.pl -collect -crshome /u01/app/12.1.0/grid
Production Copyright 2004, 2010, Oracle. All rights reserved
CRS diagnostic collection tool
The following CRS diagnostic archives will be created in the local directory.
crsData_knewracn1_20130302_0719.tar.gz -> logs,traces and cores from CRS home. Note: core files will
be packaged only with the --core option.
ocrData_knewracn1_20130302_0719.tar.gz -> ocrdump, ocrcheck etc
coreData_knewracn1_20130302_0719.tar.gz -> contents of CRS core files in text format

osData_knewracn1_20130302_0719.tar.gz -> logs from operating system
Collecting crs data
/bin/tar: log/knewracn1/cssd/ocssd.log: file changed as we read it

Collecting OCR data
Collecting information from core files
No corefiles found
The following diagnostic archives will be created in the local directory.
acfsData_knewracn1_20130302_0719.tar.gz -> logs from acfs log.
Collecting acfs data
Collecting OS logs
Collecting sysconfig data
```

This utility creates two .gz files, `chmosData_<host>timestamp.tar.gz` and `osData_<host>timestamp.tar.gz`, in the current working directory:

```
[root@knewracn1 ~]# ls -l *.gz
-rw-r--r--. 1 root root     1481 Mar  2 07:24 acfsData_knewracn1_20130302_0719.tar.gz
-rw-r--r--. 1 root root 58813132 Mar  2 07:23 crsData_knewracn1_20130302_0719.tar.gz
-rw-r--r--. 1 root root    54580 Mar  2 07:24 ocrData_knewracn1_20130302_0719.tar.gz
-rw-r--r--. 1 root root    18467 Mar  2 07:24 osData_knewracn1_20130302_0719.tar.gz
```

These .gz files include various log files that can be used for the diagnosis of your cluster issues.

You also can use the OCLUMON command-line tool to query the CHM repository to display node-specific metrics for a specified time period. You also can print the durations and the states for a resource on a node during a specified time period. The states can be based on predefined thresholds for each resource metric and are denoted as red, orange, yellow, and green, in decreasing order of criticality. OCLUMON command syntax is as follows:

```
 $oclumon dumpnodeview [[-allnodes] | [-n node1 node2] [-last "duration"] |
[-s "time_stamp" -e "time_stamp"] [-v] [-warning]] [-h]
-s indicates the start timestamp and –e indicates the end timestamp
```

For example, we can run the command like this to write the report into a text file:

```
$GRID_HOME/bin/oclumon dumpnodeview -allnodes -v -s "2013-03-0206:20:00" -e "2013-03-0206:30:00">
/home/grid/chm.txt
```

A segment of /home/grid/chm.txt looks like this:

```
$less  /home/grid/chm.txt

----------------------------------------
Node: knewracn1 Clock: '13-03-02 06.20.04' SerialNo:178224
----------------------------------------

SYSTEM:
#pcpus: 1 #vcpus: 2 cpuht: Y chipname: Intel(R) cpu: 7.97 cpuq: 2 physmemfree: 441396 physmemtotal:
5019920 mcache: 2405048 swapfree: 11625764 swaptotal: 12583912 hugepagetotal: 0 hugepagefree: 0
hugepagesize: 2048 ior: 93 iow: 242 ios: 39 swpin: 0 swpout: 0 pgin: 90 pgout: 180 netr: 179.471
netw: 124.380 procs: 305 rtprocs: 16 #fds: 26144 #sysfdlimit: 6815744 #disks: 5 #nics: 4 nicErrors: 0

TOP CONSUMERS:
topcpu: 'gipcd.bin(4205) 5.79' topprivmem: 'ovmd(719) 214072' topshm: 'ora_ppa7_knewdb(27372)
841520' topfd: 'ovmd(719) 1023' topthread: 'crsd.bin(4415) 48'

CPUS:
cpu0: sys-4.94 user-3.10 nice-0.0 usage-8.5 iowait-10.93
cpu1: sys-5.14 user-2.74 nice-0.0 usage-7.88 iowait-4.68

PROCESSES:

name: 'ora_smco_knewdb' pid: 27360 #procfdlimit: 65536 cpuusage: 0.00 privmem: 2092 shm: 17836 #fd:
26 #threads: 1 priority: 20 nice: 0 state: S
name: 'ora_gtx0_knewdb' pid: 27366 #procfdlimit: 65536 cpuusage: 0.00 privmem: 2028 shm: 17088 #fd:
26 #threads: 1 priority: 20 nice: 0 state: S
```

name: 'ora_rcbg_knewdb' pid: 27368 #procfdlimit: 65536 cpuusage: 0.00 privmem: .....

RAC Database Hang Analysis

In this section, we will explore the conceptual basis for invoking and interpreting a hang analysis dump to diagnose a potential RAC database hung/slow/blocking situation. When a database either is running unacceptably slow, is hung due to an internal system developed interdependence deadlock or a latch causing database hung/slowness,

or else a prolonged deadlock/block hurts overall database performance, it is advisable to perform a hang analysis, which helps greatly in identifying the root cause of the problem. The following set of examples explains how to invoke and use the hang analysis:

```
 SQL> sqlplus / as sysdba
 SQL> oradebug setmypid
 SQL> oradebug unlimit
 SQL> oradebug setinst all         -- enables cluster-wide hang analysis
 SQL> oradebug –g all hanganalyze 3   --is the most commonly used level
<< wait for couple of minutes >>
SQL> oradebug –g all hanganalyze 3
```

The hang analysis levels can be the currently set value between 1 to 5 and 10. When hanganlyze is invoked, the diagnostic information will be written to a dump file under $ORACLE_BASE/diag/rdbms/dbname/instance_name/trace, which can be used to troubleshoot the problem.

We have built the following test case to develop a blocking scenario in a RAC database to demonstrate the procedure practically. We will then interpret the trace file to understand the contents to troubleshoot the issue. The following steps were performed as part of the test scenario:

Create an EMP table:

```
SQL> create table emp (eno number(3),deptno number(2), sal number(9));
```

Load a few records in the table. From instance 1, execute an update statement:

```
SQL> update emp set sal=sal+100 where eno=101; -- not commit performed
```

From instance 2, execute an update statement for the same record to develop a blocking scenario:

```
SQL> update emp set sal=sal+200 where eno=101;
```

At this point, the session on instance 2 is hanging and the cursor doesn't return to the SQL prompt, as expected.

Now, from another session, run the hang analysis as follows:

```
SQL>oradebug setmypid
Statement processed.
SQL >oradebug setinst all
Statement processed.
SQL >oradebug -g all hanganalyze 3 <level 3 is most suitable in many circumstances>
Hang Analysis in /u00/app/oracle/diag/rdbms/rondb/RONDB1/trace/RONDB1_diag_6534.trc
```

Let's have a walk-through and interpret the contents of the trace file to identify the blocker and holder details in context. Here is the excerpt from the trace file:

```
Node id: 1
List of nodes: 0, 1,       << nodes (instance) count >>

*** 2012-12-16 17:19:18.630
================================================================================
HANG ANALYSIS:
  instances (db_name.oracle_sid): rondb.rondb2, rondb.rondb1
  oradebug_node_dump_level: 3        << hanganlysis level >>
```

```
  analysis initiated by oradebug
  os thread scheduling delay history: (sampling every 1.000000 secs)
    0.000000 secs at [ 17:19:17 ]
      NOTE: scheduling delay has not been sampled for 0.977894 secs    0.000000 secs from
[ 17:19:14 - 17:19:18 ], 5 sec avg
    0.000323 secs from [ 17:18:18 - 17:19:18 ], 1 min avg
    0.000496 secs from [ 17:14:19 - 17:19:18 ], 5 min avg
================================================================================
Chains most likely to have caused the hang:
 [a] Chain 1 Signature: 'SQL*Net message from client'<='enq: TX - row lock contention'
     Chain 1 Signature Hash: 0x38c48850

================================================================================
Non-intersecting chains:

--------------------------------------------------------------------------------
Chain 1:
--------------------------------------------------------------------------------
    Oracle session identified by:       << waiter >>
    {
                instance: 2 (rondb.rondb2)
                   os id: 12250
              process id: 40, oracle@hostname (TNS V1-V3)
              session id: 103
          session serial #: 1243
    }
    is waiting for 'enq: TX - row lock contention' with wait info:
    {
                      p1: 'name|mode'=0x54580006
                      p2: 'usn<<16 | slot'=0x20001b
                      p3: 'sequence'=0x101fc
             time in wait: 21.489450 sec
            timeout after: never
                  wait id: 33
                 blocking: 0 sessions
              current sql: update emp set sal=sal+100 where eno=1

and is blocked by
 => Oracle session identified by:              << holder >>
    {
                instance: 1 (imcruat.imcruat1)
                   os id: 8047
              process id: 42, oracle@usdbt42 (TNS V1-V3)
              session id: 14
          session serial #: 125
    }
    which is waiting for 'SQL*Net message from client' with wait info:
    {
                      p1: 'driver id'=0x62657100
                      p2: '#bytes'=0x1
             time in wait: 27.311965 sec
```

```
           timeout after: never
                 wait id: 131
                blocking: 1 session

*** 2012-12-16 17:19:18.725

State of ALL nodes
([nodenum]/cnode/sid/sess_srno/session/ospid/state/[adjlist]):
[102]/2/103/1243/c0000000e4ae9518/12250/NLEAF/[262]
 [262]/1/14/125/c0000000d4a03f90/8047/LEAF/
*** 2012-12-16 17:19:47.303
================================================================================
HANG ANALYSIS DUMPS:
  oradebug_node_dump_level: 3
================================================================================

State of LOCAL nodes
([nodenum]/cnode/sid/sess_srno/session/ospid/state/[adjlist]):
[102]/2/103/1243/c0000000e4ae9518/12250/NLEAF/[262]

================================================================================
END OF HANG ANALYSIS
================================================================================
```

In the preceding example, `SID 102` on instance 2 is blocked by `SID 261` on instance 1. Upon identifying the holder, either complete the transaction or abort the session to release the lock from the database.

It is sometimes advisable to have the `SYSTEMSTATE` dump along with the `HANGANALYSIS` to generate more detailed diagnostic information to identify the root cause of the issue. Depending upon the level that is used to dump the `SYSTEMSTATE`, the cursor might take a very long time to return to the SQL prompt. The trace file details also can be found in the `database alter.log` file.

You shouldn't be generating the `SYSTEMSTATE` dump under normal circumstances; in other words, unless you have some serious issues in the database or are advised by Oracle support to troubleshoot some serious database issues. Besides, the `SYSTEMSTATE` tends to generate a vast trace file, or it can cause an instance crash under unpredictable circumstances.

Above all, Oracle provides the `HANGFG` tool to automate the collection of `systemstate` and `hang analysis` for a non-RAC and RAC database environment. You need to download the tool from `my_oracle_suport` (previously known as `metalink`). Once you invoke the tool, it will generate a couple of output files, named `hangfiles.out` and `hangfg.log`, under the `$ORACLE_BASE/diag/rdbms/database/instance_name/trace` location.

Summary

This chapter discussed the architecture and components of the Oracle Clusterware stack, including the updates in Oracle Clusterware 12cR1. We will talk about some other new Oracle Clusterware features introduced in Oracle 12cR1 in Chapter 4.

This chapter also discussed tools and tips for Clusterware management and troubleshooting. Applying the tools, utilities, and guidelines described in this chapter, you can diagnose many serious cluster-related issues and address Clusterware stack startup failures. In addition, you have learned how to modify the default tracing levels of various Clusterware daemon processes and their subcomponents to obtain detailed debugging information to troubleshoot various cluster-related issues. In a nutshell, the chapter has offered you all essential cluster management and troubleshooting concepts and skills that will help you in managing a medium-scale or large-scale cluster environment.

CHAPTER 3

RAC Operational Practices

by Riyaj Shamsudeen

When running a RAC cluster, some operational practices can reduce overall administration costs and improve manageability. The design of a RAC cluster is critical for the effective implementation of these practices. While this chapter discusses numerous RAC design patterns, implementation of these design practices will lead to better operational results.

Workload Management

Workload management is an essential tool that simplifies the management of RAC clusters. Skillful use of workload management techniques can make the administration of a production cluster much easier. It can also speed up the detection of failures and accelerate the failover process. *Services, VIP listeners,* and *SCAN listeners* play important roles in workload management.

Workload management is centered on two basic principles:

1. Application affinity: Optimal placement of application resource usage to improve the performance of an application.
2. Workload distribution: Optimal distribution of resource usage among available nodes to maximize the use of cluster nodes.

Application affinity is a technique to keep intensive access to database objects localized to improve application performance. Latency to access a buffer in the local buffer cache is on the order of microseconds (if not nanoseconds with recent faster CPUs), whereas latency to access a buffer resident in a remote SGA is on the order of milliseconds, typically, 1–3ms.[1] Disk access latency is roughly 3 to 5ms for single block reads; that is nearly the same as that for remote buffer cache access latency, whereas local buffer cache access is orders of magnitude faster. So, if an application or application components (such as a group of batch programs) access some objects aggressively, connect those application components to an instance so that object access is mostly localized, improving application performance.

A complementary (though not necessarily competing) technique to application affinity is *workload distribution*. For example, if there are four nodes in a cluster, then you would want to distribute workload evenly among all four nodes so that a node is not overloaded. *Service* is a key tool to achieving workload distribution evenly among all nodes of a cluster.

[1] In Exadata platforms, the remote buffer cache access latency is about 0.5ms. Both faster CPUs and infiniband fabric hardware provide lower latency.

In practice, application affinity and workload distribution coexist. Some application components use application affinity to improve performance, and a few application components use workload distribution.

In addition to *administrator-managed* databases, you can also use *policy-managed* databases for workload management. Oracle 11gR2 introduces policy-managed databases and Oracle version 12c enhances policy-managed databases for dynamic policy management. Policy-managed databases are discussed later in this chapter.

Services

A service is a logical abstraction designed to group application components for optimal workload management. By implementing a service-based connection design, the application workload can be affinitized, distributed, or both. A database connection string specifies a service as a connection property, and the cluster processes distribute new connections to an instance depending on the workload specification of the service. The performance metrics of a service are also used by the listener processes to redirect the connection. Further, database processes calculate, maintain, and propagate service metrics to facilitate workload distribution and failover techniques.

A typical database connection using a service is done according to the following steps:

1. Application process sends a connection request specifying a service_name to the SCAN (Single Client Access Name) listener.
2. SCAN listener identifies the instance that can provide the best quality of service.
3. SCAN listener redirects the connection to the VIP (virtual IP address) listener listening locally for that instance.
4. VIP listener creates a database connection process, and the application continues processing the data by communicating to the connection process.

By default, every database has some default services created, as shown in the following query output. Default services are marked as preferred in all instances; therefore, default services are not useful for workload management. For example, a service SOLRAC matching the database name (db_name initialization parameter) is created, and that service is preferred in all instances of the database. So, workload distribution among the instances of the database is not effective using the SOLRAC service since connecting through that service can connect to any instance.

```
SQL> select name, goal from v$active_services

NAME                           GOAL
------------------------------ ------------
SOLRAC                         NONE
SYS$BACKGROUND                 NONE
SYS$USERS                      NONE
```

You need to create a new service to implement workload management properly. Only a service created by you can be used effectively for workload management. There are many ways to create a service, such as using the *srvctl* command, the *dbms_service* package call, or Oracle Enterprise Manager (OEM). In a single-instance database, you can use the dbms_service package to create and manipulate a service. In RAC, you should not use the dbms_service package call. Instead, it is better to use the srvctl command to create a service. The srvctl command creates a cluster resource in Clusterware, in addition to creating a service in the database. This cluster resource for a service is monitored for availability, and the resource is failed over in cases of instance failure.

The following command adds a new service: po to solrac database. This command also specifies that this service should be preferred in solrac1 and solrac3 instances, and the service should fail over to solrac2 and solrac4 instances if a preferred instance fails. Flag –r specifies *preferred* instances, and flag –a specifies *available* instances.

```
$ srvctl add service -d solrac -s po -r solrac1,solrac3 \
     -a solrac2,solrac4 -m BASIC -j short -B service_time
```

In Oracle Database version 12c, the syntax to add a service is different and the options are more verbal. While the command options in 11.2 will also work in version 12c, the abbreviated command options are deprecated in version 12c. Earlier version 11.2 commands to add a service can be rewritten as follows using version 12c syntax.

```
$ srvctl add service -db solrac -service po
   -preferred solrac1,solrac3 \
   -available solrac2,solrac4  -failovermethod BASIC \
   -clbgoal short -rlbgoal service_time
```

Adding a service using the srvctl command adds a resource in Clusterware, too. Whereas in 11.2 versions, the user has to start the service manually after adding a service, in version 12c, the service is automatically started. This cluster resource is a dependent of database and VIP resources. So, as soon as the database and VIP resources are online, the service resource will be altered online.

```
$ crsctl stat res ora.solrac.po.svc -p |more
NAME=ora.solrac.po.svc
TYPE=ora.service.type
...
SERVICE_NAME=PO
START_DEPENDENCIES=hard(ora.solrac.db,type:ora.cluster_vip_net1.type)
weak(type:ora.listener.type) pullup(type:ora.cluster_vip_net1.type) pullup:always(ora.solrac.db)
```

Starting the database resource will recursively start dependent Clusterware resources. As the services are defined as dependent resources, starting the Clusterware resource of a service enables the service in the database by implicitly calling the dbms_service package.

You could explicitly start a service using the srvctl command, as well. Query the view *v$active_services* to see that a new service has been created and started.

```
$ srvctl start service –d solrac –s po
SQL> select name, creation_date from v$active_services
     where name='PO';
NAME                                     CREATION_DATE
---------------------------------------- --------------
PO                                       17-DEC-12
```

Service Metrics

The MMON (Memory Monitor) database background process calculates the service-level metrics using service statistics, and these metrics are visible at v$activeservice dynamic performance view. Metrics are calculated at intervals of 5 seconds and 1 minute. In the following output, the first line shows the metrics calculated at 1-minute intervals, and the second line shows the metrics calculated at 5-second intervals.

```
SQL> select inst_id, begin_time, end_time from gv$servicemetric
     where service_name='PO' order by 1, 2;
```

```
INST BEGIN_TIME                    END_TIME
---- ------------------------- -------------------------
   1 14-JAN-2013 16:05:57       14-JAN-2013 16:06:57
   1 14-JAN-2013 16:07:06       14-JAN-2013 16:07:11
...
```

Two sets of metrics are calculated by the MMON background process, one indicating service_time and the other indicating throughput. Depending on the service configuration, goodness of a service is calculated and PMON (Process Monitor) or LREG (Listener Registration - 12c) background processes communicate the service metrics to the SCAN and VIP listeners.

Figure 3-1 shows the MMON propagation details. It gives an overview of service metrics calculation and propagation.

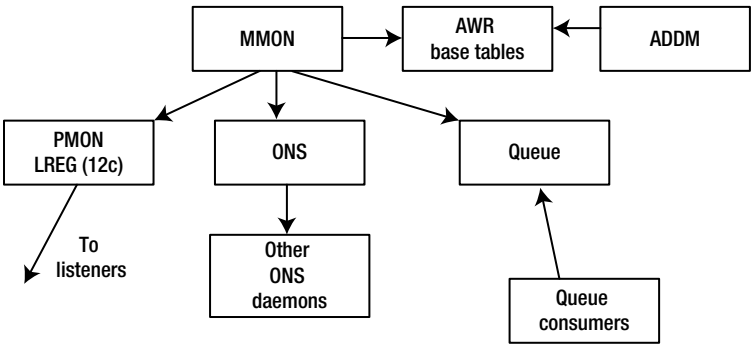


Figure 3-1. *Service metrics calculation and propagation*

5. MMON process populates service metrics in AWR (Automatic Workload Repository) base tables.[2]
6. Service metrics are propagated to the ONS (Oracle Notification Service) daemon, and ONS daemons propagate the service metrics to the client-side driver and client-side ONS daemon.
7. In addition, service metrics are enqueued in the sys$service_metrics queue. A client program can register as a subscriber to the queue, receive queue updates, and take necessary action based on the queue alerts.

Also, MMON propagation is a key component of the FAN (Fast Application Notification) feature discussed later in this chapter.

A set of columns, namely, elapsedpercall, cpupercall, and dbtimepercall, indicate the response time quality of the service. The following data from v$servicemetric shows that response time metrics are better in instance 1 than in instance 2, as dbtimepercall is lower in instance 1 than in instance 2. Using these statistics, the SCAN listener redirects the connection to an instance that can provide better quality of service.

```
SQL> select inst_id,service_name, intsize_csec, elapsedpercall,
     cpupercall,dbtimepercall
     from gv$servicemetric where service_name='PO'
     order by inst_id, 2,3;
```

[2] AWR report also prints the service-level metrics.

```
INST SERVICE    INTSIZE_CSEC ELAPSEDPERCALL CPUPERCALL DBTIMEPERCALL
---- ---------- ------------ -------------- ---------- -------------
   1 PO                  511     45824.8171 19861.5611    45824.8171
   1 PO                 5979     32517.4693 13470.9456    32517.4693
   2 PO                  512     89777.4994 41022.0318    89777.4994
   2 PO                 5938      50052.119 23055.3317     50052.119
...
```

Columns callspersec and dbtimepersec indicate the throughput metrics of a service.

```
SQL> select inst_id,service_name, intsize_csec, callspersec,
     dbtimepersec
     from gv$servicemetric where service_name='PO'
     order by inst_id, 2,3;

INST SERVICE    INTSIZE_CSEC CALLSPERSEC DBTIMEPERSEC
---- ---------- ------------ ----------- ------------
   1 PO                  512  1489.64844   4099.97602
   1 PO                 6003  1200.56638     4068.117
...
```

Note that these metrics are indicative of an average of performance statistics of all the workloads connecting to that service in an instance. As averages can be misleading, it is important to understand that in a few cases listeners can redirect the connections to instances that may not be optimal for that service.

Load Balancing Goals

Services can be designed with specific goals, to maximize either response time or throughput. Service metrics are factored to identify best-instance matching with the service goals. Two attributes of a service definition specify the service goal; namely, GOAL and CLB_GOAL.

- Attribute GOAL can be set to a value of either LONG or SHORT. The value of LONG implies that the connections are long-lived, such as connections originating from a connection pool. When the attribute GOAL is set to LONG, connections are evenly distributed among available instances, meaning that the listener tries to achieve an equal number of connections in all instances where services are available. An attribute value of SHORT triggers runtime load balancing based upon performance metrics.
- If the attribute GOAL is set to SHORT, then only the CLB_GOAL attribute is applicable. If the attribute CLB_GOAL is set to SERVICE_TIME, then connections are redirected to the instance that can provide better response time for that service. If CLB_GOAL is set to THROUGHPUT, then connections are redirected to the instance that can provide better throughput for that service.

If the attribute CLB_GOAL is set to response_time, then service metrics with names ending with percall are considered to measure goodness of an instance for a service. If the attribute CLB_GOAL is set to THROUGHPUT, then metrics ending with persec are considered to measure goodness of an instance for a service. Table 4-1 summarizes these service attributes and offers a short recommendation.

Table 4-1. *Service Goals*

GOAL	CLB_GOAL	Service metrics
LONG	Ignored	Suitable for long-lived connections such as connection pool connections.
SHORT	Response_time	Service is redirected to an instance that can provide better response_time. Suitable for OLTP workload.
SHORT	Throughput	Service is redirected to an instance that can provide better throughput. Suitable for DSS/Warehouse workload.

The MMON process calculates `goodness` and `delta` columns using service metrics. A lower value for the goodness column indicates quality of service in that instance (in retrospect, it should have been called a badness indicator). The delta column indicates how the value of the goodness column will change if a new connection is added to that service. In this example output, PO service is defined with the GOAL attribute set to the LONG value, and so the goodness column simply indicates the number of connections to that service in that instance. Since instances 2 and 3 have lower values for the goodness column, a new connection to PO service will be redirected to instance 2 or 3.

```
SQL> select inst_id, service_name, intsize_csec,goodness,delta
     from gv$servicemetric
     where service_name='PO' order by inst_id, 2,3;

INST SERVICE    INTSIZE_CSEC   GOODNESS      DELTA
---- ---------- ------------ ---------- ----------
   1 PO                  514       1883          1
   1 PO                 5994       1883          1
   2 PO                  528       1820          1
   2 PO                 5972       1820          1
```

In versions 11.2 and earlier, the PMON process propagates the service metrics to the listeners registered in `local_listener` and `remote_listener` initialization parameters. As the remote_listener specifies the address of SCAN listeners and the local_listener parameter specifies the address of the VIP listener, the PMON process propagates the service metrics to both SCAN and VIP listeners. You can trace listener registration using the following command.

```
alter system set events='immediate trace name listener_registration level 15';
```

The PMON trace file shows how the quality of service is propagated to listener processes. Here is an example (output of PMON process sending the service quality to the listener process).

```
kmmgdnu: po
goodness=100, delta=10,
flags=0x4:unblocked/not overloaded, update=0x6:G/D/-
```

In version 12c, listener registration is performed by a new mandatory background process named LREG. At database start, the LREG process polls the listeners specified in the local_listener and remote_listener parameters to identify whether the listeners are running. If the listeners are available, then the services are registered to the listeners by the LREG process. The earlier command to trace listener registration in version 11.2 is applicable to version 12c also, but the listener registration details are written to an LREG trace file instead of a PMON trace file.

```
Unix process pid: 3196, image: oracle@oel6rac1.example.com (LREG)
...
Services:
...
  2 - po.example.com
       flg=0x4, upd=0x6
       goodnes=0, delta=1
...
Listen Endpoints:
0 -(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=oel6rac1.example.com)(PORT=5500))
       (Presentation=HTTP)(Session=RAW))
       flg=0x80000000, nse=0, lsnr=, lflg=0x13
       pre=HTTP, sta=0
```

If a service is not registered properly to the listeners or if the service quality decisions taken by the listeners are incorrect, then you can trace the service registration to check if the listeners received correct service metrics.

Version 11.2 introduces the Valid Node Checking for Registration feature. SCAN listeners will accept registration from any IP address in the same subnet as the SCAN IP address. The VIP listener will accept the registration only from the local node. You can use invited_nodes to specify the permitted nodes for listener registration.

Runtime Failover

At runtime, a connection can be configured to fail over quickly to an available instance by configuring the failover type service attribute. A service supports failover types of NONE, SESSION, or SELECT. Failover type NONE does not provide any failover. If the failover type is set to SESSION, then after a node failure, sessions will fail over to surviving instances and the application must restart all work, including in-transit work. If the failover type is SELECT, then cursor fetch details are remembered by the client-side JDBC/OCI driver, the SELECT statement is restarted, and FETCH continues from the last row fetched before the failure.

In version 11.2, while the SELECT failover type supports a transparent restart of queries, it doesn't restart any DML operation. The application must restart DML statements, and failures in DML statements must be handled by the application.

Version 12c introduces `TRANSACTION`, a new failover type, and this failovertype attribute value is specific to a new feature called Application Continuity. Application Continuity also captures DML changes and replays the DML activity using the Oracle client-side replay driver. The Application Continuity feature is discussed later in this chapter.

In version 12c, command options are specified more verbally. For example, in 11.2, option -e specifies the failover type, and in version 12c, service attribute `failovertype` specifies failover type.

Service in Second Network

A service can be configured to operate in a subnet. For example, to specify a service to be available in only the second network, you would specify option –k 2 while creating a service. In version 12c, clause -netnum 2 can be used to specify that the service will operate only in the second network. Please refer to Chapter 9 to learn more about the second network in a RAC cluster.

Guidelines for Services

The following guidelines are useful while creating services to improve cluster management.

- Create a separate service for each major application component. Service is an excellent method to implement application affinity. For example, if PO application accesses numerous PO-related tables and if one instance is sufficient to service the PO application, then keep PO service contained in one instance. You can keep the PO service connecting to an instance by creating the service with just a preferred instance. Application affinity improves performance by reducing global cache latency.
- Design the service in such a way that both preferred and available instances are configured. While designing services, consider the scenario of each node shutting down and construct the service placement based upon those scenarios.
- Use policy-managed databases instead of administrator-managed databases if there are numerous nodes (12+) in your cluster.
- Use of SCAN listeners and services reduces administration overhead. Especially for sites upgrading from versions 11.1 and earlier, it is important to alter the connection string to use SCAN listeners and SCAN IP addresses. It is very hard to change the connection string post-upgrade.
- Specify optimal clbgoal and rlbgoal parameters matching the application workload. Service metrics are calculated as an average for all the workloads connecting to that service. So, if your application has different workloads and if it is possible to segregate the application components depending upon the workload characteristics, then use disjointed, unique services for those application components. This segregation of application workload to different services improves the accuracy of service metrics and improves the accuracy of load balancing decisions. This segregation also improves the ability to identify application components consuming resources quickly.

There are only a few minor pitfalls in creating a new service. Production clusters with 100+ services operate without any major side effects. While we are not recommending that you create hundreds of unnecessary services, you should create as many services as you need to split the applications into manageable, disjointed workloads. In version 11.2, if there are numerous services (100+) and numerous listeners, then there is a possibility that the PMON process might spend more time on service registration to listeners due to the sheer number of services and listeners. But, in version 12c, this possibility is eliminated as the LREG parameter performs service registration and PMON is freed from listener registration.

SCAN and SCAN Listeners

Introduced in version 11.2, SCAN (Single Client Access Name) is an effective tool to enhance workload management in RAC, to eliminate complexity in connection string management, and to provide faster connect time/runtime failover. Further, the SCAN feature simplifies connection management by providing a logical abstraction layer of the cluster topology. Cluster topology changes do not trigger changes to connect strings.

Consider an environment with numerous applications and users who are connecting to a RAC database with three nodes in the cluster, using the following connect string:

```
PO =
  (DESCRIPTION=
   (ADDRESS_LIST=
     (LOAD_BALANCE=YES)
     (FAILOVER=YES)
```

```
(ADDRESS=(PROTOCOL=tcp)(HOST=rac1.example.com)(PORT=12000))
(ADDRESS=(PROTOCOL=tcp)(HOST=rac2.example.com)(PORT=12000))
(ADDRESS=(PROTOCOL=tcp)(HOST=rac3.example.com)(PORT=12000))
    )
    (CONNECT_DATA=
        (SERVICE_NAME=PO)
      )
    )
```

The preceding connect string has some drawbacks:

1. If we add another node to the cluster, say rac4.example.com, then connect strings must be modified in numerous application configuration files. Worse, topology changes might trigger changes to the tnsnames.ora file in thousands of user computers. This type of configuration change requires seamless coordination.

2. Similarly, if a node is removed from the RAC cluster, then connect strings must be altered to eliminate unnecessary waits for a TCP timeout while creating new connections. For example, consider that the rac1.example.com node is shut down for a prolonged period of time. As connection strings can be configured to try to connect to all VIP listeners by specifying load_balance=ON in the connection string, connection attempts will eventually succeed, after a brief TCP timeout period. Nevertheless, there may be a delay induced by TCP timeout duration. This connection delay is visible in applications that open new connections aggressively.

3. Without SCAN listeners, the connect time load balancing decision is performed by VIP listeners. The use of VIP listeners for workload management is not optimal, as the load balancing decision is not centralized, and so VIP listeners can redirect connections inaccurately. VIP listeners do not use all service attributes for load balancing either.

SCAN listeners resolve these connect time issues and improve load balancing accuracy. SCAN listeners act as a forefront of connection management, applications connect to SCAN listeners, and SCAN listeners act as a conduit mechanism to forward connections to an instance that can provide better quality of service. A SCAN listener identifies an instance to forward using the service metrics received from the database processes.

SCAN listeners know about topology changes. Any change in the cluster topology, such as addition of a node, deletion of a node, etc., is propagated to SCAN listeners immediately. Application connects only to the SCAN listeners, and so cluster topology changes have no effect on application connect strings.

The following is an example of a connect string utilizing the SCAN listener feature; this connect string specifies the DNS alias of the SCAN IP address (also known as SCAN name) and the port number of SCAN listeners. In this example, rac-scan.example.com, is the SCAN name and the listener listens on port 25000.

```
PO =
(DESCRIPTION=
 (ADDRESS_LIST=
   (ADDRESS=(PROTOCOL=tcp)(HOST=rac-scan.example.com)(PORT=25000))
     )
    (CONNECT_DATA=(SERVICE_NAME=PO))
   )
```

A new connection request using the preceding connect string is made according to the following high-level steps. Figure 3-2 shows an example of a new connection to the SCAN listener.

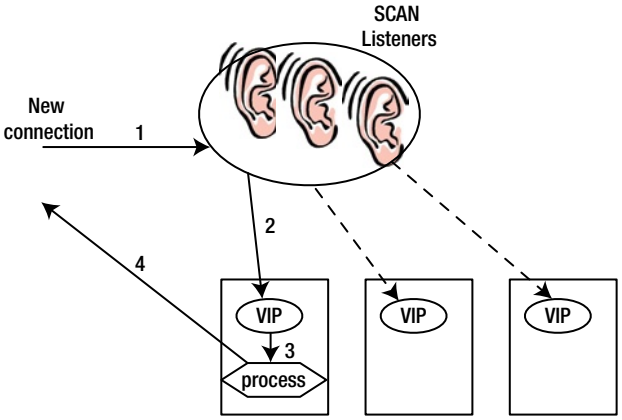


Figure 3-2. *SCAN listeners*

1. Application connection or a process queries the DNS server to identify the IP addresses associated with SCAN (`rac-scan.example.com` in this example). Note that as DNS name to IP address mapping can already be cached in a local DNS cache, the DNS query might result in a local DNS cache lookup instead of a query to the DNS server.
2. DNS server responds with an IP address associated with the DNS alias.
3. The process sends a connection request to the SCAN IP address and the PORT number providing SERVICE_NAME for the connection request.
4. SCAN listeners identify instances that can provide better quality of service for that service using service metrics. The connection is forwarded to a VIP listener local to the chosen instance.
5. A VIP listener creates a connection process (dedicated server mode), and the connection process attaches to the database, and the user process continues processing the data communicating with that connection process.

In a RAC cluster, up to three SCAN IP addresses can be configured. Since a SCAN listener is a simple redirection mechanism (and hence a lightweight process), SCAN listeners can manage spurious connection requests without any delay.

SCAN IP addresses must be mapped to *a* SCAN DNS alias in the DNS server or GNS (Grid Naming Service). It is recommended that the DNS server be configured to retrieve SCAN IP addresses in a round-robin fashion. IP addresses retrieved in a round-robin fashion provide additional connection load balancing capability and prevent overloading of *a* SCAN listener. Following nslookup shows that three IP addresses are associated with the SCAN name.

```
$ /usr/sbin/nslookup rac-scan.example.com
...
Name:   rac-scan.example.com
Address: 10.7.11.110
Name:   rac-scan.example.com
Address: 10.7.11.111
Name:   rac-scan.example.com
Address: 10.7.11.112
```

While installing Grid Infrastructure software, you can provide the SCAN name and port number. Grid Infrastructure queries the DNS server to identify IP addresses associated with SCAN names and creates as many SCAN listener resources as there are SCAN IP addresses. For example, if DNS returns three SCAN IP addresses, then three SCAN IP resources and three SCAN listener resources are created in the Clusterware.

After the Grid Infrastructure installation, you can modify SCAN IP address and SCAN listener attributes using the srvctl command. The following configuration of SCAN IP address shows that Clusterware created three sets of SCAN IP resources and SCAN listeners for the rac-scan.example.com DNS alias.

```
$ srvctl config scan
SCAN name: rac-scan, Network: 1/10.7.11.0/255.255.254.0/eth1
SCAN VIP name: scan1, IP: /rac-scan.example.com/10.7.11.110
SCAN VIP name: scan2, IP: /rac-scan.example.com/10.7.11.111
SCAN VIP name: scan3, IP: /rac-scan.example.com/10.7.11.112
```

Enabling client-level TNS trace, you can see that the SCAN IP address retrieved is different for each connection attempt. This round-robin retrieval of SCAN IP addresses provides a load balancing mechanism among SCAN listeners.

The following lines are shown from SQL*Net trace files, and you can see that the first connection request went to the SCAN listener with IP address 10.7.11.112 and the second connection request went to the scan listener with IP address 10.7.11.110.

```
$ sqlplus apps@po
nttbnd2addr:using host IP address: 10.7.11.112
..
$ sqlplus apps@po
nttbnd2addr:using host IP address: 10.7.11.110
..
```

The initialization parameter remote_listener is set to the TNS alias of SCAN listener or EZConnect syntax using the SCAN IP address and port number. The PMON or LREG process register services and propagates service-level changes to the SCAN listener specified in the remote_listener parameter. EZConnect syntax is as follows.

```
*.REMOTE_LISTENER=rac-scan.example.com:25000
```

As a SCAN listener acts as a redirection mechanism for connection requests, reviewing services supported by a SCAN listener shows that all services supported by a SCAN listener will have a remote server keyword, indicating that the SCAN listener will redirect the connection to a VIP listener. In the following lsnrctl services command output, notice the keyword REMOTE SERVER, indicating that the LISTENER_SCAN2 listener will redirect the connection requests to the rac2 node VIP listener, thereby forwarding the connection to the RAC2 instance.

```
$ lsnrctl services LISTENER_SCAN2
LSNRCTL for Solaris: Version 11.2.0.2.0 - Production on 19-JAN-2013 16:43:26
Copyright (c) 1991, 2010, Oracle.  All rights reserved.
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=LISTENER_SCAN2)))
  Services Summary...
    Service "PO" has 1 instance(s).
      Instance "RAC2", status READY, has 1 handler(s) for this service...
Handler(s):
 "DEDICATED" established:1 refused:0 state:ready
  REMOTE SERVER
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=rac2.example.com)(PORT=12000))
       (CONNECT_DATA=(SERVICE_NAME=PO)(INSTANCE_NAME=RAC2)))
```

If a service is not visible in a SCAN listener services output, the service registration to SCAN listener(s) is incorrect. Similarly, the lsnrctl services command output of a VIP listener will print services as `LOCAL SERVER` keyword, indicating that the VIP listener will handle the connection for that service.

SCAN Listener in Second Network (12c)

From version 11.2 and later, it is possible to create a second network, and VIP listeners can be created in the second network, too. However, SCAN listeners can listen only in the default first network in version 11.2. Version 12c eliminates that restriction, and it is possible to have SCAN listeners listening on a second network also. By specifying the listener_networks parameter, SCAN listeners will redirect the connection to VIP listeners within the same network subnet.

Let us walk through an example of SCAN listener in the second network in version 12c. The following set of commands adds a network with netnumber 2 in a subnet of 192.168.8.0 (note that first network has 10.7.11.0 subnet). The next three commands add VIP addresses with –netnumber 2, specifying that these VIP listeners are associated with the second network. The fourth command adds a listener to listen in port 1521 in these node VIP addresses.

```
$ srvctl add network -netnum 2 -subnet 192.168.8.0/255.255.255.0
$ srvctl add vip -node rac1.example.com -address  rac1vip.example2.com/255.255.255.0 -netnum 2
$ srvctl add vip -node rac2.example.com -address  rac2vip.example2.com/255.255.255.0 -netnum 2
$ srvctl add vip -node rac3.example.com -address  rac3vip.example2.com/255.255.255.0 -netnum 2
$ srvctl add listener -netnum 2 -endpoints TCP:1522
```

The following commands add a SCAN name in network 2 with a SCAN name of `ebs.example2.com`. The second of the following two commands adds a SCAN listener[3] listening in port 20000, and Grid Infrastructure identifies SCAN IP addresses by querying DNS for SCAN name `ebs.example2.com`.

```
$ srvctl add scan -scanname ebs.example2.com -netnum 2
$ srvctl add scan_listener -listener LISTSCAN -endpoints TCP:20000
```

Database initialization parameter listener_networks must be modified so that the PMON or LREG process can register the services to listeners in both networks. The LISTENER_NETWORKS parameter specifies the network configuration in two parts, the first part specifying the configuration of the first network, named network1, and the second part specifying the configuration of the second network, named network2. In this example, listener_net1 and listener_net2 are TNS aliases connecting to VIP listener in local nodes; the remote_listener parameter refers to the SCAN listeners running in their respective networks specified using EZConnect syntax.

```
alter system set LISTENER_NETWORKS='((NAME=network1)(LOCAL_LISTENER=listener_net1)
(REMOTE_LISTENER=ebs.example.com:10000))','((NAME=network2)(LOCAL_LISTENER=listener_net2)
(REMOTE_LISTENER=ebs.example2.com:20000))';
```

Consider the following connection string from a user connecting to the second network (possibly a connection originating from `example2.com` domain): SCAN listeners will redirect the connection to a VIP listener in the second network only. Essentially, connection from the second network subnet will remain in the second network subnet and the network packets may not traverse from one subnet to another subnet. This network demarcation is important, as a firewall might prevent network packets from crossing over to another domain.

[3] Notice the syntax for adding scan_listener does not specify a netnum clause. This appears to be a documentation bug, so refer to public documentation to verify the syntax.

```
 PO =
  (DESCRIPTION=
     (ADDRESS_LIST=
(ADDRESS=(PROTOCOL=tcp)(HOST=ebs.example2.com)(PORT=20000)))
      (CONNECT_DATA=(SERVICE_NAME=PO)
     )
  )
```

Support of SCAN listeners in the second network is an important feature in an enterprise database cluster with multiple network segments and domains, especially after major business events such as a merger of two companies.

Guidelines for SCAN Listeners

Optimally configured SCAN IP addresses and SCAN listeners are essential production components from 11.2 and later. The SCAN feature greatly reduces administration overheads, as you can publish the TNS connection string once and that connection string is not affected by RAC cluster topology changes. As SCAN listeners are lightweight forwarders, even if you have numerous nodes in a cluster, just three SCAN listeners are able to service an onslaught of connection requests. In fact, in a busy, connection-intensive production cluster, VIP listeners are bottlenecks, while SCAN listeners are able to withstand spurious connection requests easily.

Multiple DNS servers with failover and load balancing represent an extra layer of protection to avoid a single point of failure. This load balancing between DNS servers is usually a responsibility of network administrators. If you configure GNS servers, Clusterware manages GNS daemons and VIPs. If a server running GNS daemon or VIP fails, then the Clusterware will automatically fail over the GNS daemon and VIP to a surviving instance.

In a single segment network cluster, the local_listener parameter should refer to the VIP listener running on the local node, and the remote_listener parameter should refer to the SCAN listeners using EZConnect syntax specifying the DNS alias of the SCAN IP address and port number. Do not specify remote VIP listeners from another node in the remote_listener parameter, as that can cause cross-registration and accidental connection forwarding to the other node.

Global Database Services (12c)

Oracle Database version 12c introduces the concept of Global Database Services (GDS). Services discussed so far in this chapter are those that distribute the workload between instances of a database. In a truly globalized environment, however, databases are replicated using tools such as GoldenGate, Streams, etc. Global services provide a mechanism to fail over the connections globally between the databases.

It is easier to explain the concept of GDS with a connection string example. Consider that there are two databases kept in sync using a bidirectional replication technology. The database servicing the California region is physically located near that region, and likewise the database servicing the New York region is physically located there. Thus, clients in the California region will benefit by connecting to the California database during normal operating conditions, and in the case of failure of the database in California, the connections should fail over to the New York database, and vice versa. Until version 11.2, this was achieved using a connect time failover connect string.

As of version 12c, the connect string for the California region specifies two address_list sections. These two address_lists have the FAILOVER attribute set to ON but load_balance set to OFF. This connection string will try the connection to the first address_list in the connect string; that is, the application will connect to the California database. If that database does not respond before TCP timeout, then the connection will fail over to the New York database.

```
(DESCRIPTION=
  (FAILOVER=on)
  (ADDRESS_LIST=
    (LOAD_BALANCE=ON)
    (ADDRESS=(host=ca1)(port=1522))
```

```
    (ADDRESS=(host=ca2)(port=1522))
    (ADDRESS=(host=ca3)(port=1522)))
  (ADDRESS_LIST=
    (LOAD_BALANCE=ON)
    (ADDRESS=(host=ny1)(port=1522))
    (ADDRESS=(host=ny2)(port=1522))
    (ADDRESS=(host=ny3)(port=1522)))
  (CONNECT_DATA=
    (SERVICE_NAME=PO)
    (REGION=ca))
 )
```

In addition to the failover specification among address_list sections, each address_list section also specifies multiple addresses with LOAD_BALANCE set to ON. Essentially, a new connection from the California region will try the first address_list as load_balance is set to OFF by default for description list. Then, an address from the chosen address_list will be used to send packets to GDS listeners as load_balance is set to ON for that address_list.

Note that traditionally, a VIP listener or a SCAN listener is specified in the ADDRESS section. But, in the case of GDS connect string, these are not traditional listeners, but **GDS listeners**. This feature was introduced in version 12c. GDS listeners can be visualized as global SCAN listeners. Oracle recommends three GDS listeners per region, very similar to the three SCAN listeners in a RAC cluster. Also, services are managed by srvctl command and global services are managed by the gsdctl command interface.

GDS listeners connect to the database and register as subscribers of the servicemetric queue. Any topology changes and Load Balancing Advisories (LBAs) are propagated to GDS listeners, and the GDS listeners use the LBAs to forward connections to a database that can provide better quality of service.

GDS are useful with the Active Data Guard setup also. With the Active Data Guard feature, it is possible to use a data guard instance for read-only activity. In many cases, the need arises for a global failover; that is, if the Active Data Guard database is not available, then failover will be to the primary database and vice versa. The GDS feature will be handy in implementing this requirement. Refer to the GDS Concept and Administration Guide for more details.

Failover in RAC

A RAC database offers many failover techniques. Failover methods can be classified into two categories: connection time and runtime failover methods. *Transparent Application Failover* (TAF) is a reactive runtime failover method, wherein database connection will automatically fail over to a surviving instance if the current instance fails. *Fast Connection Failover* (FCF) is a proactive runtime failover technique. Changes in node availability, service failures, etc., are propagated to the application, and the application will proactively recycle connections even before the application encounters connection-related errors.

TAF

TAF fails over the connection to a surviving instance after encountering an instance failure. TAF can operate in either SESSION or SELECT failover mode.

- In a SESSION failover mode, if a connection is severed then the connection will fail over to a surviving instance. Essentially, the connection is reattempted automatically. Only the connection is restarted, and the application must detect failures and restart failed workload.
- In a SELECT failover mode, the client-side driver remembers the cursor fetch details, and if the current connection is severed, then the connection will fail over to surviving node. The SELECT statement is re-executed, rows are fetched from the cursor and scrolled to the point of failure in the cursor fetch operation, and rows are returned from the point of failure.

The following connect string is a simple example of TAF using a SCAN listener. With this connection string, the application connects to a SCAN listener listening in port 12099. Notice that the failover_mode parameter specifies the type as SESSION, indicating session failover mode. If the connection is severed, then the client will retry the connection to the SCAN listener again. The SCAN listener will redirect the connection to the VIP listener of a surviving node.

```
PO_TAF =
 (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)
     (HOST = rac-scan.example.com)(PORT = 12099))
          (CONNECT_DATA =
            (SERVICE_NAME = PO)
            (FAILOVER_MODE= (TYPE=SESSION) (METHOD=basic))
     )
    )
```

Dynamic-view v$session can be queried to verify if TAF is configured properly or not. The preceding connect string shows that session level failover is enabled for the current connection. Killing the connection resulted in respawning of a new connection, confirming that TAF is working as designed.

```
SQL> select sid, FAILOVER_TYPE, FAILOVER_METHOD, failed_over
     from v$session where
     sid =(select sid from v$mystat where rownum =1);

       SID FAILOVER_TYPE FAILOVER_M FAI
---------- ------------- ---------- ---
      8227 SESSION       BASIC      YES
```

While it is possible to set up TAF with a client connection string, TAF can be implemented more elegantly using `service` attributes. The advantage of implementing TAF using service attributes is that configuration changes can be performed on the database server side without altering the client-side connect string. The following srvctl command shows that SESSION failover type is added to po service. Option –z specifies failover retry attempts, and the –w option specifies delay between failover attempts. In this example, connection will be attempted five times with 60-second delays between successive attempts.

```
$  srvctl add service -d orcl12 -s po -r oel6vm1 -a oel6vm2 \
             -m BASIC -e SESSION -z 5 -w 60
```

In version 12c, adding a service requires a different syntax, and options are specified more verbally. The earlier command is written in version 12c as follows.

```
$ srvctl add service -db orcl12 -service po \
      -preferred oel6vm1 -available oel6vm2 \
      -tafpolicy BASIC -failovermethod SESSION \
      -failoverretry 5 -failoverdelay 60
```

It is a better practice to keep the connection string as simple as possible and modify configuration of services to implement TAF. This strategy simplifies connection string management in an enterprise database setup.

Further, it is best to implement TAF while creating services so that you can improve availability.

Fast Connection Failover

Fast Connection Failover (FCF) is a proactive failover method to avoid downtime when there is an instance failure. Proactive connection management improves application availability and user experience, as you can reduce runtime errors. Here are a few examples where FCF can be used to avoid errors:

- After a database instance failure, all connections in a connection pool must be recycled by the connection manager. Usually, stale connections are identified only when the connection is reused by the application[4], but the failure is detected only after an error is encountered by the application. This late detection of error condition can lead to application errors and other undesirable user experiences.
- Connection pool connections that were busy in the database executing queries might need to wait for TCP timeouts. This delay can lead to slowness and errors in the application.
- After the recovery of failed instances, the application might not reconnect to the recovered node. This issue can lead to unnecessary overload of the surviving nodes.

Instance failures can be propagated earlier to the application, and errors can be minimized by requesting the application to recycle its connections after node or service failures. Essentially, service changes, node failures, node up/down events, etc., are communicated immediately to the application, and this propagation is known as FAN. After receiving a FAN event, the application proactively recycles connections or takes corrective actions depending upon the nature of the failure.

FCF is a technique that captures FAN events and reacts to the received FAN events. FAN events can be captured and connections recycled utilizing a few technologies:

1. Your application can implement connection pool technology that handles the FAN events, such as Oracle WebLogic Active GridLink data source, Oracle JDBC Universal Connection Pool, ODP.Net connection pool, OCI session pools, etc.
2. Your application can use programmatic interfaces to capture FAN events and react accordingly.
3. Or, you can use database side callouts to handle FAN events.

WebLogic Active GridLink

WebLogic Active GridLink for RAC is a data source available in Oracle WebLogic version 11g. This data source is integrated with RAC to handle FAN events and LBA without the help of any code changes. To make use of Active GridLink for FCF, you should configure an Active GridLink data source to connect to the SCAN listener and the ONS daemon, in a WebLogic configuration screen.

The ONS daemon propagates RAC topology changes to the registered clients. Any topology change is propagated to the clients listening to the SCAN IP address and 6200 ports. GridLink data source registers to the ONS daemon to receive notifications. If there are any topology changes as broadcast by the ONS daemon, such as instance restart, then GridLink data source captures the events and recycles the connections in the connection pool. Further, if there is an Instance UP event due to instance restart, then that event is captured by GridLink data source, and the connections are reapportioned among available nodes. This connection management avoids overloading of too few nodes after node recovery. Also, Active GridLink receives LBAs from ONS daemons, and GridLink data source uses connections from the least-loaded instances, implementing the runtime load balancing algorithm.

[4] Applications typically borrow connections from a connection pool, perform database activity, and return the connections to the connection pool. It is more likely that some connections may not have been reused for days, and so there may be application errors after few days.

Essentially, Active GridLink is an important tool to implement FCF, runtime load balancing, and transaction affinity. If your application uses WebLogic as a web server, you should implement Active GridLink-based data sources to maximize the failover capabilities of RAC.

Transaction Guard (12c)

A key problem with failure detection and failure recovery is that the response from the database to the application is transient. So, the response to a COMMIT statement can be lost in the case of instance failures. For example, consider a scenario in which an application modified some rows and sent a commit to the database, but the database instance failed and the application did not receive any response from the database. Hence, the application does not know about the transaction status as the response from the database was lost. Two unique scenarios are possible:

1. Database committed the changes, but success response was not received by the application.
2. Database failed even before the successful commit operation, and so the application must replay the changes.

In either scenario, the application must know the transaction status to avoid data corruption. A new feature in version 12c is the Transaction Guard, which provides a mechanism to keep track of the transaction state even after database failures.

The Transaction Guard feature introduces the concept of logical transaction identifier. A logical transaction_id is unique for a specific task from a connection, and the logical transaction_id is mapped to a database transaction_id. The outcome of the logical transaction_id is tracked in the database. If there is a failure in the database, then application connection can query the transaction status using logical transaction_id to identify if a transaction is committed or not.

To implement this feature, you need to create a service with two attributes, namely, `commit_outcome` and `retention`. The commit_outcome attribute determines whether the logical transaction outcome will be tracked or not. Attribute retention specifies the number of seconds a transaction outcome will be stored in the database (defaults to a value of 86,400 seconds = 1 day). The following command shows an example of po service creation with commit_outcome set to TRUE, enabling Transaction Guard.

```
$ srvctl add service -db orcl12 -service po -preferred oel6vm1 -available oel6vm2 \
  -commit_outcome TRUE –retention 86400
```

The Transaction Guard feature requires code changes, and your application code must be changed to react at the onset of failure. Application code must perform the following actions at the time of failure:

1. Receive FAN events from ONS daemon for failures.
2. Get the last logical transaction_id by calling getLTXID from a failed session.
3. Retrieve the status of the logical transaction_id by calling GET_LXID_OUTCOME, passing the logical transaction_id retrieved in step 2.
4. Handle the transaction depending upon the transaction state.

Application Continuity (12c)

Application Continuity is a new feature implemented using the Transaction Guard feature. Oracle introduces a new client-side JDBC driver class to implement Application Continuity. Universal Connection Pool and WebLogic Active GridLink support Application Continuity by default. If your application uses a third-party connection pool mechanism, then you need to modify code to handle Application Continuity.

The application issues database calls through the JDBC driver. The JDBC replay driver remembers each call in a request and submits the calls to the database. In the case of failures, JDBC replay driver uses the Transaction Guard feature to identify the global transaction state. If the transaction is not committed, then the JDBC replay driver replays the captured calls. After executing all database calls, replay driver issues a commit and terminates replay mode.

To support Application Continuity, service must be created by setting a few service attributes, namely, failovertype, failoverretry, failoverdelay, replay_init_time, etc. The following command shows an example of service creation. Notice that failovertype is set to TRANSACTION; this is an additional failovertype value introduced by version 12c.

```
$ srvctl add service -db orcl12 -service po -preferred oel6vm1 -available oel6vm2 \
   -commit_outcome TRUE –retention 86400 –failovertype TRANSACTION \
   -failoverretry 10 –failoverdelay 5 –replay_int_time 1200
```

Not all application workloads can be safely replayed. For example, if the application uses autonomous transactions, then replaying the calls can lead to duplicate execution of autonomous transactions. Applications must be carefully designed to support Application Continuity.

Policy-Managed Databases

Traditionally, Clusterware resources (databases, services, listeners, etc.) are managed by database administrators, and this type of management is known as administrator-managed databases. Version 11.2 introduced policy-managed databases. This feature is useful in a cluster with numerous nodes (12+) supporting many different databases and applications.

With a policy-managed database, you create server pools, assign servers to the server pool, and define policies of server pools. Depending upon the policies defined, servers are moved into and out of server pools. For example, you can define a policy such that the online server pool has higher priority during the daytime and the batch server pool has higher priority in the nighttime. Clusterware will manage the servers in the server pool such that more resources can be allocated, matching workload definitions.

By default, two server pools, that is, *free* and *generic* server pools, are created. Server pool free is a placeholder for all new servers, and as the new server pools are created, servers are automatically reassigned from free server pool to new server pools. Generic server pool hosts pre-11.2 databases and administrator-managed databases.

You can associate applications to server pools, and a database is also an application from the Clusterware perspective. An application can be defined as singleton or uniform state. If an application is defined as singleton, then that application can exist only in a server, and if defined as uniform, then that application will exist on all servers in a server pool.

Temporary Tablespaces

Temporary tablespaces in RAC require special attention, as they are shared between the instances. Temporary tablespaces are divided into extents and instances cache subset of extents in their SGA. When a process tries to allocate space in the temporary tablespace, it will allocate space from cached extents of the current instance.

Dynamic performance view `gv$temp_extent_pool` shows how temporary tablespace extents are cached. Instances try to cache extents equally from all files of a temporary tablespace. For example, approximately 4,000 extents from files 6, 7, 8, and 9[5] were cached by every instance. So, you should create temporary tablespaces with as many temp files as there are instances. In a nutshell, extents are cached from all temporary files, thereby spreading the workload among the temporary files of a temporary tablespace.

[5] Only partial output is shown. Files 1 through 5 also exhibit similar caching behavior, but extents in use were 0 for those files when the view was queried. Thus, the output of those files is not shown.

```
RS@SQL> select inst_id, file_id, extents_cached , extents_used
        from gv$temp_extent_pool
  order by inst_id, file_id;

   INST_ID    FILE_ID EXTENTS_CACHED EXTENTS_USED
---------- ---------- -------------- ------------
...
         1          6           4026            0
         1          7           4021            2
         1          8           3984            4
         1          9           4033            2
...
         3          6           3984            0
         3          7           3988            1
         3          8           3985            0
         3          9           3967            1
```

The size of the cached extent is governed by the attributes of the temporary tablespace. For example, in RAC1 database, the extent size is 128KB.

```
RS@SQL>select tablespace_name, INITIAL_EXTENT, NEXT_EXTENT
      from dba_tablespaces
where tablespace_name='TEMP';

TABLESPACE_NAME                INITIAL_EXTENT NEXT_EXTENT
------------------------------ -------------- -----------
TEMP                                   131072      131072
```

Extent caching is a soft reservation technique. Consider that a RAC1 instance used all of its cached extents (because of a session connected to RAC1 performing a massive sort or join operation); then, that instance will ask other instances to release the soft reservation for a group of extents in the temporary tablespace. After the extents are uncached from other instances, the RAC1 instance can cache the uncached extents.

Extent caching and uncaching operations are performed under the protection of SS Enqueue. Further, the DFS lock handle mechanism (discussed in Chapter 11) with CI Enqueue is used to trigger uncaching of extents in other instances. So, excessive caching/uncaching can lead to waits for SS Enqueue and waits for DFS lock handle. Since version 11g, uncaching of extents is performed in batches of 100 extents each per operation, and so SS Enqueue contention is greatly reduced.

The following guidelines can be used while creating temporary tablespaces:

- Create as many temp files as the number of instances in every temp tablespace. For example, in an eight-instance database, there should be at least eight temp files associated with every temporary tablespace. This strategy reduces the file header level locking issues dramatically.
- While adding an instance to a database, increase the size of temporary tablespace. Also, increase the number of temp files in temporary tablespaces.
- If an application component uses an excessive amount of temporary tablespace, then create a new user for that application component, create additional temporary tablespace, and use application affinity to keep that component to a node. This strategy reduces the ill effects associated with excessive extent caching and uncaching activities.

Massive Data Changes

It is inevitable that every DBA must perform massive data changes in a production environment, for reasons such as upgrade, purge, etc. In a RAC database, performing massive changes requires careful strategy.

For massive updates, if you are planning to use parallelism, consider setting the parallel_force_local parameter to true. This initialization parameter will allocate all PX servers in local node and reduce overheads due to global cache latency. However, this strategy assumes that one server has enough resource capabilities to complete the tasks. On the other hand, if the activity is spread across many nodes, then CPU, I/O, and memory resource usage is spread between all nodes. So, the choice to use one or more instances depends upon (a) whether a node can handle the workload without incurring any additional latency, and (b) whether the global cache latency is big enough that keeping workload in a node will vastly improve the performance.

If you choose to use multiple instances, then the use of table partitioning or data partitioning through effective SQL tuning to reduce global cache transfers is another effective strategy to improve application performance. Further, index blocks deserve close attention, as these can increase global cache transfers. For massive updates on indexed columns or inserts, it is preferable to drop indexes before updating columns, and recreate the indexes after the updates.

Another typical production issue is encountered during massive index creation. Index creation is usually performed with many PX servers. As PX servers read blocks into PGA directly, bypassing the buffer cache, global cache latency is minimal. Still, PX servers can distribute blocks among themselves, flooding interconnect with PX messages. So, allocating all PX servers within the same node is preferable if a single node has sufficient resources to complete index creation. Refer to Chapter 12 for further discussion of parallel query execution.

Further, if an index creation fails due to an issue, always restart that index creation connecting to the same instance. Temporary space allocated for indexes (in permanent tablespace) is soft reserved. If you restart index creation connecting to another instance, then subsequent index creation would trigger uncaching of free space in other nodes, leading to a possibility of SS Enqueue contention and DFS lock handle contention.

Performance Metrics Collection

It is critical to collect performance metrics on a regular basis in RAC cluster. In most cases, RAC node problems are side effects of some other OS issue. For example, network, CPU starvation, and memory starvation issues can lead to node failures and reboots. Root cause analysis of a node reboot requires deep review of OS metrics. It is almost impossible to identify the root cause of a node reboot without sufficient OS performance metrics.

At the minimum, OSWatcher or another tool should collect performance data in all nodes and keep those statistics for at least a few days. These OS statistics will enable root cause analysis for node failures.

Further, AWR reports or statspack data are essential to understand pattern changes in database performance. So, at least a few weeks of history must be kept in the database for future analysis.

Parameter File Management

The initialization parameter file is common to all instances. It is a good idea to keep parameter files in ASM storage, as ASM storage is available to all RAC database instances.

Initialization parameters can be specified in the scope of an instance or database. For example, the following command modifies the parameter `pga_aggregate_target` in a RAC1 instance only.

```
alter system set pga_aggregate_target=10240M scope=both sid='RAC1';
```

The following command modifies the parameter globally. If a parameter is specified both at the database level and at the instance level, then the instance-level parameter specification overrides the global-level parameter.

```
alter system set pga_aggregate_target=5120M scope=both sid='*';
```

SPFILE will contain the following entries for the pga_aggregate_target parameter after executing the preceding two commands.

```
*.pga_aggregate_target=5120M
RAC1.pga_aggregate_target=10240M
```

The command to reset or remove a parameter from spfile should match with the scope of the parameter if the parameter exists in spfile already. For example, to remove *.pga_aggregate_target from spfile completely, you must specify the sid clause matching the spfile scope.

```
alter system reset pga_aggregate_target scope=both sid='*';
```

Not all parameters can be modified at the instance level. Column ISINSTANCE_MODIFIABLE in v$parameter shows whether a parameter can be modified at the instance level or not. The following output of SQL querying v$parameter shows that pga_aggregate_target parameter can be modified at the instance level.

```
SQL> select name, ISINSTANCE_MODIFIABLE from v$parameter
            where name='pga_aggregate_target'
NAME                           ISINS
------------------------------ -----
pga_aggregate_target           TRUE
```

Password File Management

Password files allow non-sys users to log in to the database with elevated privileges such as sysdba, sysoper, sysbackup (12c), sysdg (12c), etc. In 11gR2, each database instance is considered to have a non-shared password file, and so grants in an instance are not propagated to other database instances. You must copy password files to each node and rerun grants connecting to local instance in version 11.2.

In version 11.2 and earlier, it is a better approach to store the password files in a shared file system such as NFS or cluster file system. With this shared password file approach, you can maintain the password files with ease, but grants still need to be executed in every node.

```
$ orapwd file='orapworcl12' entries=10
```

In version 12c, password files can be stored in ASM, and this greatly simplifies password file management. By default, password files are stored in the ASM disk group.

```
+DATA/ORCL12/PASSWORD/pwdorcl12.268.804626815
```

The following command shows how a password file can be created in Oracle ASM disk group. As the password file is stored in ASM, all instances can access the password file.

```
$ orapwd file='+DATA/orcl12/orapworcl12' entries=8 dbuniquename='orcl12' format=12
```

Prior to version 11.2, you must grant privileges in every instance explicitly, as the sysdba grants operate at the instance level. This behavior is due to an inherent assumption that the password file is not shared between the instances. For example, in the following output, granting sysdba to rs user connecting to RAC2 instance populated the password file only for instance 2.

```
SQL> grant sysdba to rs;

Grant succeeded.
```

```
SQL>select * from gv$pwfile_users where username='RS';

   INST_ID USERNAME                       SYSDB SYSOP SYSAS
---------- ------------------------------ ----- ----- -----
         2 RS                             TRUE  FALSE FALSE
```

In version 12c, sysdba grants are considered to operate at the database level, and so sysdba grants are populated for all instances immediately. This behavior change in version 12c is logical, since the password files are stored in ASM and shared between instances. The following output shows that grants to a user are propagated to all instances, as is visible in the password file.

```
SQL> grant sysdba to rs;
Grant succeeded.

SQL> select * from gv$pwfile_users where username='RS';
INST_ID USER SYSDB SYSOP SYSAS SYSBA SYSDG SYSKM CON_ID
------- ---- ----- ----- ----- ----- ----- -----     --
1         RS       TRUE  FALSE FALSE FALSE FALSE FALSE  0
2         RS       TRUE  FALSE FALSE FALSE FALSE FALSE  0
```

The Clusterware resource for the database also has a new attribute referring to the password file; in this example, the password file is stored in ASM.

```
$ crsctl stat res ora.orcl12.db -p |more
...
PWFILE=+DATA/orcl12/orapworcl12
...
```

Version 12c introduces new administrative privileges, namely, sysdg, sysbackup, and syskm, in addition to sysdba and sysoper privileges available in earlier versions. These new privileges can be secured with passwords too, but the password file should be created or migrated to version 12c format using the `format=12` option in the orapwd command while creating password files.

It is important to maintain the password files consistently in all instances. Storing password files in a shared file system is a good practice in earlier versions, and storing the password files in ASM is a good practice in version 12c.

Managing Databases and Instances

To manage a database, it is advisable to use the srvctl command. While it is possible to manage databases as Clusterware objects using crsctl commands, you should try to use the srvctl command whenever possible. The srvctl command provides an additional layer of security and avoids unnecessary activity in the Clusterware owner account.

The general syntax of the srvctl command is given in the following. To start or stop a database, you would use the following command:

```
$ srvctl stop database -d ORCL
srvctl <operation> <object> [<options>]
```

By default, the database will be stopped in immediate shutdown mode. You could also specify other shutdown modes using the –o option as follows. Other options, such as transactional, immediate, etc., can be specified too.

```
$ srvctl stop database -d ORCL -o abort
```

The database can also be started using the srvctl command. The following commands start the database in OPEN startup mode. Startup mode can be explicitly specified too.

```
$ srvctl start database -d ORCL
$ srvctl start database -d ORCL -o open
```

The parameter file can be explicitly specified using the pfile option.

```
$ srvctl start database -d ORCL -o open,pfile=initORCL1.ora
```

Instances can be managed individually, too. For example, an ORCL1 instance alone can be stopped using the following command. It is also possible to specify a node name, as the mapping between instance_name and node_number is already stored in OCR. You could also stop multiple instances in a single command specifying a list of instances.

```
$ srvctl stop instance -d ORCL –i ORCL1
$ srvctl stop instance -d ORCL –n RAC1
$ srvctl stop instance -d ORCL –i ORCL1,ORCL2
```

Configuration of a database can be queried using the `config database command` and shows the spfile location, oracle_home location, default start options, default stop options, and instances of a database, etc.

```
$ srvctl config database -d ORCL
Database unique name: ORCL
Database name: ORCL
Oracle home: /u01/app/oracle/product/11.2.0/dbhome_1
Oracle user: oracle
Spfile: +DATA/orcl/spfileorcl1.ora
Domain: rac.example.com
Start options: open
Stop options: immediate
Database role: PRIMARY
Management policy: AUTOMATIC
Server pools: ORCL
Database instances: ORCL1,ORCL2
Disk Groups: DATA
Services:
Database is administrator managed
```

Mapping between the node and instance names is stored in OCR, and array variable USR_ORA_INST_NAME maintains this mapping. For example, in the following configuration, instance ORCL1 will be started in RAC1 node and ORCL2 instance will be started in RAC2 node. It is possible to modify the USR_ORA_INST_NAME array variable and change the mapping if needed.

```
$ crsctl stat res ora.racdb1.db -p|grep ORA_INST_NAME
...
GEN_USR_ORA_INST_NAME@SERVERNAME(RAC1)=ORCL1
GEN_USR_ORA_INST_NAME@SERVERNAME(RAC2)=ORCL2
...
USR_ORA_INST_NAME@SERVERNAME(RAC1)=ORCL1
USR_ORA_INST_NAME@SERVERNAME(RAC2)=ORCL2
```

However, ASM instance mapping cannot be changed that easily, since the ASM instance name is dynamically derived from the node number. Updating the node number would require Clusterware reconfiguration, as the node number to node name mapping is stored in the voting disk.

```
$ crsctl stat res ora.asm -p |grep INST_NAME
..
GEN_USR_ORA_INST_NAME@SERVERNAME(RAC1)=+ASM1
GEN_USR_ORA_INST_NAME@SERVERNAME(RAC2)=+ASM2
USR_ORA_INST_NAME=+ASM%CRS_CSS_NODENUMBER%
```

In a cluster with a large number of nodes, it is typical to keep the node name with an incrementing suffix, such as RAC1, RAC2, RAC3, etc. It is also a matter of convenience to keep suffixes of both database and ASM instances matching with node name suffixes; for example, ORCL1 and ASM1 instance in RAC1 node, ORCL2 and ASM2 instance in RAC2 node, etc. As the node number is assigned at the time of cluster configuration, execute root.sh in the order desired; for example, execute root.sh in RAC1 node, wait for completion, execute root.sh in RAC2 node, wait for completion, etc. This execution order will maintain correct suffix mapping.

Managing VIPs, Listeners

To manage VIPs and listeners, use of the srvctl command is recommended. Configuration of VIP can be queried using the `config vip` parameter.

```
$ srvctl config vip  -n RAC1
VIP exists.:RAC1
VIP exists.: /RAC1_vip/1.2.1.101/255.255.255.0/eth1

$ srvctl config vip  -n RAC2
VIP exists.:RAC2
VIP exists.: /RAC2_vip/1.2.1.102/255.255.255.0/eth1
```

VIPs can be started and stopped using the srvctl command, but Clusterware daemons usually monitor these resources, and so these commands are seldom used.

```
$ srvctl stop vip  -n RAC2
$ srvctl start vip  -n RAC2
```

Similarly, listeners can be managed using the srvctl command. For example, configuration of a listener can be queried using the following config command.

```
$ srvctl config listener  -l LISTENER
Name: LISTENER
Network: 1, Owner: oracle
Home: <CRS home>
End points: TCP:1521
```

Listeners can be stopped and started using the srvctl command with the following syntax. Executing the lsnrctl command is not a recommended practice.

```
$ srvctl stop listener  -l LISTENER
$ srvctl start listener  -l LISTENER
```

The srvctl command is an option-rich tool, and if you are not sure about the syntax, type `srvctl -h` to understand the options in the srvctl tool.

Miscellaneous Topics

There are some important topics that require special attention in RAC. For example, in a single-instance database, memory starvation can lead to performance issues, but in RAC, memory starvation can lead to a node reboot.

Process Priority

In RAC, a few background processes must have higher CPU scheduling priority. For example, an LMS background process should run with higher CPU priority, as the LMS process is the workhorse of global cache transfer, and CPU starvation of an LMS process can increase global cache latency. Further, if an LMS process suffers from CPU starvation, network packets can be dropped, leading to gc blocks lost waits in other nodes. Underscore initialization parameter _high_priority_processes controls the processes that will execute at an elevated priority. By default, this parameter is set to elevate the priority of VKTM (Virtual Keeper of Time) and LMS background processes. Generally, you do not need to modify this parameter.

```
_high_priority_processes= 'LMS*|VKTM'
```

For example, in the Solaris platform, the LMS process will execute with RT priority. Of course, elevating a process priority requires higher-level permissions, and this is achieved using oradism binary in Unix platforms and oradism service in Windows platform. Notice in the following that oradism executable is owned by root userid with setuid flag enabled. OS group `prod` has execute permission on this oradism binary. Essentially, with setuid permission of the root user, the oradism executable is utilized to elevate the priority of background processes.

```
$ ls -lt oradism
-rwsr-x---   1 root     prod     1320256 Sep 11  2010 oradism
```

Setuid permissions on oradism executable are needed to alter the priority of background processes. When you execute root.sh during GI software installation, this setuid permission is set on oradism executable, but during software cloning, DBAs typically do not run root.sh, and so the oradism executable does not have correct permissions. Hence, the priority of LMS process is not elevated, leading to global cache latency issues during CPU starvation. Executing root.sh script and verifying that the oradism executable has correct permission after the completion of software cloning is recommended.

Also, GRID processes, such as cssd.bin, cssdagent, and cssdmonitor, etc., should run with elevated priority. Again, oradism binary in Grid Infrastructure software must have proper permissions. Permissions on an ordism file in GI were not a big problem until version 11.2, as cloning of GI was not widely used. However, in version 12c, cloning of GI software is allowed and binary permissions require careful attention if GI software is cloned.

While elevated priority for background processes protects the background processes from CPU starvation, it does not mean that CPU starvation will not cause node stability issues. For example, the LMS process must wait for the LGWR background process to do a log flush sync before sending the blocks (for certain types of global cache transfer). If there is CPU starvation in the server, then LGWR might not get enough CPU; LGWR will suffer from CPU starvation, and that can lead to longer waits by the LMS process. Remote instances will suffer from prolonged global cache transfers. So, CPU starvation in one node can increase the global cache latency in other nodes, inducing slowness in the application; in a few scenarios, this CPU starvation can also lead to node reboots. It is a recommended practice to keep CPU usage below 75% to avoid CPU starvation and reduce node stability issues.

Memory Starvation

Memory starvation in a cluster node can cause stability issues in a RAC cluster. Elevated process priority protects the background process from CPU starvation but does not prevent memory starvation issues. Memory starvation of an LMS process can lead to higher latency for global cache wait events in other nodes. Further, memory starvation of Grid Infrastructure daemons can lead to node reboots. It is a recommended practice to design cluster nodes with sufficient memory and avoid memory starvation.

Virtual memory swappiness is an important attribute to consider in a Linux kernel. This kernel parameter affects swapping behavior. As Linux kernel version 2.6.18 improves swapping daemons throughput, you should upgrade the kernel to at least version 2.6.18. If you cannot upgrade the kernel to at least 2.6.18, then the vm.swappiness kernel parameter is an important parameter to consider. If the parameter is set to a lower value, then pages will be kept in memory as long as possible. If the vm.swappiness parameter is set to a higher value, then the pages will be swapped more aggressively. If you are using a version prior to 2.6.18 kernels, you should reduce the vm.swappiness kernel parameter to a lower value such as 30 (default is 60).

We also recommend that you implement HugePages in the Linux platform. At the time of this writing, Oracle has made HugePages as one of the important ways to improve the stability of a RAC cluster. With HugePages, two important benefits can be achieved:

- SGA is locked in memory. Linux kernel daemons have fewer pages to scan during memory starvation. Locking SGA also protects from paging.
- Further, default page size of memory mapping is 4KB in the Linux platform. Pointers to these pages must be maintained in page table entries, and so page table entries can easily grow to a few gigabytes of memory. With the HugePages feature, page size is increased to 2MB and the page table size is reduced to a smaller size.

HugePages and Automatic memory management (AMM) do not work together. We recommend you use HugePages instead of AMM to improve RAC stability.

SGA size

If the data blocks are already residing in local SGA, then access to those buffers is in the range of microseconds. Thus, it is important to keep SGA size bigger in a database with OLTP workload. It is a best practice to increase the SGA size when you convert the database to RAC.

If the application workload is mostly Decision Support System (DSS) or Data Warehouse type, then keep PGA bigger and buffer cache smaller, as PX server processes read blocks directly into the PGA (Program Global Area). The parameter pga_aggregate_target is *not* a hard limit and the sum of PGA allocated by the database processes can far exceed the pga_aggregate_target parameter. This unintended excessive memory usage can induce memory starvation in the server and lead to performance issues. Release 12c introduces pga_aggregate_limit parameter and the total PGA used by the database cannot exceed the pga_aggregate_limit value. If the database have allocated pga_aggregate_limit amount of PGA, then the process with largest PGA allocation is stopped until the PGA size falls below the pga_aggregate_limit value. Therefore, in 12c, you can avoid excessive memory swapping issues by setting pga_aggregate_limit to an optimal value.

It is advisable to keep the SGA size as big as possible without inducing memory starvation, as discussed earlier. Database nodes with ample physical memory are essential while converting a database to a RAC database. More memory can also reduce unnecessary CPU usage, which can be beneficial from a licensing point of view.

Filesystem Caching

For Oracle database software and Grid Infrastructure software, the file system must have caching enabled. Without file system caching, every access to binary file pages (software binary pages are accessed frequently from an executing UNIX process) will lead to a disk access. This excessive disk reads induces latency in the executable, leading to instance reboot or even node reboot.

Summary

In summary, the following guidelines are offered for the operational aspects of a RAC cluster.

- Use services effectively to partition the application workload. Load balancing and failover can be achieved elegantly with the use of services.
- Use SCAN listeners and SCAN IP addresses. Publish only SCAN name to the application. Do not allow an application to connect to VIP listeners directly.
- Use three SCAN IP addresses mapping to a SCAN name.
- Set up TAF attributes of a service to implement TAF.
- Use tools such as universal connection pool or Active GridLink data source to implement faster failover and load balancing capabilities.
- Use as many temp files as the number of instances for temporary tablespaces. Allocate more space if you are planning to add more instances to the database.
- Manage application connection and users so as to reduce locking due to soft reservation in temporary tablespaces.
- Verify that oradism binary has correct permissions so that background processes can execute at an elevated priority.
- In Linux, implement HugePages to avoid unnecessary reboots.

CHAPTER 4

New Features in RAC 12c

by Syed Jaffar Hussain, Kai Yu, and Tariq Farooq

The goal of this chapter is to explain some of the key new features and enhancements in Oracle 12cR1 pertaining to Clusterware and RAC Database technologies. Cloud computing is a key focus of the Oracle 12c database, and "cloud" is what the letter "c" in Oracle 12c stands for. In the first part of this chapter, we will explain two core new features of Oracle Clusterware: ***Oracle Flex Cluster*** and ***Oracle Flex Automatic Storage Management (ASM)***. One reason why we put these features in this chapter is that they are very closely related. It would be very difficult to discuss one without mentioning the other; in particular, it is not possible to have Flex Clusters without Flex ASM. The use cases of these two new features are very similar and it is likely that they will be implemented together in real database environments.

The new Flex Clusters and Flex ASM features are designed to provide scalable and high-availability cluster infrastructure for database cloud and application cloud. The chapter will discuss the architecture of Oracle 12c Flex Clusters and Flex ASM, and how this Flex architecture helps to improve the scalability and availability of the cluster infrastructure. We will also discuss the configuration and management of the Flex Clusters and Flex ASM, and how to configure and run Oracle databases on the new Flex Cluster environment.

Pluggable databases (PDBs) are one of the key new features introduced in Oracle Database 12c. They allow multi-tenancy in a database by having multiple PDBs within a big container database (CDB). And these PDBs can be easily plugged into or unplugged from a CDB. This solution simplifies consolidation of multiple-application databases in a single CDB database. This also allows moving the application database from one CDB to another CDB. After a brief discussion of the PDB architecture, this chapter will focus on the RAC-based PDB and CDB. It will discuss how to create CDB and PDBs, how to connect to PDBs, and how to manage them with their associated database services.

The latter part of this chapter will give you a brief overview of other miscellaneous new features of 12cR1, such as:

- IPv6 Support for Public Networks
- Global Data Services
- Oracle ASM Cluster File System (ACFS) and Oracle ADVM: Enhancements
- Online Resource Attribute Modification
- Policy-Based Management and Administration
- ASM Disk Group: Shared ASM Password File
- Valid Node Checking: Restricting Service Registration
- Shared Grid Naming Service (GNS)
- Restricting Service Registration
- NFS High Availability

- Cluster Health Monitor (CHM) Enhancements
- Windows: Support for Oracle Home User
- Introducing Application Continuity
- Transaction Idempotence and Java Transaction Guard

Oracle Flex Clusters

Oracle Flex Cluster Architecture

As stated in Chapter 1, previous releases of Oracle Clusterware such as 11gR2 and earlier supported only the tightly connected cluster architecture:

1. Each node in the cluster is connected to other nodes through the private interconnect.
2. Each node in the cluster is directly connected to the shared storage.

This architecture will present significant technical challenges and performance overhead to cluster scalability if the cluster needs to be scaled out to many more nodes that most clusters today.

One issue is a dramatic increase of the interconnect traffic between cluster nodes. In a tightly connected cluster which we can call the standard cluster, since the interconnect connects each pair of nodes and every node is connected to the shared storage, an N-node cluster will have N *(N-1)/2 possible interconnect paths for cluster heartbeats and data exchanges between two nodes and N connection paths to the shared storage.

In a small or modestly sized cluster such as a 16-node cluster, 120 different interconnect paths and 16 storage connections may still be manageable. However, if we want to scale a cluster to a much bigger scope, for example 500 nodes, it will have 124,750 interconnect paths and 500 storage connections. Not only the complexity of the number of interconnect paths and storage connections makes the cluster very difficult to manage, the network traffic overhead due to this extremely high number of interconnect paths will practically impact the cluster performance hugely. It is obvious that this tightly connected cluster architecture prevents the cluster from being scaled further.

Oracle 12c Flex Clusters are designed to tackle this limitation by introducing a new two-layered hub-and–spoke topology to the cluster architecture. This new cluster consists of two types of nodes: Hub nodes and Leaf nodes. The group of Hub nodes is tightly connected as the nodes in the standard cluster in Oracle RAC 11gR2: all the Hub nodes are connected each other through the private interconnects and they also are directly connected to the shared storage through the physical storage network connections.

On the other hand, Leaf nodes run a lighter-weight stack and they are not connected to each other like Hub nodes. Leaf nodes also do not require direct access to the storage. There is no direct communication between Leaf nodes. Instead, each Leaf node communicates with its attached Hub node exclusively and is connected to the cluster through its Hub node. Leaf nodes get data through the attached Hub node. The cluster heartbeats for Leaf nodes only occur between Leaf nodes and their attached Hub nodes.

Now let's use a 16-node cluster as an example to show the two-layered hub-and–spoke topology of an Oracle Flex Cluster in Figure 4-1.

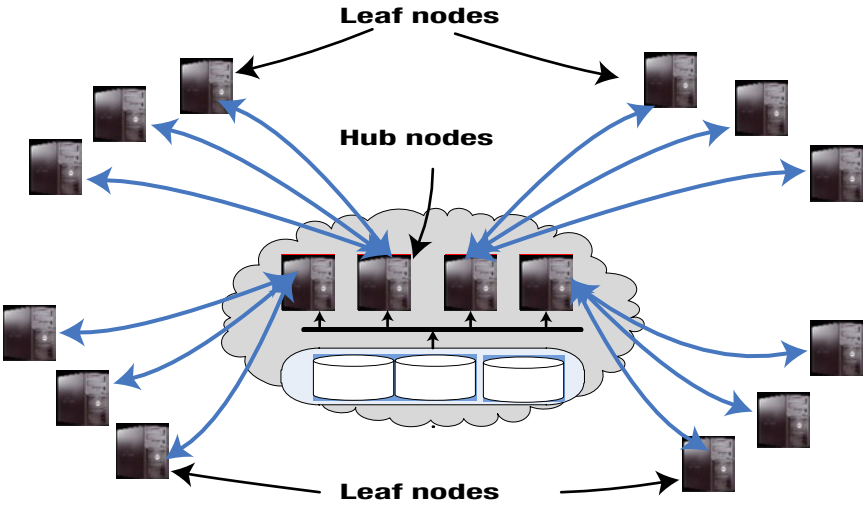


Figure 4-1. *Hub-and–spoke topology of Oracle Flex Clusters*

In this example, four Hub nodes in the Hub center form the hub of the cluster, which is similar to a standard cluster supported in Oracle 11gR2 Clusterware. These four Hub nodes are tightly connected with the private interconnect, which is the same as the one in the standard Oracle 11g R2 cluster. The cluster heartbeat and the data block transfer for RAC cache fusion among the Hub nodes are based on this interconnect network. All of these Hub nodes are also required to have direct access to the shared storage, which stores the database files as well as the Oracle Cluster Registry (OCR) and the voting disks for the Oracle Clusterware. The database instances running on their Hub nodes will function in the same way as in a standard cluster.

Outside of the Hub center, 12 Leaf nodes form four groups. Each group of Leaf nodes is connected to one Hub node, which is the attached node for all the Leaf nodes in the group. None of the Leaf nodes are connected to any other Leaf nodes in the cluster. Oracle RAC database instance can also run on Leaf nodes. Since these Leaf nodes do not have direct access to the shared storage, the RAC database instances on the Leaf nodes will need to get the database through the Hub nodes. The technology to implement this remote storage access is another new Oracle 12c feature known as Oracle Flex ASM. From here, we can see that Flex Clusters require Flex ASM to enable storage access for Oracle database instances on Leaf nodes. For this reason, when you configure a Flex Cluster, Oracle Flex ASM is automatically enabled.

One benefit of running the loosely coupled architecture of the Flex Cluster is to provide the high availability of the Oracle Clusterware of the applications tier. We can run application tiers on the Flex Cluster, which provides high availability, such as failover capability, against server hardware failover and planned maintenance.

Scalability and Availability of Flex Clusters

Oracle Flex Clusters increase cluster scalability. The hub-and–spoke topology in the Oracle Flex Clusters significantly reduces the number of network connections among the cluster nodes. In the 16-node cluster example shown in Figure 4-1, where we set up 4 Hub nodes and 12 Leaf nodes with 3 Leaf nodes per Hub node, the number of private interconnects among 4 Hub nodes is 6 and the total number of connections between the Leaf nodes and their Hub nodes is 12. This makes the total number of interconnects 18, compared to 120 in a 16-node standard cluster. The number of storage network connections is also reduced from 16 to 4. If we extend this cluster to a 500-node cluster on which we set up 25 Hub nodes and 475 Leaf nodes with 19 Leaf nodes per Hub node, the interconnect network connections consists of 300 interconnects between Hub nodes plus 475 connections between Leaf nodes and their attached Hub nodes. The total number of the connections is 300+475 = 775, while the total number of interconnects

for a 500-node standard cluster is 124,750, as we discussed in the beginning this chapter. The number of storage network connections is also reduced from 500 to 25.

This significant reduction of network connections allows us to further scale out the cluster. With this hub-and–spoke topology, the Flex Cluster in Oracle 12cR1 is designed to scale up 64 Hub nodes and many more Leaf nodes.

The Flex Cluster architecture helps to maintain the availability and reliability of the cluster even when the cluster is scaled out to a very large number of nodes. This is achieved by having the OCR and voting disk accessible only to Hub nodes and not Leaf nodes. For example, we will get the following error messages if we query the voting disks or OCR access from a Leaf node:

```
$ crsctl query css votedisk
CRS-1668: operation is not allowed on a Leaf node

$ ocrcheck
PROT-605: The 'ocrcheck' command is not supported from a Leaf node.
```

As shown in the previous 500-node example, in Flex Clusters there are only a small number of Hub nodes and the majority of the cluster nodes are Leaf nodes. By allowing only the Hub nodes to access the OCR and voting disks, scaling out a Flex Cluster will not significantly increase resource contention for OCR and voting disks. As a result, the chance of node eviction caused by contention for OCR and voting disk will not increase as the cluster is scaled out.

Like a standard cluster, an Oracle Flex Cluster is built with high-availability design. If a Hub node fails, this node will be evicted from the cluster in the same way as a node in a standard cluster. The services on the failed node will be failed over to other surviving Hub node in the cluster. The Leaf nodes that were connected to the failed Hub node can be reconnected to another surviving Hub node within a grace period. The private interconnect heartbeat between two Hub nodes are the same as the private interconnect heartbeat in the standard cluster. You can check the heartbeat misscount setting between Hub nodes using the following crsctl command:

```
$crsctl get css misscount
CRS-4678: Successful get misscount 30 for Cluster Synchronization Services.
```

If a Leaf node fails, this node will be evicted from the cluster. The services running on the failed Leaf node are failed over to other Leaf nodes that are connected the same Hub node. This failover mechanism keeps the failover within the group of Leaf nodes that are connected to the same Hub node. In this way, the other part of the cluster nodes will be not impacted by this Leaf node's failure.

The network heartbeat is used to maintain network connectivity between a Leaf node and the Hub node to which the Leaf node connects. Similar to the private interconnect heartbeat between the Hub nodes, the maximal threshold time that this heartbeat is tolerable is defined by the ***leafmisscount*** setting, which by default is 30 seconds. If the heartbeat failure passes this ***leafmisscount*** setting, then the Leaf node either will be reconnected to the other Hub node or will be evicted from the cluster. You can query this setting by running this command:

```
$ crsctl get css leafmisscount
CRS-4678: Successful get leafmisscount 30 for Cluster Synchronization Services
```

You can also manually reset this setting by running this command on a Hub node.

```
$ crsctl set css leafmisscount 40
CRS-4684: Successful set of parameter leafmisscount to 40 for Cluster Synchronization Services.
```

You cannot reset this setting from a Leaf node:

```
$ crsctl set css leafmisscount 40
CRS-1668: operation is not allowed on a Leaf node
```

Configuring Flex Clusters

With the new Oracle Flex Clusters feature, Oracle Clusterware 12cR1 has a new cluster-mode setting that allows us to enable the Flex Clusters functionality. By default, this setting is on standard cluster mode on which the Flex Clusters functionality is disabled. If you don't need to use the Flex Clusters functionality, you can just keep this mode setting as default. Then, your cluster will be on the standard cluster mode, which is similar to Oracle Clusterware 11gR2. Otherwise, users must explicitly enable the Flex Cluster in one of two ways:

Enable the Flex Clusters option during the new cluster configuration; or

Change the existing the cluster mode from Standard Cluster to Flex Cluster.

In either case, some preparation work should be done in advance. The first task is to determine how many nodes and which nodes will be the Hub nodes or the Leaf nodes. The second task is to configure the GNS as the prerequisite of Flex Cluster implementation. The Flex Cluster configuration requires a fixed GNS virtual IP (VIP) on one of the Hub nodes. In Oracle Clusterware 12cR1, there are two configuration options to meet this requirement: standard and static. With the standard option, a static GNS VIP and a subdomain delegation are configured in DNS so that the Single Client Access Name (SCAN) VIPs and all other cluster names and VIP addresses of the subdomain within the cluster are forwarded to the GNS service. With the static option, no subdomain is configured. Instead, the GNS VIP along with all the cluster names, VIP names, and their addresses are static and registered in DNS. All these names are resolved with DNS. Although both of these GNS configuration options work for Flex Clusters, you should at least implement the static option, as the standard option with subdomain delegation is not required in Flex Clusters. Chapter 9 will discuss the details of the GNS configuration.

Configuring a Flex Cluster with OUI

If you plan to enable the Flex Clusters feature in the new cluster configuration, you can enable it during Grid Infrastructure (GI) installation of Oracle Universal Installer (OUI) by selecting "Configure a Flex Cluster" option as shown in Figure 4-2.

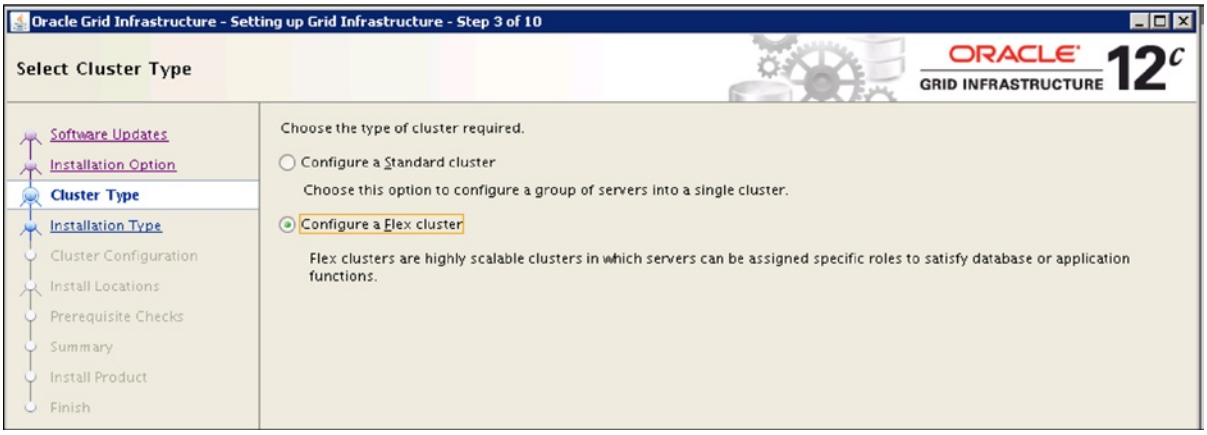


Figure 4-2. *Select Flex Cluster in Grid Infrastructure installation*

After you select this Flex Cluster configuration option, as the prerequisite for Flex Cluster installation, you need to specify the GNS configuration in the installation as shown in Figure 4-3: selecting the "Configure GNS" option and specifying the new fixed GNS VIP address, which will be used in the rest the installation steps.

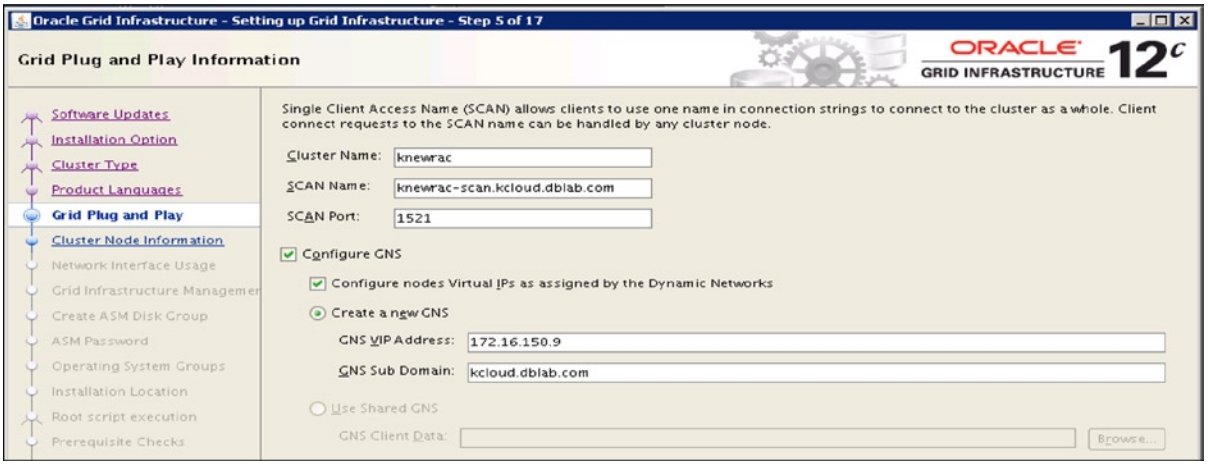


Figure 4-3. *Specify GNS in Grid Infrastructure installation*

As a part of the configuration, you need to specify the Hub nodes and Leaf nodes. For example, Figure 4-4 shows a seven-node Flex Cluster with three Hub nodes and four Leaf nodes.

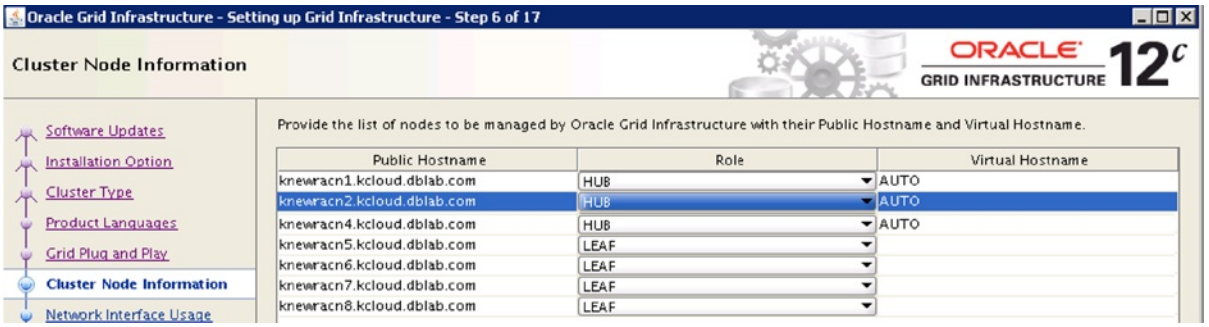


Figure 4-4. *Specify the role of the Cluster nodes*

As a part of Flex Cluster configuration, Oracle Flex ASM is implicitly enabled.
After the successful installation of the GI, you should be able to verify the cluster is in Flex Cluster.

```
crsctl get node role {config|status} [-node <nodename> | -all]
```

For example, to check the role of the current cluster node:

```
$ crsctl get cluster mode status
Cluster is running in "flex" mode
```

List all roles of all the nodes in a cluster by running this on any node:

```
$ crsctl get node role status -all
Node 'knewracn1' active role is 'hub'
Node 'knewracn2' active role is 'hub'
```

```
Node 'knewracn4' active role is 'hub'
Node 'knewracn6' active role is 'leaf'
Node 'knewracn5' active role is 'leaf'
Node 'knewracn8' active role is 'leaf'
Node 'knewracn7' active role is 'leaf'
```

Changing the Existing Cluster Mode to Flex Cluster

If your Clusterware was originally installed as a standard cluster, you can change its cluster mode from the standard cluster to a Flex Cluster. However, before you make this change, you need to first make sure that as a prerequisite the GNS is configured with a fixed VIP. The GNS may already be configured during the Clusterware installation. You need to ensure that the GNS is configured with a fixed VIP with the srvctl command, for example:

```
$ srvctl status gns
GNS is running on node knewracn2
GNS is enabled on node knewracn2
```

If the GNS is not enabled, you will see a result like this:

```
 $ srvctl status gns
PRCS-1065 : GNS is not configured.
```

Then, as the prerequisite, you need to convert this cluster from non-GNS cluster to GNS cluster by running the following command as the root user with a valid VIP address and a domain name:

```
#srvctl add gns –vip <VIP_address> –domain <doman_name>
```

And you need to enable Oracle Flex ASM option with ASMCA. Refer to the next section, Then, run this command as root to change the cluster mode from the standard mode to Flex Cluster mode:

```
#crsctl set cluster mode flex
```

You need to stop and restart Oracle Clusterware as root on each node:

```
#crsctl stop crs
#crsctl start crs –wait
```

Managing Oracle Flex Clusters

You can use various Oracle Clusterware Control (CRSCTL) utility commands to manage the Oracle Flex Clusters. In a Flex Cluster, you can change the role of a node between Hub node and Leaf node by running this crsctl command on the local node as the root user:

```
# crsctl set node role {hub | leaf}
```

Then, as the root user, you need to stop and restart Oracle High Availability Services on the node where you changed the role:

```
# crsctl stop crs
# crsctl start crs –wait
```

You also can check the current role of the node with this command:

```
$crsctl get node role config
Node 'knewracn2' configured role is 'hub'
```

If you need to change a node from a Leaf node to a Hub node, you may need to first check if the VIP exists on the node with this command:

```
$ srvctl config vip -n knewracn5
PRKO-2310 : VIP does not exist on node knewracn5
```

If there is no VIP for the node as in the example above, you need to add a VIP to this node by running the srvctl add vip command. The syntax of this command is as follows:

```
srvctl add vip -n node_name -A {name|ip}/netmask[/if1[if2|...]] [-k network_number] [-v]
# srvctl add vip -n knewracn5 -A 172.16.150.201/255.255.255.0/eth0 -k 1
```

You also can manage some of the settings of the Flex Cluster. To check the role of a particular node, run this command:

```
$crsctl get node role status -node knewracn5
Node 'knewracn5' active role is 'leaf'
```

To check the hubsize of the Flex Cluster, run this command:

```
$ crsctl get cluster hubsize
CRS-4950: Current hubsize parameter value is 32
```

To check the css misscount and leafmisscount setting, run the following commands:

```
$ crsctl get css misscount
CRS-4678: Successful get misscount 30 for Cluster Synchronization Services.

$ crsctl get css leafmisscount
CRS-4678: Successful get leafmisscount 30 for Cluster Synchronization Services
```

You also can get the details about a cluster node with this command:

```
$ crsctl status server knewracn7 -f
NAME=knewracn7
MEMORY_SIZE=4902
CPU_COUNT=2
CPU_CLOCK_RATE=4687
CPU_HYPERTHREADING=1
CPU_EQUIVALENCY=1000
DEPLOYMENT=other
CONFIGURED_CSS_ROLE=leaf
RESOURCE_USE_ENABLED=1
SERVER_LABEL=
PHYSICAL_HOSTNAME=
STATE=ONLINE
ACTIVE_POOLS=Free
STATE_DETAILS=
ACTIVE_CSS_ROLE=leaf
```

Flex ASM Architecture

Oracle Flex ASM Architecture

In 11gR2 or earlier ASM architecture or the standard 12c ASM, the ASM instance runs on every RAC node and the RAC databases have to rely on the ASM instance on the local node to do the storage I/O. This requirement has several limitations:

Requiring ASM instance running on every RAC instance consumes CPU and memory resources on each RAC node.

All Database instances on each node depend on the local ASM instance. The ASM instance failure will cause all local database instances to fail.

Oracle Flex ASM introduced in Oracle 12cR1 removes these two limitations. In Oracle Flex ASM architecture, only a small number of cluster nodes run Oracle ASM instances. These ASM instances are connected by all the database instances on the cluster to provide storage access to these database instances. When an ASM instance fails, the database instances that connected to the failed ASM instance reconnect to other ASM instances.

Oracle Flex ASM is an option in Oracle 12c ASM which you can enable or disable. If you choose to disable the Flex ASM feature, your cluster is a standard ASM cluster which is same as the ASM in 11gR2 or before, in which there is an ASM instance running on each RAC node and all the database instances have to access a local Oracle ASM instance.

If you choose to enable the Flex ASM feature in your cluster, the ASM configuration on your cluster becomes a Flex ASM. In this case, your cluster will have two kinds of nodes: those that have an ASM instance and those that don't have an ASM instance. Figure 4-5 shows a four-node Flex ASM configuration where three RAC nodes run Oracle ASM instances, which is set by default and can be reset with a cardinality setting. Node 4 doesn't have an ASM instance, and the database instance on node 4 connects to the ASM instance on node 1 for ASM service.

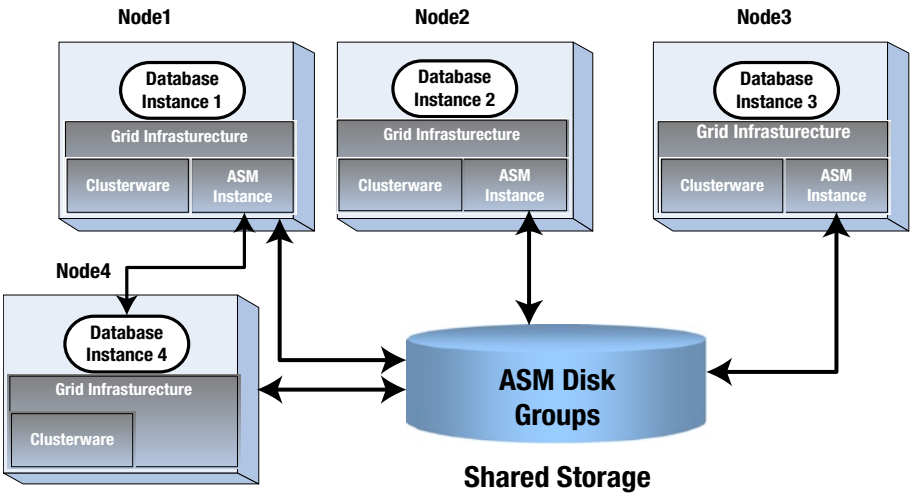


Figure 4-5. *An Oracle Flex ASM configuration example*

As shown in Figure 4-5, a Flex ASM consists of three kinds of Oracle ASM configurations:

1. Local Oracle ASM Clients: Oracle database instances such as those on nodes 1, 2, and 3 access the local ASM instances on the same node. In this case, these Oracle database instances have direct I/O access to the ASM disks in the shared storage and also access the Oracle ASM metadata in the local ASM instance. This configuration is same as for the ASM in 11gR2 or before.
2. Flex ASM Clients: Oracle database instances such as the one on node 4 have to connect to a remote ASM instance running in another node. This remote ASM instance is the ASM instance on node 1 in Figure 4-5.
3. Oracle ACFS and ADVM through Oracle ASM proxy instance: In Flex ASM configuration, Oracle ASM proxy is introduced to provide support for Oracle ACFS and Oracle ASM Dynamic Volume Manager (Oracle ADVM). Figure 4-6 shows the architecture of such a configuration. On the left node, the ACFS and ADVM talk to the ASM proxy instance, which connects to the local ASM instance. On the right node, the ACFS and ADVM connect to the ASM proxy instance, which connects to the remote ASM instance that is located in the left node, as the right node doesn't run local ASM instance.

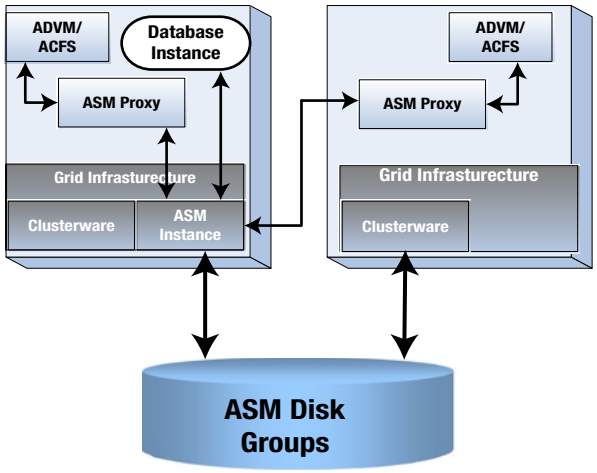


Figure 4-6. *ASM proxy for Oracle ACFS and Oracle ADVM*

In order for Flex ASM clients to connect to remote ASM instances, Flex ASM introduces a new type of network called the ASM network between Flex ASM clients and ASM instances. With this new type of network, inside of Oracle Clusterware 12c, there are four types of networks:

1. Public network, which usually connects the cluster nodes to the Corporate network;
2. Private network for the interconnect communication between the cluster nodes;
3. ASM network for ASM clients to connect to ASM instances;
4. Storage network for the cluster nodes to access the shared storage.

ASM network is needed only for a Flex ASM configuration.

Flex ASM and Flex Clusters

In Flex Clusters, Leaf nodes are not required to have direct storage access. In order for applications' Leaf nodes to access ASM disks on the shared storage through the Hub nodes, Oracle Flex ASM is required in Oracle Flex Clusters. If you configure an Oracle Flex Cluster during the Oracle GI installation OUI, the Oracle Flex ASM is enabled by default and the Flex ASM will be configured as a part of the GI installation. If you convert a standard cluster to a Flex Cluster after the OUI installation, you need to enable the Flex ASM on the cluster before changing the cluster mode to the Flex Cluster.

In a Flex Cluster configuration, ASM instances have to run the Hub nodes because Hub nodes are required to have direct access the shared storage. However, not all Hub nodes may run Oracle ASM instances. As shown in Figure 4-5, by default only three Hub nodes run Oracle ASM instances and other Hub nodes, such as node 4, run Flex ASM client without ASM instances on it. Therefore, in a Flex Cluster only a subset of the Hub nodes hosts the Flex ASM I/O service.

Oracle Flex ASM also can run on a standard cluster mode. In this case, all the cluster nodes are Hub nodes and only a subset of the standard cluster nodes run ASM instances.

Configuring Flex ASM

Oracle Flex ASM can be configured during the GI installation with OUI or can be converted from a standard ASM after the installation.

During the GI installation, as we mentioned previously, if you select the Flex Cluster option, the Flex ASM is automatically selected by default. If you select the standard Cluster option, you will have options to select the standard ASM or the Flex ASM. As long as the Flex ASM option is selected either by default or manually, as part of the GI installation, you will need to configure the ASM network by providing at least one physical network interface for the ASM network. In Oracle 12cR1, you can either have a separate network for the ASM network or have the ASM network and Private interconnects sharing the same physical network interfaces. Figure 4-7 shows a configuration where the ASM network and the Private interconnect share both eth1 and eth2 network interfaces.

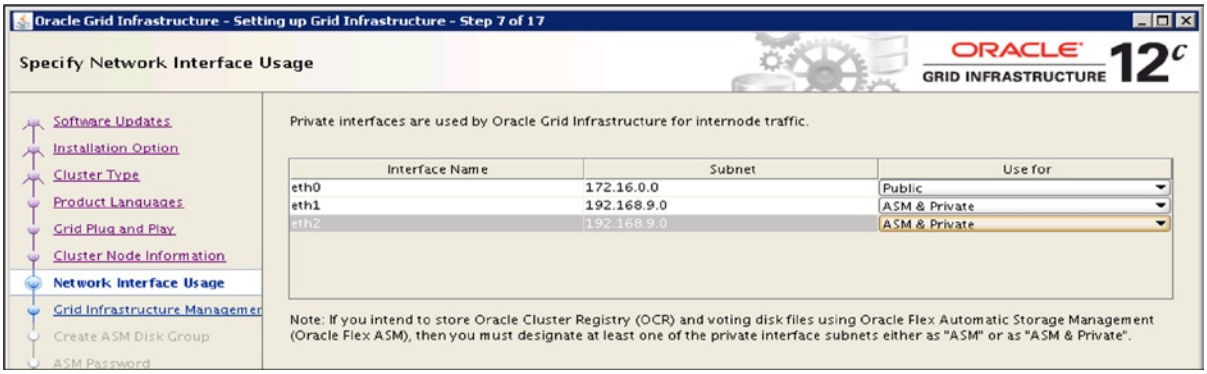


Figure 4-7. *ASM network and private interconnects share the same physical NICs*

The Flex ASM configuration is completed as a part of the GI installation process. By default, three-node Hub nodes will be chosen to run the ASM instances and the rest of the cluster nodes will be Flex ASM clients. The Flex ASM listener will also be created.

You also can enable the Flex ASM on a standard ASM cluster by converting the standard ASM to the Flex ASM. However, as the prerequisite, prior to the conversion, you need to have at least one network interface for the ASM network.

ASMCMD Command-line interface: asmcmd provides an option to convert a standard ASM to a Flex ASM. The syntax of this conversion command is as follows:

```
$asmca -silent -convertToFlexASM
-asmNetworks interface_name/subnet, interface_name/subnet...
-asmListenerPort ASM_listener_port_numbe
```

For example, we can convert a standard ASM cluster to a Flex ASM with the following command:

```
$asmca -silent -convertToFlexASM
-asmNetworks eth1/192.168.9.0, eth2/192.168.9.0
-asmListenerPort 1521
```

Then you also need to execute the converttoFlexASM.sh script as the root user on all the nodes, one node at a time. This command completes the conversion by restarting the ASM cluster to enable the Flex ASM. This converttoFlexASM.sh script is located in the directory of $ORACLE_BASE/cfgtoollogs/asmca/scripts.

Managing Flex ASM

Once you initially configure a Flex ASM instance through OUI or through the conversion method, there is not much specific work you need to do for this Flex ASM feature. You can do all the administrative work on a Flex ASM instance in the same way that you do for a standard ASM instance. There is not any specific instance parameter designed for the Flex ASM instance, and all the instance parameters for a Flex ASM instance are same as for a standard ASM instance.

You can check whether or not Oracle ASM is enabled in your cluster environment using the asmcmd command:

```
$asmcmd showclustermode
ASM cluster : Flex mode enabled
```

The SRVCTL status command shows the cluster nodes where ASM instances run:

```
$ srvctl status asm -detail
ASM is running on knewracn2,knewracn1,knewracn4
ASM is enabled.
```

And the SRVCTL config command shows more details about the Flex ASM configurations:

```
 $ srvctl config asm
ASM home: /u01/app/12.1.0/grid
Password file: +DATA1/orapwASM
ASM listener: LISTENER
ASM instance count: 3
Cluster ASM listener: ASMNET1LSNR_ASM
```

Notice that the ASM password is stored in ASM diskgroup, which is also an Oracle 12c ASM new feature.

ASM Clients and Relocating

With Flex ASM being enabled, an ASM instance may be connected with various ASM clients from either the local node or other remote nodes. After logging into the ASM instance, run the following query to see which ASM clients are connected to this ASM instance:

```
SQL>  select instance_name, db_name, status from v$asm_client;

INSTANCE_N DB_NAME    STATUS
---------- ---------- ----------
knewdb_2   knewdb     CONNECTED
+ASM2      +ASM       CONNECTED
+APX2      +APX       CONNECTED
-MGMTDB    _mgmtdb    CONNECTED
```

Using this query, we can see how these ASM clients will be failed over to another ASM instance. For example, we can see how the ASM clients that are connected to +ASM2 are failed over to +ASM1 after +ASM2 fails.

Before ASM2 fails, ASM1 instance has three clients: knewdb_1, +ASM1, and +APX1.

```
SQL> select instance_name, db_name, status from v$asm_client;

INSTANCE_N DB_NAME    STATUS
---------- ---------- ----------
knewdb_1   knewdb     CONNECTED
+ASM1      +ASM       CONNECTED
+APX1      +APX       CONNECTED
```

And ASM2 is connected with three clients: knewdb_2, +APX2, and -MGMTDB. After ASM2 fails, three clients from ASM2 instance are relocated to ASM1:

```
SQL> select instance_name, db_name, status from v$asm_client;

INSTANCE_N DB_NAME    STATUS
---------- ---------- ----------
knewdb_1   knewdb     CONNECTED
knewdb_2   knewdb     CONNECTED
+ASM1      +ASM       CONNECTED
+APX1      +APX       CONNECTED
+APX2      +APX       CONNECTED
-MGMTDB    _mgmtdb    CONNECTED
```

New ASM Storage Limits

There is a significant increase in the ASM storage limit pertaining to the number of ASM disk groups. ASM instance in 12c supports 511 disk groups in contrast to 63 in the previous release.

Replacing ASM Disk in Disk Group

Pre-12c, to replace an existing ASM disk in an ASM disk group for whatever reason, you had to add a new disk first and then drop the existing disk. With the new REPLACE CLAUSE in 12c, you can easily replace an ASM disk in an ASM disk group with just a single operation, in contrast to the previous add-and-remove disk procedure. The following example demonstrates how to replace an ASM disk (ensure you connected to an ASM instance as sysasm):

```
SQL> ALTER DISKGROUP dg_data REPLACE DISK dg_data_004 WITH '<new_asm_disk>' POWER 4;
```

Scrubbing ASM Disk Groups and Files

The new ASM SCRUB feature in 12c got the potential to verify any logical data corruption on any ASM disks, disk groups, or files and optionally repair them using ASM mirror disks when an ASM disk group is configured in either a high or normal redundancy level. The disk-scrubbing operation can be performed with the combination of HIGH, MAX, LOW, and/or AUTO option to control the I/O on the system, and this sort of operation typically will have very little I/O impact on the server.

The following examples demonstrate how to execute the command to verify logical corruptions as well as repair them automatically:

```
SQL> ALTER DISKGROUP dg_data SCRUB POWER AUTO:LOW:MAX:HIGH;
       -- Default POWER option is set to AUTO
```

The following example runs through logical data corruption on the data file and repairs any corruption using the mirror copy:

```
SQL> ALTER DISKGROUP dg_data SCRUB FILE '+DG_DATA/MYDB/DATAFILE/users.374.817049597'
REPAIR POWER HIGH;
```

If no REPAIR clause is specified, then only logical data corruption verification is performed. You can monitor the progress of the procedure in the V$ASM_OPERATION dynamic view.

Reading Data Evenly in ASM Disk Group

A new feature in 11g, Preferred Read Failure Groups, provides the functionality to read the data from an extent which is close to the node to improve the read efficiency for a disk group configured with a normal or high redundancy level. A new default feature in 12c, Even Read for Disk Groups, provides the ability to distribute the data read operation call evenly across all disks in a disk group. When a disk group is configured with a normal or high redundancy level, each read request can possibly be sent to the least-loaded source ASM disk to improve the read efficiency.

Measure and Tune Rebalance Operation

To measure any ASM disk group rebalance operation in order to tune or control the overall duration required for the rebalance operation, you can use the new EXPLAIN WORK FOR statement on AMS instance. The new SQL statement calculates the estimated work required for the rebalancing operation and puts the result in the V$ASM_ESTIMATE

dynamic view. This result will help you adjust the AMS POWER LIMIT values to rebalancing operation. The following demonstration explains how to do that:

```
EXPLAIN WORK FOR ALTER DISKGROUP dg_data REPLACE disk dg_data_004 WITH '<new_asm_disk>' POWER 3;

SQL> SELECT est_work FROM V$ASM_ESTIMATE;
```

Based on the result you get, you can increase or decrease the power limit of the disk rebalancing operation.

What-If Command Evaluation

Clusterware administrators often need to perform certain cluster management operations on the cluster environment. It has been a challenge to evaluate potential impact of these operations on the production of the cluster and RAC environment prior to these operations. Oracle Clusterware 12c provides a new feature called the What-if command Evaluation for this kind of purpose. This feature allows us to determine the impact of certain operations without actually executing these commands to the cluster. And of course, since the command has not actually been executed, the actual impact doesn't actually occur.

The What-if command Evaluation consists of a set of evaluation commands and what-if API to preview a cluster management operation. We can use these commands to analyze and determine the impact of a certain operation before actually executing the operation. As a result, this feature allows you to simulate the command and review potential results without actually making the changes on the system.

You can perform What-if command Evaluation application resources with CRSCTL using the eval option:

```
$crsctl eval {start | stop | relocate | modify | add | fail|  resource
```

You can perform What-if command Evaluation on Oracle Clusterware resource:

```
$ crsctl eval {add | delete | modify | serverpool
$ crsctl eval {add | delete | relocate| delete server
$ crsctl eval active policy
```

Here is an example that shows the potential impact of deleting server knewrac4 from the cluster:

```
$ crsctl eval delete  server knewracn4

Stage Group 1:
--------------------------------------------------------------------------------------------------------
--------
Stage  Number  Required   Action
--------------------------------------------------------------------------------------------------------
--------

               1           Y             Server 'knewracn4' will be removed from pools
                                           [ora.knewspool]
                            Y             Resource 'ora.ASMNET1LSNR_ASM.lsnr' (knewracn4)
                                           will be in state [OFFLINE]
                            Y             Resource 'ora.DATA1.dg' (knewracn4) will be in
                                           state [OFFLINE]
                            Y             Resource 'ora.LISTENER.lsnr' (knewracn4) will be
                                           in state [OFFLINE]
```

```
                              Y             Resource 'ora.LISTENER_SCAN2.lsnr' (1/1) will be
                                              in state [OFFLINE]
                              Y             Resource 'ora.asm' (3/1) will be in state
                                              [OFFLINE]
                              Y             Resource 'ora.knewdb.db' (2/1) will be in state
                                              [OFFLINE]
                              Y             Resource 'ora.knewracn4.vip' (1/1) will be in
                                              state [OFFLINE]
                              Y             Resource 'ora.net1.network' (knewracn4) will be
                                              in state [OFFLINE]
                              Y             Resource 'ora.ons' (knewracn4) will be in state
                                             [OFFLINE]
                              Y             Resource 'ora.proxy_advm' (knewracn4) will be in
                                             state [OFFLINE]
                              Y             Resource 'ora.scan2.vip' (1/1) will be in state
                                             [OFFLINE]

     2      Y          Resource 'ora.knewracn4.vip' (1/1) will be in
                         state [ONLINE|INTERMEDIATE] on server
                         [knewracn1]
             Y          Resource 'ora.scan2.vip' (1/1) will be in state
                          [ONLINE] on server [knewracn1]

     3       Y         Resource 'ora.LISTENER_SCAN2.lsnr' (1/1) will be
                         in state [ONLINE|INTERMEDIATE] on server
                         [knewracn1]
```

To evaluate the impact of these operations on resources with the .ora prefix, you must use the SRVCTL command with eval option. For example, to evaluate the failure consequence of DAT1 diskgroup by using SRVCTL command, use the following:

```
$srvctl predict diskgroup -g DATA1
Resource ora.DATA1.dg will be started on nodes knewracn1,knewracn2,knewracn4
```

PDBs on Oracle RAC

In 11gR2 and previous versions of the database, if we want to implement a multi-tenancy database by consolidating multiple sets of data for different business applications on a single database, we could put them into different database schemas of one database. However, it can be very challenging to manage security and access auditing on this type of configuration, as these different data sets share the same database. On the other hand, some applications such as Oracle E-Business Suite (EBS) Applications have fixed schema names such as APPS, AR, GL, etc. In this case, it is simply not allowed to consolidate more than one applications data set on the same database, as these data sets have identical schema names, which is not allowed in the same Oracle database. One alternative option is to run multiple database instances on the same database server, in which case you have multiple sets of database background processes and SGAs, etc. This will definitely increase overhead on performance and capacity requirements as well as administrative tasks.

Oracle 12c introduced a better solution to this multi-tenancy issue with the new architecture called PDBs. This feature essentially allows having multiple PDBs within a big CDB that is run with a database instance of the Oracle database software. This architecture can consolidate several data sets into separate PDBs within the CDB. And these PDBs are a self-contained and can be easily plugged or unplugged to a CDB.

This section will give brief coverage of PDB components. Then we will focus on how to run this PDB architecture in the Oracle RAC environment, such as configuring PDBs, managing PDBs, and accessing PDBs through services.

PDB Architecture Overview

In this PDB architecture, a PDB is a self-contained collection of schemas, schema objects, and non-schema objects that are similar to those in the traditional database. For example, if you want to consolidate multiple Oracle EBS databases, you can store each of these Oracle EBS applications' databases in a PDB. Another component of this architecture is the CDB, which is a subset of the PDBs that can contain zero, one, or more PDBs. A CDB has the following containers of the database objects.

The root named *CDB$ROOT* is a collection of schemas and objects to which all the PDBs of this CDB belong. The data dictionary of CDB stores the metadata about each PDB.

The seed named *PDB$SEED* is a template that is used to create a new PDB. You cannot make any change on the seed. The seed is named PDB$SEEd.

A PDB is a set of application schemas, which appears to be a traditional database.

Figure 4-8 shows the structure of a CDB that consists of five containers: root, seed and three PDBs, namely, pdb1, pdb2, and pdb3. As shown in Figure 4-8, each of these PDBs can be an Oracle EBS applications database which stores application schemas such as APPs, GL, AP, HR, etc. The identical schema name issue with multiple data sets is no longer a problem.

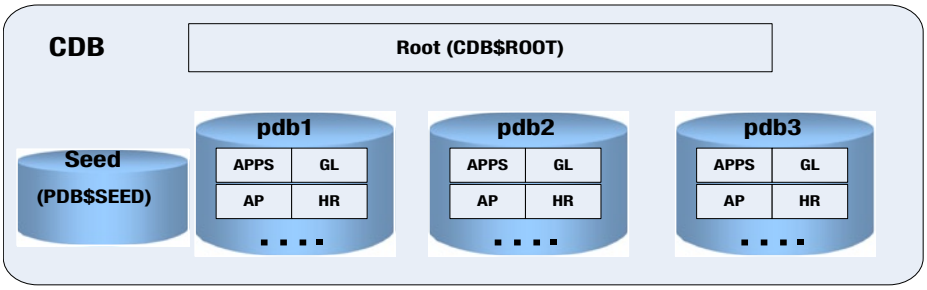


Figure 4-8. *Architecture of CDB with PDBs*

For example, the following SQL query lists these containers of a CDB:

```
SQL> SELECT NAME, CON_ID, DBID, CON_UID FROM V$CONTAINERS ORDER BY CON_ID;

NAME         CON_ID       DBID    CON_UID
-------- ---------- ---------- ----------
CDB$ROOT          1 1776735948          1
PDB$SEED          2 4057735188 4057735188
PDB1              3 3322827582 3322827582
PDB2              4 3881372142 3881372142
PDB3              5 3931156879 3931156879
```

Creating CDB and PDBs

A CDB can be created using Database Configuration Assistant (DBCA) in the same way a non-CDB database is created. Figure 4-9 shows the specification of CDB creation with three PDBs in DBCA. With these specifications, the DBCA will create the database with global database name `cpdb.kcloud.dblab.com` with the three PDBs: pdb1, pdb2, and pdb3. With the creation of the CDB and the PDBs, the default corresponding database services are created for the CDB and each of the PDBs. These database services have the same names as the CDB and PDBs.

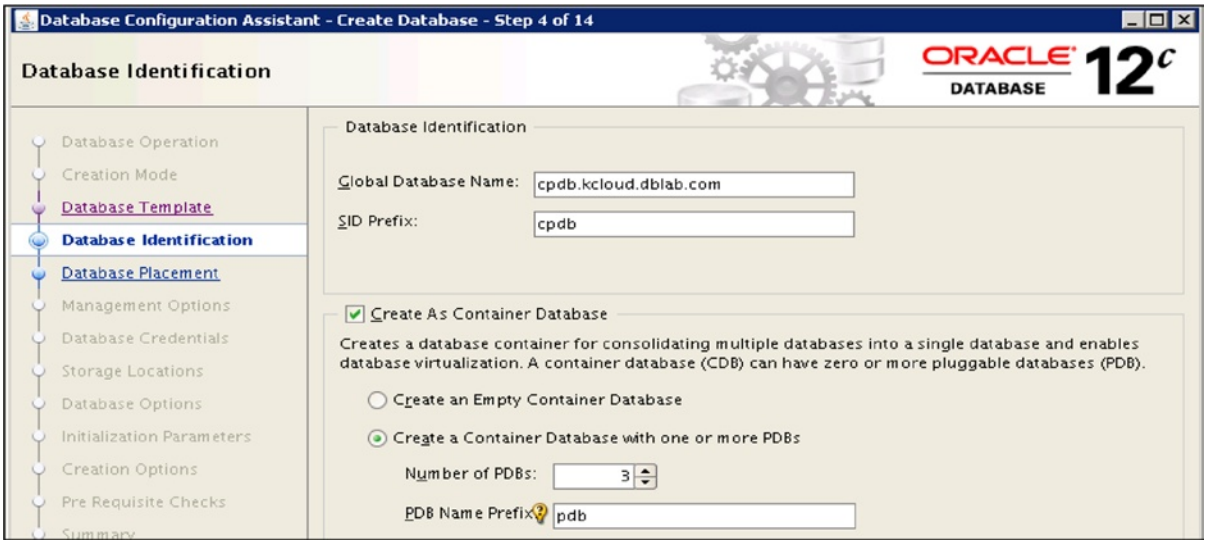


Figure 4-9. *Creation of CDB with PDBs with DBCA*

Service `cpdb.kcloud.dblab.com` for the CDB root.

Service `pdb1.kcloud.dblab.com` for the pdb1 PDB.

Service `pdb2.kcloud.dblab.com` for the pdb2 PDB.

Service `pdb3.kcloud.dblab.com` for the pdb3 PDB.

You also can create additional PDBs in the container CDB with SQL*Plus commands. There are several ways to add a PDB to a CDB:

- Creating PDB using the seed
- Cloning an existing PDB
- Plugging an unplugged PDB to the CDB
- Creating PDB from a non-CDB

Refer to the "Creating and Removing PDBs with SQL*Plus" section of the *Oracle Database Administrator's Guide 12c Release 1 (12.1)* for detailed steps.

Connect to CDB Root and PDBs

Once the CDB and PDBs are created, the listener listens to the default services for CDB root and PDBs:

```
$lsnrctl status
....
Service "cpdb.kcloud.dblab.com " has 1 instance(s).
   Instance "cpdb1", status READY, has 1 handler(s) for this
Service "pdb1.kcloud.dblab.com" has 1 instance(s).
  Instance "cpdb1", status READY, has 1 handler(s) for this service...
Service "pdb2.kcloud.dblab.com" has 1 instance(s).
  Instance "cpdb1", status READY, has 1 handler(s) for this service...
Service "pdb3.kcloud.dblab.com" has 1 instance(s).
  Instance "cpdb1", status READY, has 1 handler(s) for this service...
...
```

The corresponding net service names for these database services are defined in the tnsnames.ora, which will allow us to connect to the CDB root and PDBs. For example, we have the net service name 'cpdb' for the CDB root and the pdb1 net service for the pdb1 PDB:

```
cpdb =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = knewrac-scan.kcloud.dblab.com
                                PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = cpdb.kcloud.dblab.com)))
pdb1 =
  (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = knewrac-scan.kcloud.dblab.com )
                             (PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = pdb1.kcloud.dblab.com)))
```

To connect to the root (CDB$ROOT), you can use the operating system authentication that has the SYSDBA administrative privilege:

```
$sqlplus / as sysdba or
SQLPLUS> connect / as sysdba
```

Or connect with the net service name for the root in the CDB:

```
SQLPLUS> connect system@cpdb
```

To connect to the PDB, you need to connect to a net service name with the PDB property. For example, to connect the pdb1 PDB on SQLPLUS through the new service name pdb1:

```
SQLPLUS> connect system@pdb1
```

Once you connect to the pdb1 PDB, you can do administrative tasks such as creating schema or other database objects in the PDB. These default database services should be used only for administrative tasks. Additional database services should be created for the applications to connect to the PDBs. The next section will discuss how to create additional services for PDBs in the Oracle RAC environment.

PDBs on Oracle RAC

The CDB with PDBs can be created and configured on a single-node or a multiple-node RAC environment. Similarly to a non-CDB database, a RAC-based containter database can be shared and accessed through multiple database instances on the RAC nodes. However, the CDB has introduced several layers of containers in the database: the entire CDB, the root, the seed, and PDBs. Similar to the non-CDB, a PDB can be on different open modes:

- Mounted
- Read only
- Read and write

You can check the open modes of all the PDBs on a RAC instance when you connect to the CDB root:

```
SELECT NAME, OPEN_MODE, RESTRICTED FROM V$PDBS;

NAME     OPEN_MODE  RESTRICTED
-------- ---------- ----------
PDB$SEED READ ONLY  NO
PDB1     READ WRITE NO
PDB2     READ WRITE NO
PDB3     Mounted    NO
```

To start a PDB, connect to the PDB with SQL*Plus and run the startup or the shutdown command with different startup open options:

```
STARTUP OPEN
STARTUP OPEN READ ONLY
STARTUP RESTRICT OPEN READ ONLY
SHUTDOWN IMMEDIATE
```

You also can change the open mode of a PDB using the ALTER PLUGGABLE DATABASE statement, for example, 'ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE' to close the PDB, 'ALTER PLUGGABLE DATABASE OPEN READ ONLY' to open the PDB for read only.

Figure 4-10 shows such a RAC-based CDB database architecture. Each PDB may be in the open mode in different RAC instances: the pdb1 PDB is open on instance 1; pdb2 is open on instance 2; pdb3 is open on instances 1, 2, and 3; and pdb4 is open on instance 3.

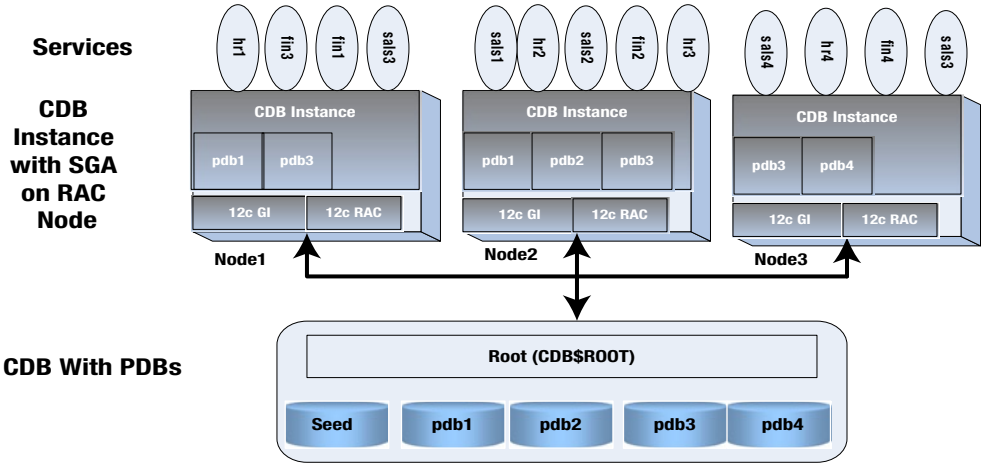


Figure 4-10. *Oracle RAC-based CDB*

Creating Services Associated with PDBs

PDBs in a RAC-based CDB should be managed through the database services. The default database service with the same name as the PDB is created automatically during PDB creation. You should only use this default service to connect to the PDB for administrative tasks. For applications to access the PDB, you need to create user-defined services for the PDB. In Oracle 12c, the database service has an optional PDB property which allows you to create a database service that is associated with a PDB. For example, we can create two database services, hr1 and sales1, to be associated with the pdb1 PDB with this command:

```
$srvctl add service -db cpdb -service hr1 -pdb pdb1 -preferred cpdb1
  -available cpdb2

$srvctl add service -db cpdb -service sales1 -pdb pdb1 -preferred cpdb2
 -available cpdb3
```

Notice that these services have two different preferred RAC instances: cpdb1 on node1 and cpdb2 on node2. Figure 4-10 shows the different database services associated with four PDBs: pdb1, pdb2, pdb3, and pdb4. The Data Dictionary view all_services lists all the services associated with the CDB and the PDBs: List the services with the CDB root:

```
SQL> connect sys@CPDB as sysdba
SQL> select NAME, PDB from all_services;

NAME                 PDB
-----------          ------------------
SYS$BACKGROUND       CDB$ROOT
SYS$USERS            CDB$ROOT
cpdbXDB              CDB$ROOT
cpdb.kcloud.dblab.com CDB$ROOT
```

List the services with the pdb1 PDB:

```
SQL> connect sys@pdb1 as sysdba
SQL> select NAME, PDB from all_services;

NAME                      PDB
-----------               ------------------
pdb1.kcloud.dblab.com      PDB1
hr1                        PDB1
sales1                     PDB1
```

In the Oracle RAC-based environment, these database services are managed through the Oracle 12c Clusterware. You can check the service configuration of the cpdb CDB with the following srvctl command:

```
$ srvctl config  service -db cpdb
Service name: hr1
Service is enabled
Server pool: cpdb_hr1
Cardinality: 1
......
Pluggable database name: pdb1
......
Preferred instances: cpdb1
Available instances: cpdb2
Service name: sales1
Service is enabled
Server pool: cpdb_sales1
Cardinality: 1
......

Pluggable database name: pdb1
......
Preferred instances: cpdb2
Available instances: cpdb3
......
......
```

In the previous section, we mentioned that PDBs can be opened or shut down with the startup and shutdown command as well as the ALTER PLUGGABLE DATABASE OPEN or the ALTER PLUGGABLE DATABASE CLOSE command. Starting the service that is associated with the PDB can also open the PDB. If the PDB is currently closed and you use the SRVCTL command to start the service that is associated the PDB, the PDB will be opened in read/write on the nodes where the service is started. The following is the test:

Close the pdb1 PDB:

```
SQL> connect sys@pdb1 as sysdba
SQL> alter pluggable database close immediate;
SQL> select NAME, OPEN_MODE from V$PDBS

   NAME  OPEN_MODE
  ------ --------------
   PDB1  MOUNTED
```

Start the hr1 service:

```
$srvctl status service -db cpdb -service hr1
Service hr1 is not running
$ srvctl start service -db cpdb -service hr1
```

Check the status of the pdb1 PDB:

```
SQL> connect sys@pdb1 as sysdba
SQL> select NAME, OPEN_MODE from V$PDBS

NAME   OPEN_MODE
------ --------------
PDB1  READ WRITE
```

While the startup of the service can open the closed PDB with which the service is associated, shutdown operation of the service will not change the open mode of the PDB. In other words, if we shut down the hr1 service while the pdb1 PDB is open for READ and WRITE, the pdb1 PDB keeps open for READ and WRITE.

Creating Net Service Names for PDBs

With the user-defined services with the property of the PDB, you can create the net service names in the tnsnames.ora file for applications to access the PDBs. Using the previous example of the hr1 and sales1 services for the pdb1 PDB, we can create the HR_PDB1 and SALES_PDB1 net service names, which the HR application uses and the sales application can use, respectively, to connect to the pdb1 PDB database:

```
HR_PDB1 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = knewracscan.kcloud.dblab.com)
                               (PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = hr1.kcloud.dblab.com)
    )
  )
SALES_PDB1 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = knewracscan.kcloud.dblab.com)
                               (PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = sales1.kcloud.dblab.com)
    )
  )
```

12cR1: Miscellaneous New Features for RAC

In addition to the significant new features presented in detail in the previous sections, the following sections present a brief high-level synopsis and overview of some of the notable miscellaneous new features within Oracle 12cR1 Clusterware.

Public Networks for RAC: IPv6 Support Added

RAC 12cR1 now supports IP addresses for Public and VIP based on the IPv6 protocol. This new feature gives RAC users a choice of IPv4 or IPv6 protocol addresses. With 12cR1, Oracle database clients are pointed by the SCAN listener to the suitable listener for either Ipv4 or IPv6 IP protocols. IPv6 is also supported for the GNS with the Stateless Address Autoconfiguration Protocol.

The important thing to remember is that during the installation phase, you cannot mix and match both IPv4 and IPv6 protocol configurations for VIP and SCAN IP addresses: you have to use either IPv4 or IPv6, but not both at the same time, at the time of installation. However, once installation is complete, later on, IPv4 addresses can be added to an existing IPv6 cluster and vice versa.

It should also be noted that the IPv6 protocol is not supported and cannot be configured for the Cluster Private Interconnect (it is supported only for PUBLIC, SCAN, and VIP addresses).

Global Data Services

This is an exciting new feature that enables common services to operate for a group of replicated Oracle RAC or non-RAC databases using Data Guard, Golden Gate, or another replication technology. This new feature can be utilized for workload management functionality purposes in a distributed ecosystem: for example, Service Management, Failover, and Load Balancing, etc., for replicated databases providing a set of common services.

Online Resource Attribute Modification

Online Resource Attribute Modification is a great new feature that makes it simpler and easier to perform the dynamic editing, modification, and implementation of changed attributes related to resources in HOT mode, without the need to restart the resource. SRVCTL and CRSCTL commands can be utilized for performing Online Resource Attribute Modification without requiring a restart for the change to take effect.

RAC 12cR1: Policy-Based Management and Administration

With 12cR1, Cluster Configuration Policy Sets enable date- and time-sensitive application of policies in a RAC cluster. In the modern era of Cloud Computing, this feature comes in real handy for on-demand, consolidated, and dynamic management of highly available heterogeneous applications and workloads.

Cluster Configuration Policies define attributes for server pools such as node availability, resource placement, etc. Policy Sets take this approach a bit further by grouping Cluster Configuration Policies for server pools. Cluster Configuration Policy Sets groups various policies (one or more) together for a server pool.

Cluster Configuration Policies can be used for implementing policy-based management and administration for different days and hours of the week, depending on type of workload, peak or off-peak timeframes, server categories and capabilities, etc.

ASM Disk Group: Shared ASM Password File

Within 12cR1 ASM disk groups, shared password files are now supported, making it easier and more efficient for maintenance purposes by keeping a single copy of a shared ASM password file.

Valid Node Checking: Restricting Service Registration

With this new feature, SCAN listener registration can be restricted to a specific set of nodes and subnet using the SRVCTL command, thereby making the whole RAC experience more secure and easier to implement and maintain.

12cR1: Shared GNS

With 12cR1, the GNS can be used and shared among multiple RAC clusters, instead of just one GNS per cluster as was the case in earlier versions of RAC.

RAC 12cR1: Restricting Service Registration

With 12cR1 by default, listener registration is now restricted to local IPs only. With this new feature, IP addresses or subnets can be configured dynamically to allow listener registration requests. The SRVCTL command-line utility can be used for this purpose.

Oracle ASM, ACFS, and ADVM: Improvements and New Features

As part of Oracle GI, with 12cR1, there are multiple improvements and new features within ASM, ACFS, and Oracle ASM Dynamic Volume Manager (Oracle ADVM), for example, improved ACFS support for Oracle Grid Homes, new ACFS security features and options to augment OS-level security features, support for implementing combination security of Oracle Audit Vault and Oracle ACFS security, ACFS highly available NFS, ACFS snapshot improvements, API for ACFS file tags, ASM rebalance improvements, support for ASM chown/chgp/chmod for files that are currently open, API for ACFS plug-ins, Oracle Enterprise Manager support for new ACFS features, ASM file access control for the Windows OS family, support for OCR backups in ASM disk groups, ACFS support for all kinds of Oracle DB files, more OS family support for ACFS tagging and replication, ASM disk resynch improvements, enhanced support for Cluster resources for ACFS and ADVM, rolling migration for GI one-off patches, ASMCMD improvements and enhancements, new SQL statement functionality for ASM, etc.

NFS High Availability

With 12cR1, Oracle ACFS can be configured to provide fault-tolerant failover for NFS exports. NFS exports can be mounted with VIP addresses. This new capability ensures that NFS exports are fault tolerant and highly available.

12cR1: CHM Enhancements

With 12cR1, CHM has been improved to be a highly available monitoring tool that provides and reflects a consolidated cluster view by using data collectors on the nodes of the cluster. CHM now also has improved capability to detect failures and other issues. CHM now also supports Flex Clusters.

Windows: Support for Oracle Home User

With RAC 12cR1, an Oracle home user tied with a (non-admin) Windows domain user can be specified during the installation phase within the Windows OS family. This new functionality enables a limited number of privileges for the Oracle home user, restricting access to Oracle products.

The password for the Oracle home user can be stored within a wallet. Oracle tools can implicitly use the password from the wallet, thereby alleviating the need for the user to specify the password.

OUI: Enhancements and Improvements

With RAC 12cR1, OUI has been improved in multiple facets, providing support for the new RAC enhancements like Flex Cluster, Flex ASM, etc. Commonly used configurations can also now easily be used for installation during the initial interview process in OUI.

12cR1: Installations/Upgrades—Running Scripts Automatically

With RAC 12cR1, scripts can be set up to run automatically at the tail-end of installations and upgrades, making the whole process easier and minimizing the need for human intervention, thereby mitigating overall errors as well using this methodology.

12cR1: Introducing Application Continuity

With 12cR1, application availability and continuity have been significantly enhanced by masking planned and unplanned outages and failures (network, storage, processes, etc.) from application users.

This new feature has been implemented in a manner that is independent of the way the application behaves and operates in case of a failure in RAC.

Application continuity can be implemented at the database service level by the following steps:

Setting the -failovertype parameter to TRANSACTION.

Setting the -commit_outcome parameter to TRUE.

Application continuity is transparently available to applications using Oracle connection pools (UCP), Weblogic Server Active GridLink, and JDBC.

The next section is an extension and continuation of the application continuity paradigm and functionality.

Transaction Idempotence and Java Transaction Guard

Transaction idempotence is an exciting new feature that enhances the RAC experience by enabling application recovery by masking most of the RAC system-level failures from the user. Java Transaction Guard allows Java to use application continuity features.

Deprecated and Desupported Features

In this section, features and options are listed that are either deprecated or not supported with 12cR1. It is very important to understand these points before you plan to upgrade your existing environment and/or plan a new 12c cluster environment. The deprecated, desupported features include the following:

- Raw (block) storage devices are no longer supported by Oracle with 12c. Therefore, consider migrating the OCR/voting disk to ASM if they are on raw storage devices and if you have plans to upgrade your existing environment to 12c.
- Oracle no longer supports the Oracle Cluster File System (OCFS) on the Windows OS family.
- Single-letter commands for the SRVCTL utility, for example, -n –s, have been desupported.

Summary

With a whole array of exciting new features ranging from Flex Clusters, Flex ASM, What-if command Evaluation, and RAC-based PDBs to Application Continuity, 12cR1 brings about real change and enhancements to the overall RAC experience. These new features reflect changing times in the industry and the evolution of Cloud Computing within the IT ecosystem. Most of the new features related to RAC 12cR1 reflect the significance of the role that RAC plays within the Cloud Computing space from a Database-as-a-service (DBAAS) model perspective.

CHAPTER 5

Storage and ASM Practices

by Kai Yu

In this chapter, I will discuss one of the key infrastructure components for the Real Application Cluster (RAC) database: storage. In the RAC Database environment, storage needs to be accessible and shared by all the nodes of the RAC Database. Database files such as data files, control files, online redo log files, temp files, spfiles, and the Flash Recovery Area (FRA) are kept in this shared storage. In addition, two key sets of files of the Oracle Clusterware—Oracle Cluster Registry (OCR) and voting disk files—are also stored here. Due to the presence of these key components of the RAC Database and Clusterware, shared storage is one of the most critical components of the RAC Database infrastructure.

This shared storage plays a key role in the stability of the Oracle Clusterware. In case any node of the RAC cluster is not able to access the voting disk files that are stored in the storage within a predetermined time threshold (default is 200 seconds), the RAC node will be evicted and get rebooted. As a result, the database instance on the RAC node will also get rebooted. Even more seriously, if the OCR on the storage is lost permanently, the Oracle Clusterware on the entire RAC cluster will no longer function. If this happens, the OCR needs to be recovered from its backup to resume the normal operation of Oracle Clusterware.

Shared storage is also essential for the availability of the RAC Database. Any storage issue may lead to the loss of a partial or entire RAC Database. For example, losing access to shared storage by a RAC node will bring down the RAC Database instance on the RAC node. Loss of a data file or data corruption due to disk errors may cause the loss of a partial or entire RAC Database.

The performance of the RAC Database depends upon the I/O performance of the shared storage. Storage I/O performance is even more important in a RAC Database environment where the heavier I/Os from multiple RAC Database nodes to a single shared storage may trigger huge I/O bottlenecks that hinder database performance.

As an essential part of the life cycle management of a RAC Database, the design, configuration, and management of such a shared storage system is vital for the long-term health and performance of the RAC Database. However, in a typical IT department, storage administration and database administration are two separate job responsibilities that may belong to two different teams, in line with the separation of duties policy commonly followed in IT. Cooperation and mutual understanding between the storage administrator and the database administrator (DBA) are crucial for RAC Database storage design and configuration. In order to achieve database availability and performance requirements defined by the database SLA (Service Level Agreement), some special design considerations and best practices should be incorporated into the storage provisioning process for Oracle RAC Databases. Here we can highlight some of the major tasks of storage provisioning and examine how the different roles such as the storage administrator, the system administrator, and the DBA can play together.

1. Architecting and implementing the storage solution. Since shared storage is required by the RAC infrastructure, many IT departments use SAN (Storage Area Network) storage for the RAC Database, as it can be accessed by multiple hosts through the storage network. The reality is that a big SAN storage may be also shared by many different applications. The storage administrator and the DBA need to work together to make sure the storage requirements of the RAC Database are met. Some key areas for storage design include storage network protocol, the topology of the physical network connections among the RAC node hosts, and storage, storage capacity, and I/O load balance among the applications that share the same storage.
2. Provisioning storage volumes in the shared storage for the Oracle RAC Databases. At the very least, you need to provision the storage volume for OCR and the voting disk files and the volumes for the database files. The goal is to ensure that these volumes meet the high availability and I/O performance and capacity requirements of the RAC Database with the optimal design and configuration of these storage volumes, including the RAID configuration of the volume; the capacity and number of disk spindles and what kind of disks (speed of disks) form the storage volume; and the storage controller to which the storage volume should be assigned.
3. Making the storage volumes accessible to the RAC nodes. This includes configuring the logical network connections to the storage volumes and presenting the storage volumes as OS devices in all the RAC nodes.
4. Creating Automatic Storage Management (ASM) diskgroups on the storage volumes for OCR and voting disk files of the Oracle Clusterware.
5. Creating ASM diskgroups for database files and optional ASM Cluster File System (ACFS) for non–Oracle Database files such as Oracle RAC home and clusterfile system for other applications.
6. Ongoing maintenance tasks such as monitoring database I/O performance and identifying storage I/O issues and performing storage reconfiguration, upgrade, or migration tasks.

In the preceding task list, tasks 1 and 2 are usually completed by the storage administrator and the OS system administrator with input from the Oracle DBA. Task 3 is performed by the OS system administrator with input from the Oracle DBA. Tasks 4–6 are traditionally the Oracle DBA's responsibility. In many IT organizations, especially smaller IT departments, these responsibilities are combined and performed by one system administrator or DBA.

This chapter covers some of the techniques and best practices that are used for shared storage design and configuration of the RAC Database, with a focus on how to meet the special storage requirements for the Oracle Clusterware and RAC Database. These topics will be covered in this chapter:

- Shared architecture and configuration for RAC
- ASM
- Storing OCR and voting disk files in ASM
- ACFS

Storage Architecture and Configuration for Oracle RAC

In the Oracle RAC Database cluster, the cluster nodes connect directly to the same storage array where the critical components of the Oracle Clusterware OCR and voting disk files and the database files are stored. The availability of the storage array for each cluster node and the I/O performance of the storage array are critical to the Oracle

RAC Database. Optimal storage array design and configuration are the foundation of Oracle RAC Database design. Understanding the optimal configuration of the storage array and how it contributes to the Oracle RAC Database design help us make the right design decisions and establish the optimized infrastructure for Oracle RAC Database at the very start of RAC Database configuration.

■ **Note** Oracle RAC 12c introduced a new cluster architecture called Oracle Flex Cluster in addition to the standard cluster configuration, which is similar to the Oracle 11gR2 cluster. In the Oracle 12c Flex Cluster, there are two types of cluster nodes: Hub nodes and Leaf nodes. All the Hub nodes have direct access to the shared storage, while the Leaf nodes do not require direct access to shared storage. These Leaf nodes get data from the shared storage through Hub nodes. (Please refer to Chapter 4 for more details.) This chapter will focus mainly on the standard cluster in which all the nodes have direct access to the shared storage.

Storage Architecture and I/O for RAC

Before we design the storage architecture, you need to understand the characteristics of I/O operations between the Oracle RAC nodes and storage. The storage architecture for an Oracle RAC environment consists of OCR and voting disk files of the Oracle Clusterware and database files and online redo logs for Oracle RAC Database. In theory, these storage components can be stored in one physical volume. However, for optimal I/O performance and better manageability, separate physical volumes may be created for each of the components.

To examine the I/O operations between the RAC Database nodes and the shared storage, for example, we create four volumes for a two-node RAC Database as illustrated in Figure 5-1. These four volumes are the two online redo log volumes, one OCR and voting disk volume, and one data volume. Each of the two online redo log volumes is for one one redo log thread of one RAC node. The data volume is for database files, control files, etc. Online redo logs and database files are very critical components of the RAC Database.

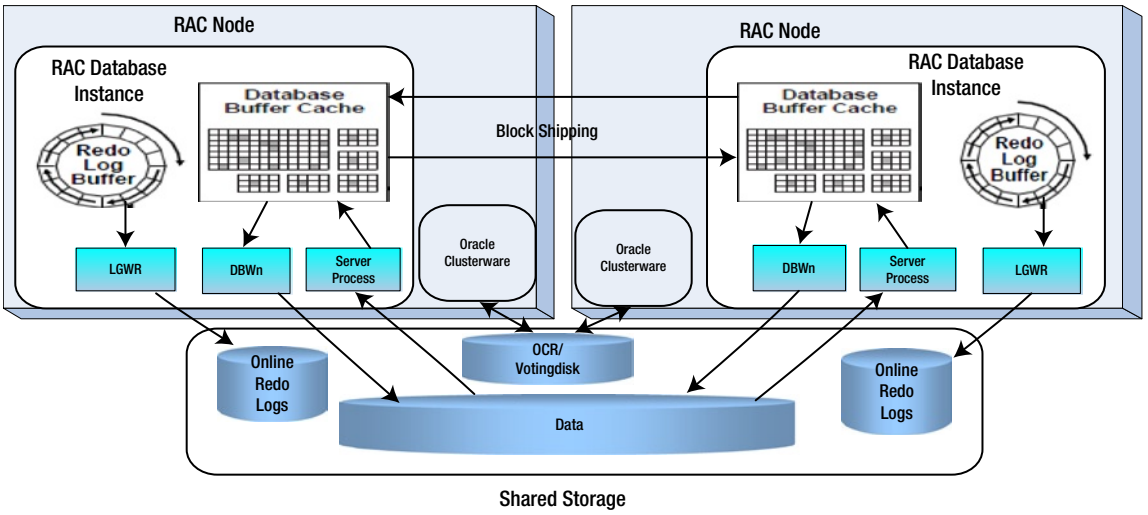


Figure 5-1. *Oracle RAC Database architecture*

The OCR and voting disk volume is for the OCR and voting disk file. The OCR stores the metatada of the resources that Oracle Clusterware manages, such as Oracle RAC Database, listeners, virtual IPs, and service. The voting disk files are used to store and manage the node membership of the cluster. Oracle Clusterware processes Cluster Ready Service (CRS), and Cluster Synchronization Services (CSS) on each RAC node constantly accesses OCR and voting disks. OCR and voting disks need to be accessible from each of the RAC nodes all the time. If a RAC node fails to access the voting disk file in 200 seconds, it will trigger a node eviction event that causes the RAC node to reboot itself. Therefore, the critical requirement for OCR and voting disk is the availability and fault tolerance of the storage volume.

To understand how the Oracle RAC Database accesses shared storage, let's review the Oracle RAC Database architecture. Figure 5-1 shows a two-node RAC Database. Each of the RAC Database instances has a set of database background processes, such as the log writers (LGWR), DB writers (DBWn) and server processes, etc., along with RAC-specific processes such as LMS, LMON, LMD, etc. Each RAC instance also has its memory structure, including the System Global Area (SGA) memory structure, where database buffer cache and redo log buffers are located. The database buffer cache is the memory area that stores copies of data blocks read from data files. The redo log buffer is a circular buffer in the SGA that stores redo entries describing changes made to the database.

When a user sends a query request to the database instance, a server process is spawned to query the database. The block request is sent to the master instance of the block to check if this block has been read into any instance's buffer cache. If the blocks cannot be found in any instance's buffer cache, the server process will have to get the block from the storage I/O by reading the data block from the data files to the local buffer cache through data file read operations. If the data block is found in the buffer cache of one or more RAC instances, the instance will ask the Global Cache Service (GCS) which is the LMS process to get the latest copy of the block. If the latest copy is on a remote instance, the copy will be shipped from the buffer cache of the remote instance to the local buffer cache. In this way, Oracle cache fusion moves the current blocks between the RAC instances. As long as the block is in an instance's buffer cache, all other instances can get the latest copy of the block from the buffer cache of an instance instead of reading from the storage.

Different types of application workloads determine the way in which RAC Database instances interact with the storage. For example, the data file read can be "random read" or "sequential read." For online transaction processing (OLTP) type database workloads, most queries involve small random reads on the data files by taking advantage of the index scan. For data warehouse or decision support (DSS) workloads, the queries involve large sequential reads of the data files due to large full-table scan operations. In order to achieve optimal I/O performance for the OLTP workload, it is very critical to have a fast I/O operation, as measured by IOPS (I/O operations per second) and I/O latency. The IOPS number is about the I/O throughput, namely, how many I/O operations can be performed per second, while I/O latency is defined as the time which it takes to complete a single I/O operation.

One way to achieve higher IOPS is to have the data striped across multiple disk drives so that these multiple disk drives can be read in parallel. Another more promising solution is to use Solid State Drives (SSD), which can significantly increase IOPS and reduce I/O latency by removing the performance bottleneck created by the mechanical parts of the traditional hard disk. For DSS workloads, it is important to be able to read a large amount of data contiguously stored in the disk to the buffer cache at high speed (as defined by MBPS (megabytes/second)). The bandwidth of the components linking the server with the storage such as HBAs (Host Bus Adapters), storage network protocol, physical connection fabrics, and the storage controllers is the key to this type of performance. In reality, many database applications fix these two types of workloads. When we look for storage for the database, its IOPS, MBPS, and I/O latency should be evaluated to ensure that they meet the database I/O requirements.

Besides reading data from storage, writing data to storage is also critical to RAC Database performance. This is especially true for the OLTP-type database workload. Two important disk write operations are as follows:

- writing redo logs from the redo log buffer in SGA to the online redo log files in the storage by the logwriter process (LGWR).
- writing modified blocks (dirty blocks) in the buffer cache to the data files by the DB writer process (DBWn).

While writing dirty blocks by DBWR involves a random write operation, writing redo logs by LGWR involves a sequential write operation. In order to guarantee that a data change made by a transaction is fully recoverable, the transaction would have to wait until all the redo logs for the transaction and the system change number (SCN) of the transaction are written to the online redo logs. In an OLTP database with high-volume transactions, this sequential writing operation by LGWR can be the performance bottleneck that holds up these database transactions. A high number of 'log file sync' wait events shown in the AWR report is a good indication of this performance bottleneck. In order to optimize database performance, we need to focus on how to improve the LGWR's I/O performance by allocating the online redo logs on storage that has high IOPS and low I/O latency.

Although user transactions don't have to wait for the DBWR directly. a slow DBWR writing operation could still delay user transactions. As we know, at a database checkpoint, the DBWR process needs to write all the dirty blocks to the data files in the storage. A slow DBWR process can delay the checkpoint operation; for instance, if during the checkpoint the logwriter needs to reuse a log file that has redo information on a dirty block. The logwriter process has to wait until DBWR finishes writing the dirty block for the checkpoint. The logwriter wait caused by the slow DBWR process will slow down the user transactions. In this case, you will see the "Checkpoint not complete, cannot allocate new log" message in the alert.log file. This indicates that the logwriter had to wait and transaction processing was suspended while waiting for the checkpoint to complete. Therefore, the speed of writing dirty blocks to data files by DBWR impacts database performance.

Having explained the requirements of storage I/O under different database workloads and its impact on database performance, in the next section I will discuss the technology options that we should consider in the storage architecture design for the Oracle RAC Database.

RAID Configuration

As you saw in the last section, the availability and performance of a RAC Database depends on the shared storage infrastructure configuration. One of the most important storage configurations is RAID (Redundant Array of Inexpensive Disks), which was introduced as a way to combine multiple disk drive components into a logical unit. The following are some of the commonly used levels of RAID configuration adopted to enhance storage reliability and performance:

- *RAID 0 for block level striping*: data blocks are striped across a number of disks in a sequential order. This is often used to improve storage performance as it allows reading data from multiple disks in parallel. As RAID 0 doesn't provide mirroring or parity, any drive failure will destroy the array.
- *RAID 1 for mirroring*: two or more disks have exact copies of the data. This is often used to achieve reliability against disk failures, as it saves more than one copy of the database. This can improve the data read performance, as the data read can get the data from the faster of multiple copies. But it will slow down the data writes, as the controller needs to write more than one copy of the data. It also cuts the storage capacity and the number of disks in half. This option requires minimal two disk drives. RAID 1 doesn't provide striping.
- *RAID 1+0 for mirroring and block level striping*: As shown on the right side of Figure 5-2, this option create mirror sets by mirroring the disks, then stripe data blocks such as B1, B2 . . . across these mirrored sets so it is also called as "stripe of mirrors". It provides both reliability and performance improvement.
- *RAID 0+1 for block level striping and mirroring*: As shown on the left side of Figure 5-2, this option create two striping sets each with data blocks such as B1, B2 . . . , and then let them mirror each other, so it is also called as "mirror of stripes". This option achieves both reliability and performance improvement.

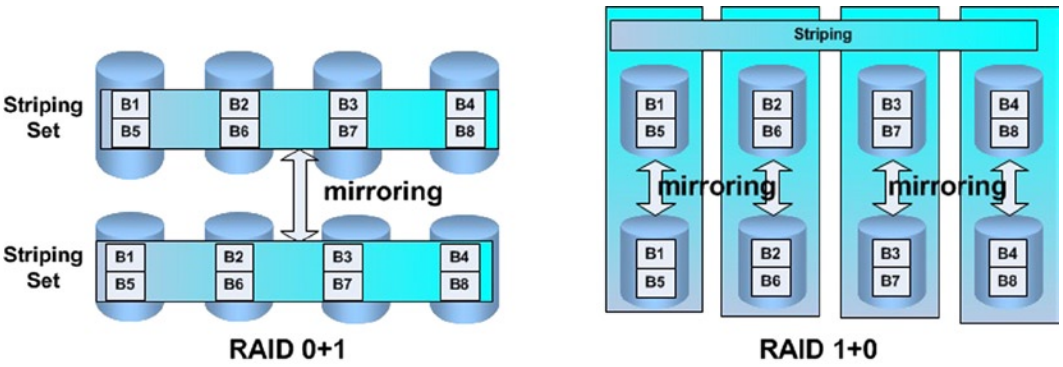


Figure 5-2. *RAID 0+1 vs. RAID 1+0*

The difference between RAID 1+0 and RAID 0+1 is that if one disk fails in RAID 0+1, the entire striping set fails. Then, if another disk in the second mirroring set fails, this array is lost. While in the RAID 1+0 configuration, as long as both disks in the same mirror set fail, the array is fine. Both these options reduce the disk capacity by half. This configuration is commonly used for OLTP-type database workload, as this option doesn't impose a heavy performance overhead for the write operation which is very frequent for OLTP type of workload.

- RAID 5 for striping blocks with parity on all the disks. To provide error protection, distributed parity blocks are kept on all the disks. In case of a disk failure, the parity blocks are used to reconstruct the errant sector. Although this option reduces the capacity of only one disk, it imposes a heavy performance overhead for write operations, as the parity block has to be recalculated every time. Considering the cost savings, RAID 5 may be an acceptable configuration option for a data warehouse–type database, which mostly does read operations and which needs a significant amount of disk capacity, but it is not a good choice for update-heavy databases engaged in OLTP workloads.
- RAID 6 for striping blocks with parity on all the disks. It is similar to RAID 5 except for having double distributed parity. It provides fault tolerance up to two failed drives.

A RAID array can be implemented as either hardware RAID or software RAID. Hardware RAID operates at the level of the RAID Controller; the OS does not know anything about it. When the data comes from the OS to the storage, the RAID controller card takes care of striping or mirroring. In software RAID, the OS takes the responsibility of striping the data and writing to the appropriate disk as needed. Software RAID is low cost as it doesn't require special hardware, but it costs the host CPU resource. Hardware RAID doesn't consume any host CPU, but there is an additional cost for the RAID controller card. The external storage used for shared storage in Oracle RAC Database usually uses SAN or something similar. The storage controller of these storage systems provides hardware RAID functionality and supports different kinds of disk RAID configuration such as RAID 0, 1, 1+0, and 5, etc. It is a part of the storage design and configuration to create a disk RAID configuration (called the RAID group in some storage product terminologies) based on a set of individual physical disks. Then, data volumes or logical unit numbers (LUNs) can be created on these RAID configurations or groups.

Figure 5-3 shows an example of the storage configuration. Table 5-1 lists the design of the RAID groups and the storage volumes associated with these RAID groups. In this example, there are three RAID 10 RAID groups—DataG1, DataG2, and DataG3—and two RAID 1 RAID groups—RedoG1 and RedoG2. There are eight storage volumes or LUNs: Data1-Data3, OCR1-OCR3, and Redo1-Redo2. These volumes are used as the storage LUNs for database files, OCT, voting disk files, and Redo logs, as indicated by their names. Figure 5-3 shows an example of the actual configuration of storage volumes on a storage Controller Management GUI tool called Dell PowerVault Modular Disk Storage Manager (MDSM). Although different storage vendors have different storage management tools, they provide similar functionality that allows storage administrators to create storage volumes based on physical disks with certain RAID configurations.

Table 5-1. *Storage RAID configuration design*

RAID Group Name	RAID Level	Number of Disks	Volumes
DataG1	10	4	Data1, OCR1
DataG2	10	4	Data2, OCR2
DataG3	10	4	Data3, OCR3
Redog1	1	2	Redo1
Redog2	1	2	Redo2

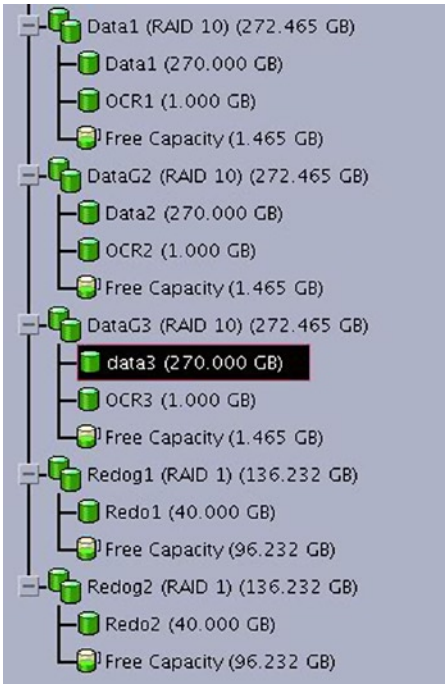


Figure 5-3. *An example of storage RAID configuration for RAC Database*

Storage Protocols

Apart from RAID configuration, it is a critical part of storage design to determine how the RAC node hosts share access to the storage and perform database I/O operations through the storage. This design includes the topology of physical links between the hosts and the storage and storage protocols that are used to transfer the data. Here we explore some widely used storage network protocols such as Small Computer System Interface (SCSI), Fibre Channel (FC), Internet Protocol (IP), and Network Area Storage (NAS). SCSI, FC, and IP protocols send block-based data, called the block base protocol, while NAS sends file-based data across the network, called the file-based protocol.

The SCSI protocol defines how host operating systems do I/O operations on disk drives. In the SCSI protocol, the data is sent in a chunk of bits called a "block" in parallel over a physical connection such as a copper SCSI cable. Every bit needs to arrive at the other end of cable at the same time. This limits the maximum distance between the host and the disk drives to under 25 meters. In order to transfer the data over a longer distance, the SCSI protocol usually works with other protocols such as FC and Internet SCSI (iSCSI).

FC is a SAN technology that carries data transfers between hosts and SAN storage at very high speeds. The protocol supports 1 Gbps, 2 Gbps, 4 Gbps, and 16 Gbps (most recently) and maintains very low latency. The FC protocol supports three different topologies: 1) connecting two devices in a point-to-point model; 2) An FC-arbitrated loop connecting up to 128 devices; 3) FC switched fabric model. FC switched fabric is the most common topology for Oracle RAC Database. The example in Figure 5-4 shows a configuration of two-node Oracle RAC connecting with Dell Compellent SC8000 FC SAN storage using two FC switches.

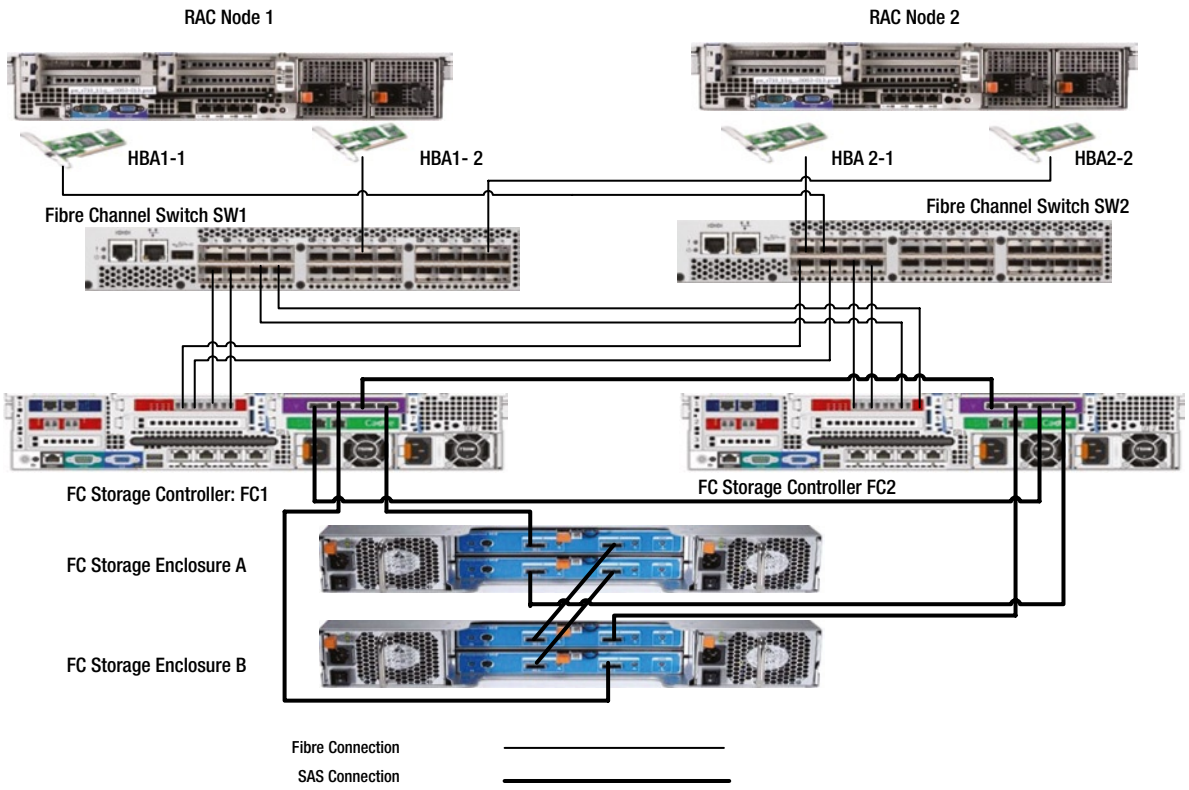


Figure 5-4. *FC switch fabric topology*

To provide highly available storage connections, each RAC host has two HBAs (Host Bus Adapter), each of which connects to an FC switch through a fiber optical cable. Each FC switch connects to both FC storage controllers. To increase the storage I/O bandwidth, each FC switch is connected to both FC controllers with two fiber cables. By using FC switches, the number of servers that can connect to an FC storage is not restricted. And the distance between hosts and storage can be up to 10 km. The components in the storage path such as HBAs, fiber optical cables, FC switches, and the storage controllers are very robust and capable of providing highly efficient and reliable data transfer between the server hosts and the storage. The FC storage controller can also connect to multiple storage enclosures. These storage enclusures are connected by daisy-chained SAS cables. The physical links between the storage controllers and the storage enclosures use SAS cables.

Figure 5-3 shows a configuration with two enclosures connected to two storage controllers. It is also possible to add more storage enclosures. Storage vendors usually specify the maximum number of storage enclosures that their controllers can support. Adding multiple storage enclosures allows you to add more disk spindles to the storage. For example, if one enclosure holds 24 disks, two enclosures can hold up to 48 disks. You also can put disks of different speeds in different enclosures. For example, to improve storage I/O performance, you can put SSDs in one enclosure and 15K rpm hard disks in another enclosure. Today, many SAN storage vendors provide some kind of storage tiering

technology that can direct application workloads to different levels of storage media to achieve the most suitable performance and cost characteristics. As well as the redundant connection paths, a disk RAID configuration such as RAID 10 or RAID 5 should be implemented to avoid any single point of failure at the disk level.

Using the configuration shown in Figure 5-4, let's examine how multiple I/O paths are formed from a RAC node to the storage servers. In an FC storage configuration, all the devices connected to the fibers such as HBAs and storage controller ports are given a 64-bit identifier called World Wide Number (WWN). For example, the HBA's WWN number in Linux can be found as follows:

```
 $ more /sys/class/fc_host/host8/port_name
0x210000e08b923bd5
```

On the switch layer, a zone is configured to connect an HBA's WWN with a storage controller port's WWN. As shown in Figure 5-4, these WWNs are as follows:

1. RAC host 1 has HBA1-1 and HBA1-2, which connect to FC switches SW1 and SW2, respectively.
2. RAC host 2 has HBA2-1 and HBA2-2, which connect to FC switches SW1 and SW2, respectively.
3. There are two FC controllers, FC1 and FC2, which are connected to FC switches SW1 and SW2, respectively.

The storage zoning process is to create multiple independent physical I/O paths from RAC node hosts to the storage through the FC switches to eliminate the single point of failure.

After zoning, each RAC host establishes multiple independent physical I/O paths to the SAN storage. For example, RAC host 1 has four paths:

- I/O Path1: HBA1-1, SW2, FC1
- I/O Path2: HBA1-1, SW2, FC2
- I/O Path3: HBA1-2, SW1, FC1
- I/O Path4: HBA1-2, SW1, FC2

These redundant I/O paths give a host multiple independent ways to reach a storage volume. The paths using different HBAs show up as different devices in the host (such as /dev/sda or /dev/sdc), even though these devices point to the same volume. These devices share one thing in common, in that they all have the same SCSI ID. In the next section, I will explain how to create a logical device that includes all the redundant I/O paths. Since this logical device is supported by multiple independent I/O paths, the storage access to this volume is protected from multiple-component failure up to a case when one HBA, one switch, and one controller all fail at the same time.

An FC SAN provides a highly reliable and high-performance storage solution for RAC Database. However, the cost and complexity of FC components make it hard to adopt for many small and medium businesses. However, the continuously improving speed of Ethernet and the low cost of its components has led to more adoption of the iSCSI storage protocol. iSCSI SAN storage extends the traditional SCSI storage protocol by sending SCSI commands over IP on Ethernet. This protocol can transfer data at a high speed for very long distances, especially by adding high-performance features such as high-speed NICs with TCP/IP Offload engines (TOE), and switches with low-latency ports. The new 10g GbE Ethernet allows iSCSI SAN storage to deliver even higher performance. Today, network bandwidths for both FC and iSCSI are improving. FC has moved to 1 Gbps, 2 Gbps, 4 Gbps, and even 16 Gbps, and iSCSI is also moving from 1 GbE to 10 GbE. Both FC and iSCSI storage are able to delive storage performance good enough to meet enterprise database needs.

As shown in Figure 5-5, iSCSI storage uses regular Ethernet to connect hosts and storage. For traditional 1GbE Ethernet, it can use regular Ethernet network cards, cables, and switches for data transfer between servers and storage. To design 10 GbE iSCSI storage SAN solution, you would have to make sure all the components support 10GbE Ethernet, including 10 GbE network adapter, high-speed cables, 10 GbE switches, and 10GbE storage controllers. And of course, this configuration will raise the cost of iSCSI storage deployment.

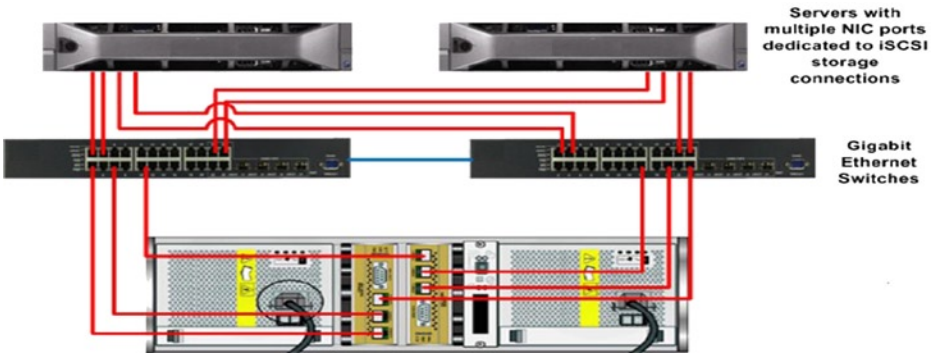


Figure 5-5. *iSCSI storage configuration for a two-node RAC*

Multipath Device Configuration

It is a recommended best practice to configure redundant I/O paths between Oracle RAC servers and storage, to ensure high availability of shared storage access, as shown in the figures in the last section. For example, from RAC host1, there are four redundant I/O paths to a same storage volume. These redundant I/O paths are represented by multiple SCSI devices on the RAC node that point to the same storage volume (also called logical unit, identified by the LUN). Since these SCSI devices are pointing to the same LUN, they have the same SCSI ID.

For example, /dev/sdc and /dev/sde represent two redundant I/O paths to the same LUN with the scsi ID: 36842b2b000742679000007a8500b2087. You can use the scsi_ID command to find the SCSI ID of a device:

```
[root@k2r720n1 $scsi_id  /dev/sdc
36842b2b000742679000007a8500b2087
```

The following Linux shell script finds the SCSI IDs of the devices:

```
[root@k2r720n1 ~]# for i in sdc sdd sde sdf ; do printf "%s %s\n" "$i" "$(scsi_id --page=0x83
--whitelisted --device=/dev/$i)"; done
sdc 36842b2b000742679000007a8500b2087
sdd 36842b2b000742679000007a5500b1cd9
sde 36842b2b000742679000007a8500b2087
sdf 36842b2b000742679000007a5500b1cd9
```

Many OS have their own multipathing software that can be used to create a pseudo-device to facilitate the sharing and balancing of I/O operations of LUNs across all available I/O paths. For example in Linux, a commonly used multipath device driver is the Linux native Device Mapper. To verify whether or not the rpm package is already installed, you can run this command:

```
[root@k2r720n1 yum.repos.d]# rpm -qa | grep device-mapper-multipath
device-mapper-multipath-libs-0.4.9-56.el6_3.1.x86_64
device-mapper-multipath-0.4.9-56.el6_3.1.x86_64
```

If the rpm package is not installed by default, you can install it from the yum repository:

```
$yum -y install device-mapper-multipath
```

Or install it manually:

```
rpm -ivh device-mapper-multipath-0.4.9-56.el6_3.1.x86_64.rpm
```

To configure the multipath driver to combine these I/O paths into a pseudo-device, you need to add related entries to the multipathing driver configuration file /etc/multipath.conf. This is an example of the /etc/multipath.conf file:

```
defaults {
        udev_dir                /dev
        polling_interval        5
        path_selector           "round-robin 0"
        path_grouping_policy    failover
        getuid_callout          "/lib/udev/scsi_id --whitelisted --device=/dev/%n"
        prio                    const
        path_checker            directio
        rr_min_io               1000
        rr_weight               uniform
        failback                manual
        no_path_retry           fail
        user_friendly_names     yes
}
...
multipaths {
        multipath {
                wwid            36842b2b000742679000007a8500b2087  #<---sdc and sde
                alias           ocrvoting
         }
         multipath {
                wwid            36842b2b000742679000007a5500b1cd9  #<--- sdd and sdf
                alias           data
         }
}
...
```

Start the multipath service:

```
# service multipathd start
Starting multipathd daemon:                                [  OK  ]
```

The device-mapper-multipath daemon creates a pseudo-device 'ocrvoting':

```
[oracle@k2r720n1 ~]$ ls -l /dev/mapper/
lrwxrwxrwx 1 root root      7 Oct 29 06:33 /dev/mapper/data -> ../dm-9
lrwxrwxrwx 1 root root      7 Oct 29 06:33 /dev/mapper/ocrvoting -> ../dm-8
This shows that the soft links such as  /dev/mapper/data pointing to block device /dev/dm-9
```

In addition to the benefit of combining multipath devices into a single pseudo-device, the multipath driver also creates a consistent device name. In Linux, by default the device name of storage LUN may not guarantee consistency every time the OS gets rebooted. For example, for the same LUN, /dev/sdc may be changed to /dev/sdd the next time the OS gets rebooted. This can create a serious problem if the RAC Database uses this volume to store the database files, as the RAC Database instance recognizes the volume by name. If the name changes, this volume cannot be found again. And in the Oracle RAC environment, we also need to make sure the same shared storage LUN has the

same consistent name on every node of the RAC. However, by default, a storage volume is not guaranteed to have the same name across all the RAC nodes. By mapping a LUN SCSI ID to the pseudo-device name and also setting 'user_friendly_names' to 'yes' in the multipath.conf file, the pseudo-device name will be kept consistent every time each of the RAC servers gets rebooted, and this name is guaranteed to be identical on every node of the RAC if we use the same multipath.conf on all the RAC nodes.

Set Ownership of the Devices

The next challenge in setting up devices for the Oracle RAC environment is to set proper ownership of these devices. When these devices were initially created, they were owned by the root user, thus:

```
$ ls -l /dev/mapper/*
lrwxrwxrwx 1 root root      7 Oct 29 17:16 ocrvoting -> ../dm-8
lrwxrwxrwx 1 root root      7 Oct 29 17:16 data -> ../dm-9
```

And block devices dm-8 and dm-9 are owned by root:

```
$ls -l /dev/dm-*
brw-rw---- 1 root disk 253,  8 Oct 29 06:33 /dev/dm-8
brw-rw---- 1 root disk 253,  9 Oct 29 06:33 /dev/dm-9
```

In the Oracle RAC Database, if Oracle ASM is selected to manage the shared storage volumes for OCR, voting disk files, and database files, the ASM instance needs to have write privileges on these devices. The solution is to change the ownership of these devices from root to the owner of the ASM instance, for example the 'grid' user.

There are two methods to change the proper ownerships in Linux:

- Use the Linux udev utility to create udev rules that change the ownership of the devices.
- Create ASM disks using ASMLib. The ASM disks will be given a new owner, which can be the owner of ASM instance like the "grid" user.

Of these two methods, the udev rule method is based on a Linux utility which is available for various versions of Linux such as Red Hat Enterprise Linux 5.x and 6.x, and Oracle Linux 5.x and 6.x. Oracle ASMLib has been generally available for both Red Hat Enterprise Linux 5.x and Oracle Linux 5.x. However, for Enterprise Linux 6.x, ASMLib was initially only available for Oracle Linux 6.x UEK kernel. ASMLib was not available for Oracle Linux 6 Red Hat–compatible kernel and Red Hat Exterprise Linux 6, until Spring 2013, when Oracle released the rpm packages for Oracle Linux 6 and Red Hat made the ASMLib kernel package for Red Hat Enterprise Linux 6 (beginning with 6.4). The dev rules method is the choice if you don't use ASMLib. Let's discuss how to set the udev rules in detail here. ASMLib will be discussed in the next section.

The following are the steps to create udev rules for the devices:

1. Get the SCSI ID (WWIDs) for all the devices that will be used for the RAC Database by using the SCSI script mentioned previously.
2. Create a udev rule file, for example /etc/udev/rules.d/99-oracle-asmdevices.rules

```
#----------------------- start udev rule contents ------------------------------------------------
----------#
KERNEL=="dm-*", PROGRAM="scsi_id --page=0x83 --whitelisted
--device=/dev/%k",RESULT=="36842b2b000742679000007a8500b2087", OWNER:="grid", GROUP:="asmadmin"
KERNEL=="dm-*", PROGRAM="scsi_id --page=0x83 --whitelisted
--device=/dev/%k",RESULT=="36842b2b000742679000007a5500b1cd9", OWNER:="grid", GROUP:="asmadmin"
#------------------------- end udev rule contents ------------------------------------------------
--------#
```

3. Run the start_udev command to apply the newly created rules:

```
 [root@k2r720n1 rules.d]# start_udev
Starting udev:                                             [  OK  ]
```

4. Check the new ownership and permission settings:

```
 # ls -l  /dev/dm-*
brw-rw---- 1 grid asmadmin 253,  8 Oct 29 12:08 /dev/dm-8
brw-rw---- 1 grid asmadmin 253,  9 Oct 29 12:08 /dev/dm-9
```

In summary, the provisioning of storage volumes for an Oracle RAC Database consists of these steps:

- Establish redundant paths from RAC hosts to the storage.
- Design and configure the proper storage volumes and assign these volumes to the RAC hosts so that these volumes are presented as the block devices on the RAC hosts.
- Configure multipathing on the block devices by mapping them to the multipath block device names.
- Set the proper ownerships and the access permissions on these multipath block device names.

ASM

In the previous section, I discussed storage design and how to present storage volumes as block devices to Oracle RAC Database hosts. In order to be able to store database files using these block devices, we need to establish database file systems on these block devices. For the Oracle RAC environment, this file system has to be a cluster file system that is shared by multiple RAC nodes. Existing volume managers like LVM in Linux and OS file systems like the ext3 file system in Linux are only for local systems; they are not designed for the cluster environment. Therefore, they cannot be used for shared storage in the Oracle RAC environment. Prior to Oracle 10g, the Oracle RAC Database was built on raw devices, which turned out to be very difficult to use and manage due to lack of flexibility. For example it required one raw device for every file, and the file could not grow or extend. For some Oracle E-Business suite databases which might need more than 255 files, it actually ran out of the 255 maximum number of available raw devices.

Oracle ASM was initially introduced in Oracle 10g as a volume manager and file system for Oracle data files. Oracle ASM supports both single-node Oracle Database and Oracle RAC Database. For Oracle RAC Database, Oracle ASM provides a cluster file system that allows all the RAC nodes to share. With ASM, database files can be created or grown and expanded as needed. Oracle ASM also provides high availability and performance features such as RAID configuration and I/O load balancing. Oracle ASM is Oracle's recommended storage solution for Oracle Database.

Oracle ASM has evolved since it was first introduced in Oracle 10gR1. For example, 10gR2 introduced the command-line interface ASMCMD, multiple database version support, and database storage consolidation with single instance and RAC. 11gR1 introduced Fast Mirror Resynchronization for ASM redundancy disk groups and ASM instance rolling upgrade and patching support, separate connect privilege SYSASM. Oracle Grid Infrastructure 11gR2 introduced quite a few new changes in ASM. The most significant ones include combining Oracle Clusterware and ASM into a single Grid Infrastructure stack, storing Oracle OCR and cluster voting disk files in ASM, and introducing the ASM cluster file system ACFS and ASM Dynamic Volume Manager ASVM. Oracle 12cR1 also brings many new features to Oracle ASM and Oracle ACFS. One of the significant new features is Oracle Flex ASM, which decouples the Oracle ASM instance from the database server and allows database instances to connect the remote ASM instances. Chapter 4 will give you more details about Oracle Flex ASM. Some other Oracle ASM enhancements introduced in Oracle 12c include increasing storage limits to support up to 511 storage diskgroups and the maximum Oracle ASM disk size to 32 Petabytes (PB); ASM shared password file in a disk group; ASM rebalance enhancements and ASM Disk Resync enhancements; ASMCMD extensions by including icmd command-line interface, a unified shell environment

that integrates all the required functionality to manage the Oracle Grid Infrastructure home, etc. The Oracle ASM software binary is installed in the Oracle ASM home directory, as the Oracle RAC Database software binary is installed in the Oracle Database home directory.

In Oracle 10g and 11g R1, although Oracle ASM and Oracle Database can be installed in the same home directory, Oracle highly recommends separating the Oracle ASM home from the Oracle Database home. This helps to improve flexibility in performing upgrade and patching operations. This is even more important if you run multiple versions of Oracle Database software on a cluster. In Oracle 11gR2 and Oracle 12cR1, the separation of the Oracle ASM home from the Oracle Database home has become mandatory as Oracle combines Oracle ASM and Oracle Clusterware into a new product called Grid Infrastructure. In this case, both Oracle ASM software binary and Oracle Clusterware binary are installed at the same time to the same Grid Infrastructure home.

ASM Instance

Oracle ASM instances provide the storage management for Oracle Database. An Oracle ASM instance has an architecture similar to an Oracle Database instance. It has SGA, the background processes, and the listener process. Since Oracle ASM needs to perform fewer tasks than the Oracle Database instance, it has a much smaller SGA size and usually has a minimal performance impact on a server. To provide storage access to Oracle Database instance(s), no matter how many versions of Oracle Database binaries and how many database instances are running, a RAC node can run a maximum of one Oracle ASM instance.

Prior to Oracle 11gR2, Oracle ASM ran above the Oracle cluster stack and provided the storage management for Oracle Database. As shown in Figure 5-6, the Oracle ASM instance depends on Oracle Clusterware to join the cluster environment and access the shared storage, and Oracle Database instances depend on the ASM instance to access the ASM diskgroups on the shared storage and depend on the Clusterware for cluster management. The entire RAC Database stack starts with Clusterware startup followed by ASM instance startup. The database instances will not start until both Clusterware and ASM instance have started up successfully.

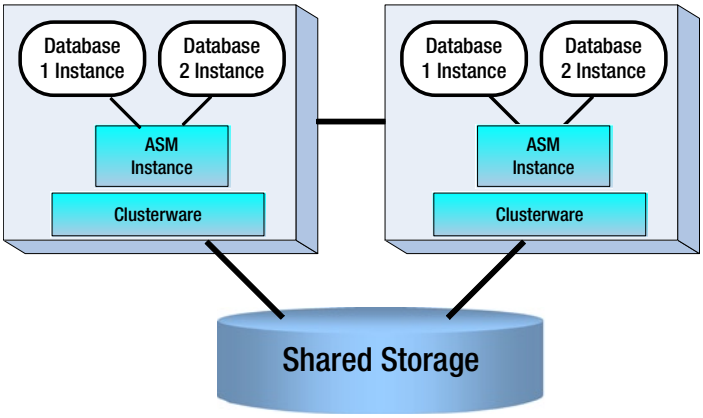


Figure 5-6. *Dependency prior to Oracle 11gR2*

Oracle 11gR2 introduced Grid Infrastructure, which combined Oracle ASM and Oracle Cluster into a single product that is installed and configured during the Grid Infrastructure installation. ASM becomes a part of the CRS of the Clusterware. A single Grid Infrastructure startup command starts up both Oracle Clusterware and the ASM instance. Then the database instances can be started. This dependency also applies to Oracle 12cR1. Prior to Oracle 12c, it is required to run an Oracle ASM instance on every RAC Database node. This requirement has been removed by introducing the Oracle Flex ASM option in Oracle 12cR1. By enabling this Flex Oracle ASM option in Oracle 12c, Oracle ASM instances may run separate nodes from Oracle Database 12c instances.

Some RAC nodes can run Oracle Database instances without an Oracle ASM instance. For the RAC nodes where there is no local ASM instance or the local ASM instance fails, the database instance on the RAC nodes can remotely connect to an Oracle ASM instance on another RAC node. Therefore in Oracle 12c, depending on whether or not the Oracle Flex ASM option is enabled, Oracle ASM can run in two distinct modes: the Standard ASM and the Flex ASM. The standard ASM works similarly to Oracle 11gR2 ASM, with every RAC node running an Oracle ASM instance and all the Oracle Database instances on the cluster connecting to the local ASM instances. Figure 5-7 shows the dependency in Oracle 11gR2 and Oracle 12cR1 Standard ASM without the Flex ASM being enabled.

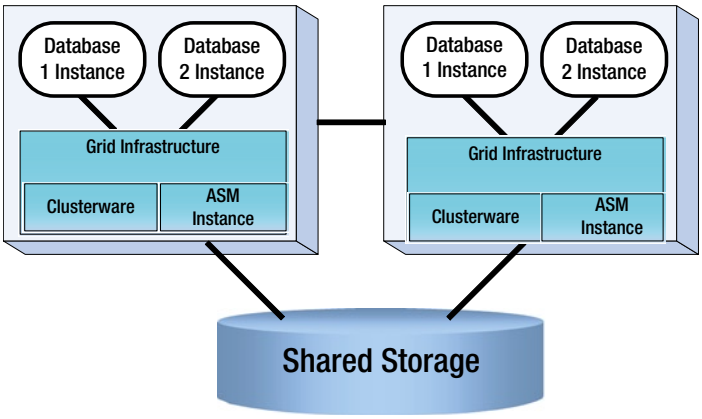


Figure 5-7. *Dependency 11gR2 ASM and 12cR1 standard ASM*

In this architecture, a single ASM instance per RAC node provides a clustered pool of storage to all the database instances running on that node; the ASM instance has access to the shared storage volumes and presents the ASM diskgroup as the file system for all the database instance(s) running on the RAC node.

In Oracle 12cR1, by enabling the Flex ASM, a small set of cluster nodes run Oracle ASM instances to provide storage access to a large number of cluster nodes. Figure 5-8 shows an example of the Flex ASM configuration. By default, three RAC nodes (nodes 1, 2, and 3) run Oracle ASM instances, and all the database instances, including ones on node 4 and 5, remotely run these ASM instances. Oracle Flex ASM often works with Oracle Flex Cluster. The nodes that have a direct connection to the shared storage are called the Hub nodes (such as nodes 1, 2, 3, and 4 in Figure 5-8), and the nodes (such as node 5 in Figure 5-8) that don't have a direct connection to the shared storage are Leaf nodes. Usually one or more Leaf nodes are connected to a Hub node to access shared storage. In Figure 5-8, Leaf node 5 accesses shared storage through Hub node 3. Chapter 4 discusses the details of Oracle Flex ASM and Oracle Flex Clusters.

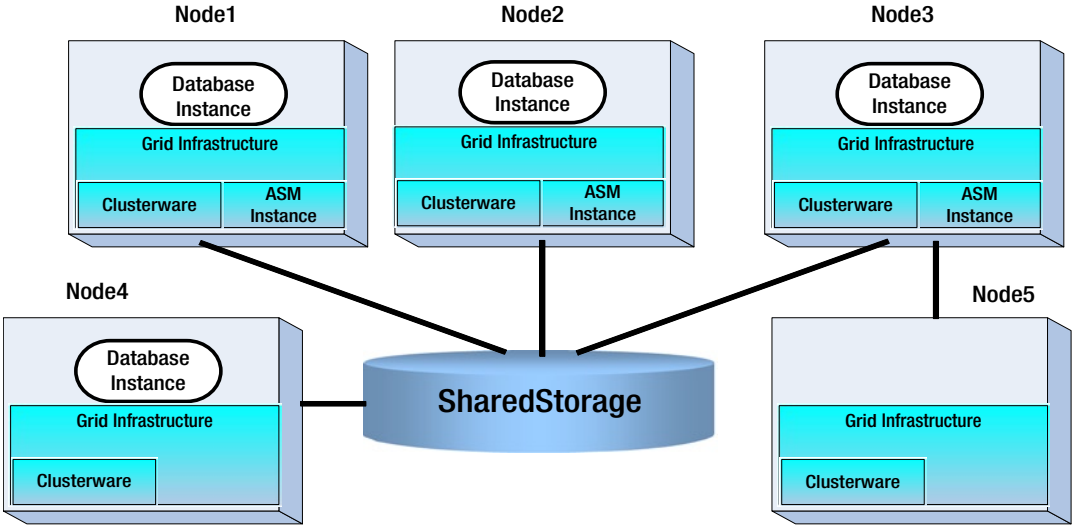


Figure 5-8. *Dependency on Oracle 12c Flex ASM*

Creation and Ownership of ASM Instance

However, the relationship between the ASM instance and Oracle Clusterware has been changed from Oracle 11g R1 to Oracle 11gR2/12cR1. The ASM instance depends on Oracle Clusterware for the basic cluster infrastructure and the communication between the cluster nodes. The 11gR1 Oracle Clusterware doesn't rely on the ASM instance to access the OCR and voting disk files, as these two important components for the Clusterware metadata are directly stored in block devices. In 11gR1, the ASM instance is created with the DBCA tool after the Oracle Clusterware installation and before database creation. However in 11gR2 and 12cR1, if you chose to store OCR and voting disks in ASM, Oracle Clusterware depends on the ASM instance to access the OCR and voting disk. This leads to mutual dependency between Oracle Clusterware and Oracle ASM. The solution to this dependency is that these two products are combined together into one product, Grid Infrastructure, and they are installed together into a single Grid Infrastructure home by the Grid Infrastructure installer. The installation process also creates the ASM instance, creates the diskgroup for storing OCR and voting disk, and finishes the Oracle Clusterware configuration on all the RAC nodes in the end.

In a RAC Database environment, the default ASM instance SID is +ASM node_number, such as +ASM1 for node 1 and +ASM2 for node 2, etc. Like an Oracle Database instance, an ASM instance has a set of background processes. For example, the following is a list of processes of an Oracle 12cR1 ASM instance:

```
$ps -ef | grep -v grep | grep asm_
grid      4335     1  0 Feb19 ?        00:06:51 asm_pmon_+ASM1
grid      4337     1  0 Feb19 ?        00:05:08 asm_psp0_+ASM1
grid      4339     1  2 Feb19 ?        07:47:18 asm_vktm_+ASM1
grid      4343     1  0 Feb19 ?        00:01:48 asm_gen0_+ASM1
grid      4345     1  0 Feb19 ?        00:00:43 asm_mman_+ASM1
grid      4349     1  0 Feb19 ?        00:18:12 asm_diag_+ASM1
grid      4351     1  0 Feb19 ?        00:03:18 asm_ping_+ASM1
grid      4353     1  0 Feb19 ?        01:12:41 asm_dia0_+ASM1
grid      4355     1  0 Feb19 ?        00:48:21 asm_lmon_+ASM1
grid      4357     1  0 Feb19 ?        00:27:40 asm_lmd0_+ASM1
grid      4359     1  0 Feb19 ?        01:00:51 asm_lms0_+ASM1
grid      4363     1  0 Feb19 ?        00:13:18 asm_lmhb_+ASM1
```

```
grid      4365     1  0 Feb19 ?        00:01:06 asm_lck1_+ASM1
grid      4367     1  0 Feb19 ?        00:00:14 asm_gcr0_+ASM1
grid      4369     1  0 Feb19 ?        00:00:44 asm_dbw0_+ASM1
grid      4371     1  0 Feb19 ?        00:00:54 asm_lgwr_+ASM1
grid      4373     1  0 Feb19 ?        00:01:52 asm_ckpt_+ASM1
grid      4375     1  0 Feb19 ?        00:00:38 asm_smon_+ASM1
grid      4377     1  0 Feb19 ?        00:00:45 asm_lreg_+ASM1
grid      4379     1  0 Feb19 ?        00:06:21 asm_rbal_+ASM1
grid      4381     1  0 Feb19 ?        00:04:37 asm_gmon_+ASM1
grid      4383     1  0 Feb19 ?        00:02:14 asm_mmon_+ASM1
grid      4385     1  0 Feb19 ?        00:03:35 asm_mmnl_+ASM1
grid      4387     1  0 Feb19 ?        00:03:49 asm_lck0_+ASM1
grid      4429     1  0 Feb19 ?        00:00:31 asm_asmb_+ASM1
grid     22224     1  0 Mar03 ?        00:00:16 asm_scrb_+ASM1
```

In this case, the "grid" user is the owner of the ASM instance as well as the owner of Grid Infrastructure. The "grid" user belongs to three operating system groups: asmdba, asmoper, and asmadmin.

```
$id grid
uid=54322(grid) gid=54321(oinstall) groups=54321(oinstall), 54325(asmadmin), 54326(asmdba),
54327(asmoper)
```

During Grid Infrastructure installation, these three OS groups, namely, asmdba, asmoper, and asmasm, are granted the Oracle ASM DBA (SYSDBA), Oracle ASM Operator (SYSOPER), and Oracle ASM Administrator (SYSASM) privileges, as shown in Figure 5-9.

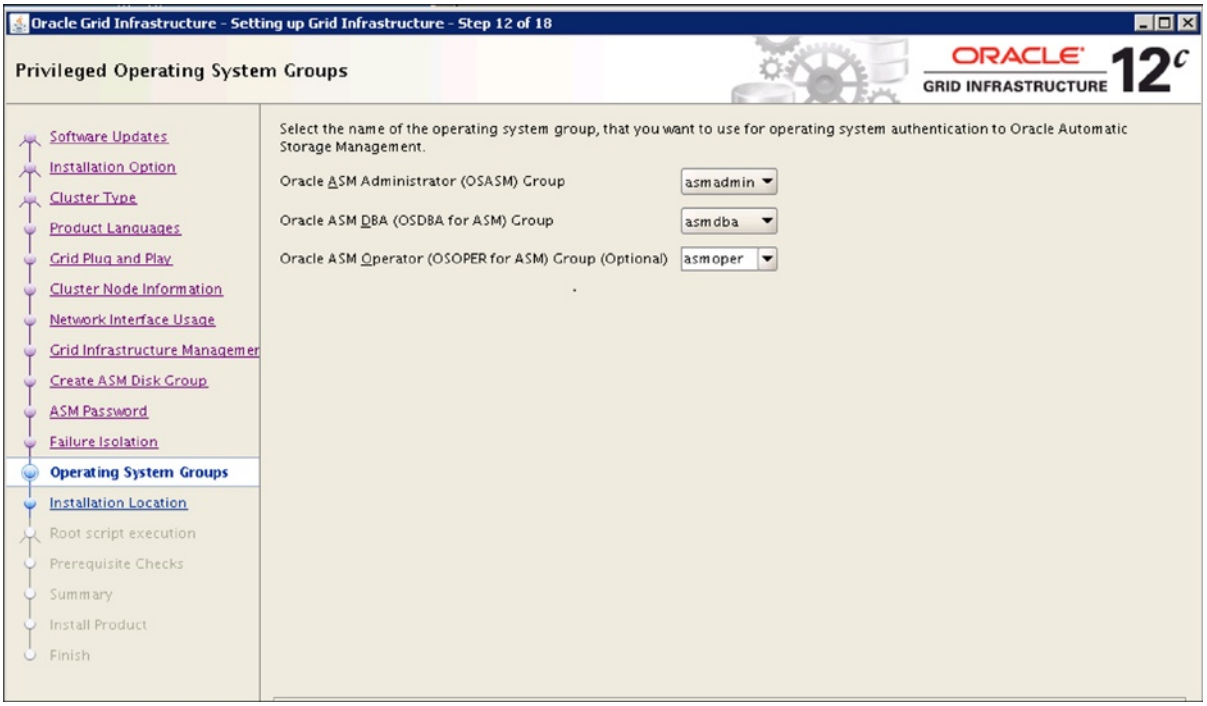


Figure 5-9. *Three OS groups for ASM management*

The SYSDBA privilege provides access to the data stored in ASM diskgroups; the SYSOPER privilege allows performing instance operations such as startup, shutdown, mount, dismount, and checking diskgroup; and the SYSASM privilege allows full administration for the Oracle ASM instance, such as creating or altering diskgroup.

Through these three OS groups, the "grid" is granted all three system privileges for the Oracle ASM instance: SYSDBA, SYSOPER, and SYSASM. This user is the owner of Grid Infrastructure, which is used to install and manage the Grid Infrastructure including the ASM instance. In your environment, you also can create additional users with one or two of these system privileges to perform specific operations such as monitoring ASM diskgroups.

Like a database instance, you can manage an ASM instance using the sqlplus command. For example, in Linux or Unix, you can set the following environment in the bash profile of the grid user:

```
export ORACLE_SID=+ASM1
export ORACLE_HOME=/u01/app/12.1.0/grid
```

Through the grid user, you can log in to the ASM instance with the sysasm privilege and perform ASM administration tasks:

```
$ sqlplus / as sysasm

SQL*Plus: Release 12.1.0.1.0 Production on Tue Mar 5 13:22:37 2013

Copyright (c) 1982, 2013, Oracle.  All rights reserved.

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Real Application Clusters and Automatic Storage Management options

SQL> show parameter spfile

NAME      TYPE  VALUE
------------------------------------ ----------- ------------------------------
spfile   string  +DATA1/knewrac/ASMPARAMETERFILE/registry.253.807834851
```

Notice that in the preceding example, the spfile of the ASM instance is stored in a diskgroup "DATA." Another user is "oracle," which is granted SYSDBA and SYSOPER privileges through asmdba and asmoper groups.

```
$ id oracle
uid=54321(oracle) gid=54321(oinstall) groups=54321(oinstall), 54322(dba), 54326(asmdba),
54327(asmoper)
```

Without being granted the SYSASM privilege, the "oracle" user cannot use the sysasm privilege to log in to the Oracle ASM instance to do operations such as creating and altering ASM diskgroup. SYSASM enables the separation of SYSDBA database administration from Oracle ASM storage administration. In this case, we log in as the grid user to do ASM administration and log in as the Oracle user to do database administration.

Listener for ASM Instance

By default, the listener runs from the Grid Infrastructure home and listens for both ASM instance and database instances on that RAC node. You don't need to run another listener from Oracle RAC home.

Here is an example of the Oracle 12c listener:

```
$ps -ef | grep -v grep | grep LISTENER
grid      4799     1  0 Feb19 ?        00:02:00 /u01/app/12.1.0/grid/bin/tnslsnr LISTENER_SCAN3
-no_crs_notify -inherit
grid      8724     1  0 Feb19 ?        00:01:11 /u01/app/12.1.0/grid/bin/tnslsnr LISTENER
-no_crs_notify -inherit

$lsnrctl status
LSNRCTL for Linux: Version 12.1.0.1.0 - Production on 05-MAR-2013 13:39:22

Copyright (c) 1991, 2013, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=LISTENER)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Linux: Version 12.1.0.1.0 - Production
Start Date                19-FEB-2013 22:47:56
Uptime                    13 days 14 hr. 51 min. 29 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /u01/app/12.1.0/grid/network/admin/listener.ora
Listener Log File         /u01/app/grid/diag/tnslsnr/knewracn1/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=LISTENER)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=172.16.9.41)(PORT=1521)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=172.16.150.37)(PORT=1521)))
Services Summary...
Service "+APX" has 1 instance(s).
  Instance "+APX1", status READY, has 1 handler(s) for this service...
Service "+ASM" has 1 instance(s).
  Instance "+ASM1", status READY, has 2 handler(s) for this service...
Service "-MGMTDBXDB" has 1 instance(s).
  Instance "-MGMTDB", status READY, has 1 handler(s) for this service...
Service "_mgmtdb" has 1 instance(s).
  Instance "-MGMTDB", status READY, has 2 handler(s) for this service...
Service "knewdb.kcloud.dblab.com" has 1 instance(s).
  Instance "knewdb_3", status READY, has 1 handler(s) for this service...
Service "knewdbXDB.kcloud.dblab.com" has 1 instance(s).
  Instance "knewdb_3", status READY, has 1 handler(s) for this service...
Service "knewpdb1.kcloud.dblab.com" has 1 instance(s).
  Instance "knewdb_3", status READY, has 1 handler(s) for this service...
Service "knewpdb2.kcloud.dblab.com" has 1 instance(s).
  Instance "knewdb_3", status READY, has 1 handler(s) for this service...
The command completed successfully
```

In this example, the listener listens for the ASM instance plus three database instances: APX1, -MGMTDB, and knewdb_3. The APX1 is the Oracle Enterprise Manager Database Express database instance,–MGMTDB is the Grid Infrastructure Management repository database instance, and knewdb_3 is the user database instance.

Startup and Shutdown of ASM Instance

Similar to an Oracle Database instance, an ASM instance can be shut down or started up using the startup or shutdown command after you log in to ASM instance using SQLPLUS or using the command srvctl stop –n <nodename>.

However if the OCR and voting disk files of the Clusterware are stored in Oracle ASM, shutting down ASM will also bring down Oracle CRS of the Clusterware. Here are some of the error messages shown in the alert.log file of Clusterware after executing the "shutdown" command in the ASM instance:

```
crsd(5491)]CRS-1013:The OCR location in an ASM disk group is inaccessible. Details in /u01/
app/12.1.0/grid/log/k2r720n1/crsd/crsd.log.2013-02-01 17:17:29.604
...
[crsd(5491)]CRS-2765:Resource 'ora.VOCR.dg' has failed on server 'k2r720n1'.2013-02-01 17:17:29.924
[/u01/app/12.1.0/grid/bin/oraagent.bin(5600)]CRS-5822:Agent '/u01/app/12.1.0/grid/bin/oraagent_grid'
disconnected from server. ...
...
[crsd(8205)]CRS-0804:Cluster Ready Service aborted due to Oracle Cluster Registry error [PROC-26:
Error while accessing the physical storage
...
[ohasd(3813)]CRS-2765:Resource 'ora.crsd' has failed on server 'k2r720n1'.
2013-02-01 17:17:50.104
[ohasd(3813)]CRS-2771:Maximum restart attempts reached for resource 'ora.crsd'; will not restart
In Cluster Registry process log: crsd.log:
...
2013-02-01 17:17:39.253: [  OCRASM][2793891616]ASM Error Stack : ORA-15077: could not locate ASM
instance serving a required diskgroup
2013-02-01 17:17:39.254: [  OCRASM][2793891616]proprasmo: The ASM instance is down
2013-02-01 17:17:39.254: [  OCRRAW][2793891616]proprioo: Failed to open [+VOCR]. Returned
proprasmo() with [26]. Marking location as UNAVAILABLE.
2013-02-01 17:17:39.254: [  OCRRAW][2793891616]proprioo: No OCR/OLR devices are usable
...
2013-02-01 17:17:39.258: [  CRSOCR][2793891616] OCR context init failure.  Error: PROC-26: Error
while accessing the physical storage
ORA-15077: could not locate ASM instance serving a required diskgroup
2013-02-01 17:17:39.258: [ CRSMAIN][2793891616] Created alert : (:CRSD00111:) :  Could not init OCR,
error: PROC-26: Error while accessing the physical storage
ORA-15077: could not locate ASM instance serving a required diskgroup
2013-02-01 17:17:39.258: [    CRSD][2793891616][PANIC] CRSD exiting: Could not init OCR, code: 26
2013-02-01 17:17:39.258: [    CRSD][2793891616] Done.
```

The crsctl check also showed that the CRS was offline.

```
$crsctl check crs
CRS-4638: Oracle High Availability Services is online
CRS-4535: Cannot communicate with Cluster Ready Services
CRS-4529: Cluster Synchronization Services is online
CRS-4533: Event Manager is online
At this point, restarting ASM instance still failed to mount the diskgroup for the OCR/voting disk,
which prevented OCR and voting disk files being accessible from the Clusterware. Without being able
to access OCR and voting disk files, we were not able to start the CRS service.
SQL> startup
ASM instance started
```

```
Total System Global Area  409194496 bytes
Fixed Size                  2228864 bytes
Variable Size             381799808 bytes
ASM Cache                  25165824 bytes
ORA-15032: not all alterations performed
ORA-15017: diskgroup "VOCR" cannot be mounted
ORA-15063: ASM discovered an insufficient number of disks for diskgroup "VOCR"
```

The crsctl check also showed that the CRS was offline.

```
$crsctl check crs
CRS-4638: Oracle High Availability Services is online
CRS-4535: Cannot communicate with Cluster Ready Services
CRS-4529: Cluster Synchronization Services is online
CRS-4533: Event Manager is online
```

ASM disk groups are registered as resources with Grid Infrastructure. Each resource with Grid Infrastructure has its resource dependencies that define the relations among resources. The dependency determines the startup order or shutdown order of the resources. For example, the following shows the dependency associated with VOCR diskgroup:

```
$  crsctl stat res ora.VOCR.dg -p | grep DEPENDENCIES
START_DEPENDENCIES=pullup:always(ora.asm) hard(ora.asm)
STOP_DEPENDENCIES=hard(intermediate:ora.asm)
```

This shows that the VOCR diskgroup has the hard STOP dependency on ASM instance, which defines if ASM instance stops the VOCR diskgroup also stops. It also shows that the VOCR diskgroup has both HARD and pull-up START dependencies on ASM instance, which defines that the VOCR diskgroup can only be started when ASM instance is running (HARD dependency) and the VOCR diskgroup must be automatically started whenever the ASM instance starts. For more details about resource dependency in 12cR1 Clusterware, refer to Chapter 8.

Because of this dependency of the VOCR diskgroup on the ASM instance, whenever ASM instance stops, the VOCR diskgroup goes offline. As a result, the OCR stored in the VOCR diskgroup is offline, which caused OCR Service to stop and bring down Oracle Clusterware. That was exactly what we saw in the alert.log and crsd.log files.

This case shows that if we have OCR/voting disk stored in ASM, the best way to shut down ASM is to shut down the entire Grid Infrastructure cleanly when all the components of the Grid Infrastructure are up:

```
$crsctl stop crs
```

We should avoid just shutting down the ASM instance as Clusterware and ASM are deeply connected and integrated. They should be started up and shut down together.

ASM Storage Structure

In this section, I discuss how Oracle ASM provides a volume manager and a cluster file system for both Oracle Clusterware and RAC Databases. I will start with the ASM storage components such as ASM disk, ASMLib, and ASM diskgroups, and then examine the ASM file system structure and how to manage ASM diskgroup and ASM files and directories.

ASM Disks

The ASM disk is the basic element of the Oracle ASM storage structure. It forms the Oracle ASM diskgroup. An ASM disk is usually based on a piece of a storage device such as a physical disk, or a storage volume like the LUN of a storage array. Depending on how the storage is presented to the RAC node, an ASM disk can be built on a block device or a partition of block device, network-attached files, or a pseudo-device.

As stated in the preceding "Multipath Device Configuration" section, in an Oracle RAC environment that has multiple I/O paths to the external shared SAN storage, the same LUN in the SAN storage is presented as multiple devices on a RAC node, for example /dev/sdc and /dev/sde. These devices point to the same storage volume and have the same SCSI ID. We should not take these devices as different ASM disks; otherwise, Oracle ASM will find these ASM disks for the same storage volume and report an error. We should also not use just one device name, such as /dev/sdc as for an ASM disk, because the ASM disk based on this single I/O path device will not take advantage of the multiple I/O paths to this storage volume. A highly recommended method is to use the multipath pseudo-device of this storage volume for the ASM disk. Although Oracle ASM itself doesn't provide a way to implement multipathing, Oracle ASM can work with the OS multipath utility by using the multipath pseudo-device for the ASM disk. This method ensures that the Oracle ASM disk can benefit from load balancing as well as high availability against the single point of failure on the I/O paths from the RAC node to the storage array.

In order for an ASM instance to use a storage volume for the ASM disk, the ASM instance needs to discover the storage volume. To help an ASM discover a storage volume, a proper value may need to be set for the ASM instance initialization parameter ASM_DISKSTRING with a pattern to match the device name that represents the storage volume. For example, to discover a storage volume that is represented by a Linux multipath pseudo-device with a name pattern like /dev/mapper/, the ASM instance initialization parameter can be set as ASM_DISKSTRING =/dev/mapper/*.

Another requirement is ownership of the storage devices. For example, in the Linux environment, all the devices are owned by the root user by default. In the previous section entitled "Set Ownership of the Devices," I discussed the method that uses the Linux udev rule utility to change the owner of the devices from root to the owner of the ASM instance such as the grid user. The following will discuss how to use ASMLib to create ASM disks with the proper ownership setting.

ASMLib: What and Why

For Linux platforms, especially Oracle Linux and Red Hat Enterprise Linux, Oracle introduced a support library called Oracle ASMLib that provides an alternative interface to access disks and gives the Oracle Database more efficient and capable access to the disk groups. Notice that since ASMLib is an optional tool, use of ASMLib is not required for you to use ASM for your Oracle Database. ASMLib provides a simplified method to meet the two requirements for preparing storage devices for ASM disks:

- Ensure the name consistency of a storage volume across all the RAC nodes
- Set the proper ownership and permissions on the disk devices that are used for ASM disks

ASMLib can achieve both requirements. If you use ASMLib in your RAC environment, you don't have to set the udev rules. Table 5-2. compares these three methods.

Table 5-2. *Comparison of Multipathing, udev Rules, and ASMLib*

	Multipathing Software	udev Rules	ASMLib
Device Name Consistency	Yes	No	Yes
Device Ownership Setting	No*	Yes	Yes

** As for Red Hat Enterprise Linux 6.x, the uid and gid parameters in multipath.conf file that were previously used to set the ownership of devices have been deprecated. The ownership of the device is set by means of udev rules.*

In addition to the two benefits mentioned previously, with ASMLib, it is possible to add a data protection feature in the Linux kernel that will allow the checksum operation of a data block. This checksum will be validated at the firmware level of the storage adapter before the data is sent to the storage network, where it is validated again at the other end of the storage network before writing the data block to the physical disk. Wim Coekaerts's ASMLib blog at https://blogs.oracle.com/wim/entry/asmLib explains this new feature, which has its first implemention through a joint effort by EMC, Emulex, and Oracle. For the details of this solution and its implementation, refer to the article "How to Prevent Silent Data Corruption" at this link: www.oracle.com/technetwork/articles/servers-storage-dev/silent-data-corruption-1911480.html.

Download and Install ASMLib Packages

ASMLib configuration starts with ASMLib package installation. All ASMLib installations require three rpms packages installed on the host:

- Support package: oracleasm-support
- Tool library: oracleasmlib
- Kernel driver: oracleasm

For Oracle Linux 5.x and Red Hat Enterprise Linux 5.x, you can download the rpms of these packages from www.oracle.com/technetwork/server-storage/linux/downloads/rhel5-084877.html and install them with the rpm tool. For example, in Oracle Linux 5.8:

```
#rpm -ivh oracleasm-support-2.1.7-1.el5.x86_64.rpm
#rpm -ivh oracleasm-2.6.18-274.el5-2.0.5-1.el5.x86_64.rpm
#rpm -ivh oracleasmlib-2.0.4-1.el5.x86_64.rpm
```

The stories for Oracle Linux 6.x and Red Hat Enterprise Linux 6.x are different. Oracle ships Oracle Enterprise Linux 6.x with two sets of kernel: Unbreakable Enterprise Kernel (UEK kernel), and Red Hat Compatible Kernel. You can boot up your operating system with either of these kernels. For either of these two kernels, you need to download the oracleasm-support from the Oracle Unbreakable Linux Network (ULN) (https://linux.oracle.com/) if you have an active support subscription or from the Oracle public Yum repository (http://public-yum.oracle.com). You also need to download oracleasmlib from the Oracle ASMLib 2.0 web site: http://www.oracle.com/technetwork/server-storage/linux/asmlib/ol6-1709075.html.

However, for the oracleasm kernel driver, you will need to treat these two kernels differently. Since the oracleasm kernel driver is already built into the UEK kernel, you don't need to load the oracleasm kernel driver if you boot up your OS using the UEK kernel. If you use Oracle Linux 6.x Red Hat Campatible kernel, you need to download the Oracleasm kernel driver kmod-oracleasm-2.0.6.rh1-2.el6.x86_64.rpm manually from ULN and install it:

```
rpm -ivh kmod-oracleasm-2.0.6.rh1-2.el6.x86_64.rpm
```

or you can install it from Oracle public yum http://public-yum.oracle.com using the yum tool:

```
# yum install kmod-oracleasm
```

This kernel driver is not version-specific, and you don't need to upgrade the driver when the kernel is upgraded. However, in order to install this rpm package, you need to run Oracle Linux Server release 6.4 and later with kernel version > 2.6.32-358.el6. Otherwise, you will get an error message similar to this one, where the OS version is Oracle Linux Server release 6.2:

```
[root]# rpm -ivh kmod-oracleasm-2.0.6.rh1-2.el6.x86_64.rpm
error: Failed dependencies:
        kernel >= 2.6.32-358.el6 is needed by kmod-oracleasm-2.0.6.rh1-2.el6.x86_64
        kernel(kmem_cache_alloc_trace) = 0x2044fa9e is needed by kmod-oracleasm-2.0.6.rh1-2.el6.x86_64
```

Starting with Red Hat Enterprise Linux Server 6.4, you can install ASMLib. The kernel driver package 'kmod-oracleasm' for Red Hat Enterprise Linux Server 6.4 now is available in the Red Hat Enterprise Linux 6 Supplementary RHN channel. You also need to download and install 'oracleasmlib' and 'oracleasm-support' packages, as they are required for the ASMLib kernel package 'kmod-oracleasm.' These two packages are maintained by Oracle and can be downloaded from Oracle Technology Network at www.oracle.com/technetwork/server-storage/linux/asmlib/rhel6-1940776.html.

For other details about Oracle ASMLib for Red Hat Enterprise Linux, refer to Oracle support note "Oracle ASMLib Software Update Policy for Red Hat Enterprise Linux Supported by Red Hat [ID 1089399.1]." For ASMLib support in other versions of Linux, you can refer to the ASMLib Release Notes at www.oracle.com/technetwork/server-storage/linux/release-notes-092521.html.

Configure ASMLib and Create ASM Disks

After you install the three ASMLib packags (two ASMLib packages for UEK kernel), you need to configure ASM library drivers with the oracleasm configure command, as shown in the following example:

```
 # oracleasm configure -i
Configuring the Oracle ASM library driver.
This will configure the on-boot properties of the Oracle ASM library
driver. The following questions will determine whether the driver is
loaded on boot and what permissions it will have. The current values
will be shown in brackets ('[]'). Hitting <ENTER> without typing an
answer will keep that current value. Ctrl-C will abort.
Default user to own the driver interface []: grid
Default group to own the driver interface []: asmadmin
Start Oracle ASM library driver on boot (y/n) [n]: y
Scan for Oracle ASM disks on boot (y/n) [y]: y
Writing Oracle ASM library driver configuration: done
```

This configuration will ensure that all the ASM disks are owned by grid user and group asmadmin. Then, you can create the ASM disks on one RAC node:

```
# service oracleasm createdisk  OCR1 /dev/mapper/ocrvoting1
# service oracleasm createdisk  OCR2 /dev/mapper/ocrvoting2
# service oracleasm createdisk  OCR3 /dev/mapper/ocrvoting3
# service oracleasm createdisk  DATA1 /dev/mapper/data1
# service oracleasm createdisk  DATA2 /dev/mapper/data2
# service oracleasm createdisk  FRA /dev/mapper/fra
```

Scan ASM disks on all other RAC nodes:

```
# service oracleasm scandisks
Scanning the system for Oracle ASMLib disks:               [  OK  ]
# service oracleasm listdisks
DATA1
DATA2
FRA
OCR1
OCR2
OCR3
```

With Oracle ASMLib, Oracle ASM disks are named with the prefix 'ORCL:', such as 'ORCL:OCR1' and 'ORCL:DATA.' The Oracle ASM instance is able to discover these ASM disks with the default ASM_DISKSTRING setting, and you can see the ASM disks in /dev/oracleasm/disks. All of these ASM disks are owned by the grid user.

```
# ls -l /dev/oracleasm/disks
total 0
brw-rw---- 1 grid asmadmin    8,       81    OCT 20, 10:35    DATA1
brw-rw---- 1 grid asmadmin    8,       49    OCT 20, 10:35    DATA2
brw-rw---- 1 grid asmadmin    8,      129    OCT 20, 10:35    DATA3
brw-rw---- 1 grid asmadmin    8,      130    OCT 20, 10:35    DATA4
brw-rw---- 1 grid asmadmin    8,       65    OCT 20, 10:35    OCR1
brw-rw---- 1 grid asmadmin    8,       97    OCT 20, 10:35    OCR2
brw-rw---- 1 grid asmadmin    8,       98    OCT 20, 10:35    OCR3
brw-rw---- 1 grid asmadmin    8,       98    OCT 20, 10:35    FRA
```

Notice that all the Oracle ASMLib commands require root privilege to execute.

ASM Diskgroup

Once an ASM disk is discovered, it can be used to create ASM diskgroups that are used to store the file systems for Clusterware and the database files and ACFS Oracle home. Every Oracle ASM diskgroup is divided into allocation units (AU), the sizes of which are determined by the AU_SIZE disk group attribute. The AU_SIZE can be 1, 2, 4, 8, 16, 32, or 64 MB. Files that are stored in an ASM diskgroup are separated into stripes and evenly spread across all the disks in the diskgroup. This striping aims to balance loads across all the disks in the disk group and reduce I/O latency. Every ASM disk that participates in striping should have the same disk capacity and performance characteristics. There are two types of striping: coarse striping and fine striping. The coarse striping size depends on the size of the AU; it is used for most files, including database files, backup sets, etc. The fine striping is used for control files, online redo logs, and flash back logs. The stripe size is 128 KB.

The failure group concept was introduced to define a subset of disks in a diskgroup that could fail at the same time; for example, disks in the same failure group could be linked to the same storage controller. If we mirror the storage in an ASM diskgroup, we want to make sure to put the mirroring copies on the disks in different failure groups to avoid losing all the mirroring copies at the same time.

ASM provides three types of redundancy for ASM diskgroup:

1. *External Redundancy*: Oracle ASM does not provide mirroring redundancy and depends on the system to provide the RAID. The ASM files are striped across all the disks of the diskgroup. In this setting, the ASM diskgroup cannot tolerate the failure of any disk in the diskgroup. A failure of one or more disks in the diskgroup leads to dismount of the diskgroup from the ASM instance.

It is highly recommended to use an external RAID configuration such as RAID 1+0 for the ASM disks to ensure the redundancy of the disks.

2. *Normal Redundancy*: Oracle ASM uses two-way mirroring. This requires having two failure groups for mirroring. The effective disk space is one-half of the entire disk's capacity. The mirroring is at the file extent level, which means all the files have two copies of every extent: the primary extent and the mirroring extent. In order to achieve the maximum I/O bandwidth and load balance on all the disks, the primary extents are distributed across both failure groups. If the primary copy of an extent fails, the mirroring copy of the extent will be read from the other failure group.

3. *High Redundancy*: Oracle ASM uses three-way mirroring. It requires at least three failure groups for mirroring. Due to this three-way mirroring, the effective disk space is one-third of all the disk capacity. With this setting, the ASM diskgroup can tolerate the failure of two failure groups: if the primary copy of the extent fails, one of the mirroring copies will be used. If both the primary copy and one of the mirroring copies fail, the remaining mirroring copy will be used.

The ASM diskgroup can be created with the ASMCA GUI tool, or with a SQL command like this one after login to ASM instance using SYSASM privilege:

```
SQL>  CREATE DISKGROUP  data  NORMAL  REDUNDANCY
   FAILGROUP fg1 disk 'ORCL:DATA1' name data1, 'ORCL:DATA2' name data2,
   FAILGROUP fg2  disk  'ORCL:DATA3' name data3, 'ORCL:DATA4' name data4;
```

If you decide to take advantage of the RAID 1+1 configuration in external SAN storage, you can use external redundancy for the diskgroup:

```
SQL>  CREATE DISKGROUP  data  EXTERNAL  REDUNDANCY
   Disk 'ORCL:DATA1' name data1, 'ORCL:DATA2' name data2;
```

You can perform the diskgroup administration tasks in an ASM instance on one of the RAC nodes with sqlplus:

```
SQL>  Drop DISKGROUP data  INCLUDING CONTENTS;
SQL>  ALTER DISKGROUP DATA ADD DISK 'OCR:DATA3'  REBALANCE Power 3
SQL> ALTER DISKGROUP DATA DROP DISK data2  REBALANCE Power 3
```

The REBALANCE Power clause specifies the power for the rebalancing operation. Combining the add disk and drop disk operations in a single alter diskgroup operation will allow you to perform online storage migration for your database: that is, to migrate your database from one storage to another while keeping it online.

For example, imagine that you want to migrate your database from an old storage to a new storage. DATA1 and DATA2 ASM disks are on the old storage and DATA3 and DATA4 ASM disks are on the new storage. Executing the following 'alter diskgroup' SQL command in the ASM instance of one RAC node will migrate your database from the old storage to the new storage without any database downtime:

```
SQL>ALTER DISKGROUP DATA ADD DISK 'ORCL:DATA3' , 'ORCL:DATA4'
                  DROP DISK 'ORCL:DATA1','ORCL:DATA2' REBALANCE Power 8;
```

This feature has been widely used for storage migration and can be considered one of the greatest benefits of using ASM as the storage solution for Oracle Database.

The range of values for Power clause is 0-11 inclusive, if the diskgroup ASM compatibility is set to less than 11.2.0.2. This range is extended to 1-1024 if the diskgroup ASM compatibility is set to 11.2.0.0 or higher; for example, COMPATIBLE.ASM=12.1.0.0 in Oracle 12cR1 ASM.ASMCMD Utility and File System.

Oracle ASM provides the volume manager and a file system for the Oracle Database files.

When an ASM diskgroup is created in ASM, this diskgroup will be presented as a file system for Oracle Databases. Unlike the OS file system, which you see and manage through OS file system commands, ASM files have to be managed through the following two interfaces:

- *Oracle ASM command-line utility (ASMCMD)*: This utility provides a tool to administer Oracle ASM; for example, to manage ASM instances, diskgroups, and file access control for disk groups, files, and directories with a diskgroup.
- *SQL commands*: You can log in to the ASM instance to execute the SQL command. It also provides a set of V$ ASM views for you to query the status of the ASM disks, diskgroup, etc.

ASMCMD Utility

To use the ASMCMD utility, you need to log in as an OS user such as a grid user in SYSDBA group and set the ORACLE_SID and ORACLE_HOME environment variables:

```
$ env | grep ORA
ORACLE_SID=+ASM1
ORACLE_HOME=/u01/app/12.1.0/grid
```

Get into the ASMCMD utility by running the OS command 'asmcmd' to connect to the ASM instance.

```
[grid@k2r720n1 ~]$ asmcmd
ASMCMD>
```

Then you can run the ASMCMD commands in the interactive mode from the ASMCMD> prompt.

```
ASMCMD> du
Used_MB      Mirror_used_MB
1036439       1036439
```

Most ASMCMD commands are similar to Linux file system commands, such as ls, cd, mkdir, pwd, rm, etc. You can type "help" to get a list of ASMCMD commands. The ASMCMD utility can also be used in non-interactive mode. In non-interactive mode, you can run a single asmcmd command with the ASMCMD utility.

```
[grid@k2r720n1 ~]$ asmcmd ls
```

Or put a series of ASMCMD commands in a command file like this test.cmd file:

```
[grid@k2r720n1 ~]$ cat test.cmd
ls
pwd
du
```

And execute this command file using the ASMCMD utility

```
[grid@k2r720n1 ~]$  asmcmd <test.cmd
ASMCMD> DATA/
VOCR/
ASMCMD> +
ASMCMD> Used_MB      Mirror_used_MB
                 1036439                 1036439
```

ASM File System

We can use the ASMCMD utility to examine the ASM file system structure. When you enter the ASMCMD utility, you are at the very top of the ASM file system marked as "+," which is similar to "/" on the Linux file system. From the top, you can see all the diskgroups in ASM which are subdirectories under "+" and navigate the file system directory down using the cd command:

```
[grid@k2r720n1 ~]$ asmcmd
ASMCMD> pwd
+
```

```
ASMCMD> ls
DATA/
VOCR/
ASMCMD> cd DATA
ASMCMD> cd KHDB
```

On the ASM file system, each file is given a fully qualified file name during its creation.

```
+diskgroup/dbname/filetype/filetypetag.file.incarnation.
```

For example, the system tablespace is stored in: `file:'+VOCR/KHDB/DATAFILE/SYSTEM.256.789808227'`

```
ASMCMD> ls -l +VOCR/KHDB/DATAFILE/SYSTEM.256.789808227
Type      Redund  Striped  Time             Sys  Name
DATAFILE  UNPROT  COARSE   NOV 01 07:00:00  Y    SYSTEM.256.789808227
```

The control file: '+DATA/KHDB/CONTROLFILE/ Current.256.789808289':

```
    ASMCMD> ls -l +DATA/KHDB/CONTROLFILE/Current.256.789808289
Type         Redund  Striped  Time             Sys  Name
CONTROLFILE  UNPROT  FINE     NOV 01 07:00:00  Y    Current.256.789808289
```

This fully qualified file name also indicates the file directory structure where the file is located. You can use the cd command to navigate the directory structure to reach the leaf directory where this file is located.

```
ASMCMD> cd +VOCR/KHDB/DATAFILE
ASMCMD> ls
SYSAUX.257.789808227
SYSTEM.256.789808227
UNDOTBS1.258.789808227
UNDOTBS2.264.789808307
USERS.259.789808227
```

The ASM file that stores OCR and voting disks looks like this:

```
ASMCMD> ls -l +VOCR/kr720n-scan/OCRFILE/REGISTRY.255.789801895
Type     Redund  Striped  Time             Sys  Name
OCRFILE  UNPROT  COARSE   NOV 01 07:00:00  Y    REGISTRY.255.789801895
```

This is the ASM instance spfile:

```
ASMCMD> ls -l +VOCR/kr720n-scan/ASMPARAMETERFILE/REGISTRY.253.789801893
Type              Redund  Striped  Time             Sys  Name
ASMPARAMETERFILE  UNPROT  COARSE   JUL 28 05:00:00  Y    REGISTRY.253.789801893
```

■ **Note** Since these two files are not the database file, they use the Clusterware name 'kr720n-scan' instead of a database name on the file path.

Each ASM diskgroup can store multiple database files and Clusterware and ASM instance files. For example: +VOCR stores the files of the KHDB database and the kr720n-scan cluster's files:

```
ASMCMD> pwd
+VOCR
ASMCMD> ls
KHDB/
KHRN/
kr720n-scan/
```

Manage ASM Using SQL Command and V$ASM Views

A set of SQL commands and V$ ASM views can be used to administer ASM diskgroups and ASM disks as well as the ASM instance itself. In the last section, I showed the SQL commands that are used to create and alter diskgroups. This section covers some V$ ASM views and how to write queries using these views to monitor the capacity and performance information of ASM disks and ASM diskgroup. Some of the commonly used V$ ASM views include V$ASM_DISKGROUP, V$ASM_DISKS, VASM_DISK_STAT, VASM_DISKGROUP_STAT.

The following query shows the capacity and space usage of ASM diskgroups and the ASM disks:

```
SQL>SELECT D.PATH, D.TOTAL_MB DISKSIZE, G.NAME GROUP_NAME,
G.TOTAL_MB GROUPSIZE,  G.FREE_MB GROUP_FREE
FROM V$ASM_DISK D, V$ASM_DISKGROUP G WHERE
 D.GROUP_NUMBER = G.GROUP_NUMBER;

PATH            FAILGROUP       DISKSIZE GROUP_NA GROUPSIZE GROUP_FREE
---------------    ---------------          --------          --------          ---------
----------
/dev/dm-7      DATA_0000       1433589      DATA          1433589           515833
/dev/dm-5      VOCR_0000       138231       VOCR           138231           19298
```

And this query shows the I/O performance on each ASM disk:

```
SQL> SELECT PATH, READS, WRITES, READ_TIME, WRITE_TIME,
READ_TIME/ READS  AVEREADTIME,  WRITE_TIME/ WRITES  AVGWRITETIME
FROM  V$ASM_DISK_STAT;

PATH            READS     WRITES  READ_TIME    WRITE_TIME AVEREADTIME AVGWRITETIME
---------- ---------- ---------- ---------- -                              ---------            ------
-----                ------------
/dev/dm-7        2371     193779      .747134            258.353864  .000315113          .00133324
/dev/dm-5     2237295  600232      635.231634     453.596385   .000283928          .000755702
```

Store OCR and Voting Disk in ASM

This section will look at ways of storing the two most important components of Oracle Clusterware: OCR and voting disk files in ASM.

Choose ASM for OCR and Voting Disk at GI Installation

On different Oracle Clusterware releases, storage for OCR and voting disk varies: raw devices on 10gR2, block devices on 11gR1. Starting with 11g R2, OCR file and voting disk files can be stored in ASM. Unless you are upgrading your Clusterware from 11gR1 to 11gR2 where you can keep the voting disk in the block device, you should put the voting disk file in ASM. OCR and voting disk files can be stored in the same ASM diskgroup as Oracle Database files or in a separate ASM diskgroup. The ASM diskgroup that stores voting disk files can be configured with one of the following three redundancy level settings. These are similar to a regular diskgroup for database files, but their requirement for the minimum numbers of failure groups is higher than those diskgroups that only store database files. The number of the voting disk files is determined by the number of the failure groups of the ASM diskgroup that stores the voting disk files:

1. *External Redundancy*: no mirroring; only one failure group is needed. This provides only one copy of the voting disk file. It is strongly recommended to have an external RAID configuration for this setting.
2. *Normal Redundancy*: at least three failure groups are needed. This provides three copies of the voting disk files.
3. *High Redundancy*: at least five failure groups needed. This provides five copies of the voting disk files.

If you decide to store the voting disk file in the same ASM diskgroup as the data files, adding an additional failure group could mean adding a huge disk space, which you may not have. To solve this problem, a new quorum failure group is introduced. When this new failure group is added to a diskgroup, it is only used to store an extra copy of voting disk file, and non-database files will be stored in this failure group. As a result, the size of the disk for this failure group can be much smaller than other disks in the ASM diskgroup. For example, the ASM disk for a quorum failure group can be as small as 300MB to cover the 200MB voting disk file. The following example shows how to use the quorum failgroup clause to include a quorum failure group in a diskgroup:

```
CREATE DISKGROUP data NORMAL REDUDANCY
FAILGROUP fg1 DISK 'ORCL:DATA1'
FAILGROUP fg1 DISK 'ORCL:DATA2'
QUORUM FAILGROIUP fg3 DISK 'ORCL:OCR1'
ATTRIBUTE  'compatible.asm' = 12.1.0.0.0;
```

In this case DISK ORCL:DATA1 and ORCL:DATA2 can be 200GB each, DISK ORCL:OCR1 can be small as 300MB.

During the Grid Infrastructure installation you will be asked to provide the disks to create the ASM diskgroup for OCR and voting disk files (Figure 5-10). You have an option to select the redundancy level setting of the ASM diskgroup for OCR and voting disks.

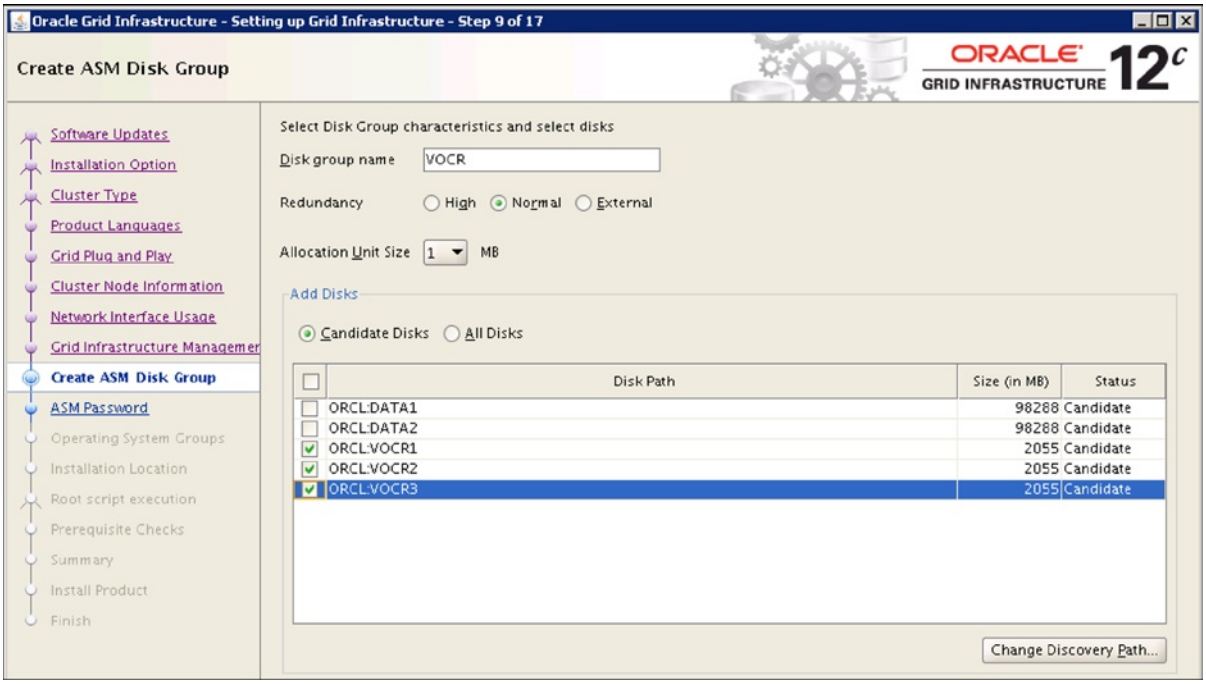


Figure 5-10. *Select three-ASM disk for the high-redundancy diskgroup*

Oracle 12c R1 introduced the Grid Infrastructure Management repository (also called Cluster Health Monitor (CHM) repository), which is a central database used to store all the metrics database collected by the System Monitor Service process of the Clusterware in all the cluster nodes. Currently, this repository is configured during Oracle Grid Infrastructure installation. Refer to the CHM section in Chapter 2. Figure 2-6 shows the Grid Infrastructure Management Repository Option that you can select during Grid Infrastructure installation. If you select the Grid Infrastructure Management Repository option, the diskgroup you are going to create needs to be big enough to store the CHM repository as well as OCR and voting disks; otherwise you will get the error message shown in Figure 5-11. In my test run, I allocated a 20GB diskgroup to store CHM repository, OCR, and voting disks. Once I made this ASM diskgroup a bigger size such as 100 GB, this issue was solved.

Figure 5-11. *The diskgroup has insufficient space for the CHM repository*

As part of a successful Grid Infrastructure installation, the VOCR diskgroup was created with three failure diskgroups: VOCR1, VOCR2, and VOCR3. Three copies of voting disk files are stored in this diskgroup, as shown in the following queries:

```
SQL> SELECT d.PATH, g.NAME, d.FAILGROUP FROM V$ASM_DISK d, V$ASM_DISKGROUP g
           where d.GROUP_NUMBER = g.GROUP_NUMBER and g.name='OCRVOTDSK';

PATH                NAME                      FAILGROUP
----------          ----------               ------------
OCRL:VOCR1          VOCRVOTDSK                VOCR1
OCRL:VOCR2          VOCRVOTDSK                VOCR2
OCRL:VOCR3          VOCRVOTDSK                VOCR3
```

Three copies of voting disk files are as follows:

```
[grid@kr72On1 bin]$ ./crsctl   query css votedisk
##    STATE    File Universal Id                                 File Name          Disk group
--    -----    -----------------------------------              --------------     ------------
 1.   ONLINE   8b562c9b2ec34f88bfe8343142318db7                  (ORCL:VOCR1)        [VOCR]
 2.   ONLINE   1c88a98d5c9a4f88bf55549b6c2dc298                  (ORCL:VOCR2)        [VOCR]
 3.   ONLINE   7581ef0020094fccbf1c8d3bca346eb1                  (ORCL:VOCR3)        [VOCR]
]

Located three voting disk(s).
```

Unlike the voting disk files, only one copy of OCR can be stored in one ASM diskgroup or a cluster file system. Oracle requires at least one OCR copy with the ability to add up to five copies of OCR. If you want to store multiple copies of the OCR, you can store them on multiple ASM diskgroups or a combination of ASM diskgroups and cluster file systems.

In order to add an additional copy of OCR on a different ASM diskgroup (for example, DATA2), you can use the following command:

```
 $ocrconfig -add +DATA2
```

This example shows how to add an additional copy of your OCR on a cluster file system labeled /u01/data:

```
 $ocrconfig –add /u01/data/ocr.dbf
```

Another example showing how to move the original OCR to a different ASM diskgroup DATA3 is as follows:

```
$ocrconfig –replace +VOCR replacement +DATA3
```

The following example shows how to list all copies of OCR:

```
$ ocrcheck
Status of Oracle Cluster Registry is as follows :
Version : 3
Total space (kbytes) : 262120
Used space (kbytes) : 2860
Available space (kbytes) : 259260
ID : 2322227200
Device/File Name :+VOCR Device/File integrity check succeeded
```

```
Device/File Name :+DATA1 Device/File integrity check succeeded
Device/File Name :+DATA2 Device/File integrity check succeeded
Device/File not configured
Device/File not configured
Cluster registry integrity check succeeded
```

In this example, there are three copies of OCR. They are stored in three ASM diskgroups: +VOC, +DATA1, and +DATA2, respectively.

Move OCR and Voting Disk Files to a New ASM Diskgroup

Although it may not frequently occur, you may need to move your OCR and voting disk files to a different ASM diskgroup. This kind of task may be required on two occasions: 1) moving the OCR and voting disk files of a newly upgraded 11gR2 Clusterware from a raw device or block device to an ASM diskgroup; and 2) moving the OCR and voting disk files from one diskgroup to a new diskgroup. The methods for moving OCR and voting disks in these two cases are very similar. The following is an example of implementing the OCR and voting disk files from an ASM diskgroup. These steps also apply if you need to move them from raw devices or block devices.

1. Show that the current OCR and voting disks are in +OCRV diskgroup:

```
$ ocrcheck
Status of Oracle Cluster Registry is as follows:
         Version                  :          3
         Total space (kbytes)     :     262120
         Used space (kbytes)      :       3480
         Available space (kbytes) :     258640
         ID                       : 1318929051
         Device/File Name         :      +OCRV
                                    Device/File integrity check succeeded

                                    Device/File not configured

                                    Device/File not configured

                                    Device/File not configured

                                    Device/File not configured

         Cluster registry integrity check succeeded

         Logical corruption check bypassed due to non-privileged user

$ crsctl query css votedisk
##  STATE    File Universal Id                  File Name      Disk group
--  -----    -----------------                  ---------      ---------
1. ONLINE    327b1f75f8374f1ebf8611c847ffbdad    (ORCL:OCR1)    [OCRV]
2. ONLINE    985958d30e4f4f45bf8134a9b07cae4f    (ORCL:OCR2)    [OCRV]
3. ONLINE    80650a28b2fa4ffdbf93716a8bc22668    (ORCL:OCR3)    [OCRV]
```

2. Create the ASM diskgroup “VOCR”:

```
SQL>CREATE DISKGROUP VOCR NORMAL REDUNDANCY
FAILGROUP VOCRG1 DISK  'ORCL:VOCR1' name VOCR1
FAILGROUP VOCRG2 DISK  'ORCL:VOCR2' name VOCR2
FAILGROUP VOCRG3 DISK  'ORCL:VOCR3' name VOCR3;
SQL>alter diskgroup VOCR set attribute 'compatible.asm'=12.1.0.0.0';
```

3. Move OCR to the new ASM diskgroup:

   Add the new ASM diskgroup for OCR

```
$ocrconfig -add +VOCR
```

   Drop the old ASM diskgroup from OCR

```
$ocrconfig -delete +OCRV
```

4. Move the voting disk files from old ASM diskgroup to the new ASM diskgroup:

```
 $ crsctl replace votedisk +VOCR
Successful addition of voting disk 29adbae485454f72bf9d66519c921e17.
Successful addition of voting disk 529b802332674f9fbf8543d5acd55672.
Successful addition of voting disk 017523650cd64f47bf65bb90e8ed98e6.
Successful deletion of voting disk 327b1f75f8374f1ebf8611c847ffbdad.
Successful deletion of voting disk 985958d30e4f4f45bf8134a9b07cae4f.
Successful deletion of voting disk 80650a28b2fa4ffdbf93716a8bc22668.
Successfully replaced voting disk group with +VOCR.
CRS-4266: Voting file(s) successfully replaced

$ crsctl query css votedisk
##  STATE    File Universal Id                  File Name       Disk group
--  -----    -----------------                  ---------       ---------
1. ONLINE    29adbae485454f72bf9d66519c921e17   (ORCL:VOCR1)    [VOCR]
2. ONLINE    529b802332674f9fbf8543d5acd55672   (ORCL:VOCR2)    [VOCR]
3. ONLINE    017523650cd64f47bf65bb90e8ed98e6   (ORCL:VOCR3)    [VOCR]
Located 3 voting disk(s).
```

   Make sure the /etc/oracle/ocr.loc file gets updated to point to the new ASM diskgroup:

```
$ more /etc/oracle/ocr.loc
ocrconfig_loc=+VOCR
local_only=FALSE
```

5. Shut down and restart CRS using the force option “the Clusterware on all the nodes”:

```
#./crsctl stop  crs –f
# ./crsctl start crs
CRS-4123: Oracle High Availability Services has been started.
# ./crsctl check crs
CRS-4638: Oracle High Availability Services is online
CRS-4537: Cluster Ready Services is online
```

```
CRS-4529: Cluster Synchronization Services is online
CRS-4533: Event Manager is online
```

Before you can dismount the old ASM diskgroup, you need to check if there is anything on it. In this example, we found that the ASM instance spfile was on that diskgroup and needed to move the spfile to the new ASM diskgroup.

6. Move the ASM spfile to the new diskgroup.

   Check the current ASM spfile location:

```
  SQL> show parameter spfile

NAME         TYPE            VALUE
----         -----------     ------------------------------------------------------------
spfile       string         +OCRV/k2r720n-cluster/asmparameterfile/registry.253.773325457
    Create a pfile from the spfile:
       SQL> create pfile='/home/grid/init+ASM.ora' from spfile;

    Recreate the new spfile on the new diskgroup VOCR from the pfile:

    SQL> create spfile='+VOCR' from pfile='/home/grid/init+ASM.ora';

    Restart HAS (Oracle High Availability Services)

# ./crsctl stop has
# ./crsctl start has

    Check the new spfile location:

SQL> show parameter spfile;

NAME         TYPE           VALUE
----         -----------    ------------------------------------------------------------
spfile       string         +VOCR/k2r720n-cluster/asmparameterfile/registry.253.779041523
```

In this example, there are a few places where you need to pay special attention. Since the new ASM diskgroup VOCR is for the OCR and voting disk files, you need to follow the failure group rules, namely, three failure groups for normal redundancy and five failure groups for high redundancy; otherwise, the command 'crsctl replace votedisk +VOCR' would fail with the error "ora-15274 error, Not enough failgroups(s) to create voting file." You also need to set the compatible parameter to 12.1.0.0.0 for the new VOCR diskgroup in order for the VOCR diskgroup to store the OCR and the voting disk file. The default setting of the compatible parameter of an ASM diskgroup is '10.1.0.0.0'.

ACFS

When it was introduced in Oracle 10gR1, Oracle ASM was designed to be a volume manager and a file system for Oracle Database files, not for general-purpose files. This remained the case until Oracle ACFS was introduced in Oracle 11gR2. Oracle ACFS is an extension of Oracle ASM to store those non–Oracle Database files. Such files can be software binaries such as Oracle Database binary, applications files, trace file, BFILES, video, audio, text, images, and other general-purpose files. To help you understand the file types supported by Oracle ASM and Oracle ACFS, here are some general guidelines:

1. Oracle ASM is designed for and optimized to provide the best performance for Oracle Database files such as data files, controlfiles, spfiles, redo logs, tempfiles, and the OCR and voting disk files of Oracle Clusterware.

2. In Oracle 11gR2, Oracle ACFS provided support for non-database files and didn't support database files. This limitation has been removed in Oracle 12c. With Oracle 12c, Oracle ACFS provides support for all the database files for Oracle Database 12cR1 except for data files and redo logs in an Oracle Restart configuration and database files on Windows. Oracle ACFS on Windows supports only RMAN backups, archive logs, and data pump dumpsets.

3. Oracle ACFS supports Oracle Database home files for Oracle Database release 11gR2 or higher. However, it doesn't support any files associated with the management of Oracle ASM. It doesn't support Oracle Grid Infrastructure home, which should be stored in the local disk in each RAC node.

4. Starting with 11.2.03, Oracle ACFS supports the RMAN backups archive logs (ARCGIVELOG file type) and data pump dump sets (DUMPSET file type).

In Oracle 12cR1, Oracle ACFS supports all database files for Oracle Database 12cR1, except for data files and redo logs in an Oracle Restart configuration. It also doesn't support data files for Oracle 12cR1 on Windows. Like Oracle ASM, Oracle ACFS is based on the Oracle ASM technology. Since Oracle ACFS files are stored in the Oracle ASM diskgroup, ACFS leverages data reliability through redundancy and mirroring within the diskgroup and also benefits from I/O load balancing through data striping among the ASM disks. Oracle ACFS is a cluster file system which can be accessed from all nodes of a cluster. Some of the use cases of this cluster file system include the shared Oracle RAC Database home and other applications' binaries and files that need to be shared by multiple hosts.

Unlike Oracle Database files in Oracle ASM, which are created in an ASM diskgroup directly, Oracle ACFS is based on ASM volumes which are created in an Oracle ASM diskgroup. These volumes are equivalent to the "logical volume" in Linux. Like the LVM (Logical Volume Manager), which provides logical volume management in Linux, starting with Oracle 11gR2, a new feature called Oracle ASM Dynamic Volume Manager (ADVM) was introduced to manage ASM volumes and also to provide the standard device driver interface to file systems such as ACFS, OCFS2, and regular ext3fs.

Figure 5-12 shows an example of ASM diskgroup configuration. In this example, there are two ASM diskgroups in ASM: DATADG and ACFSDG. The diskgroup DATADG is for the Oracle Database files and Oracle Clusterware OCR and voting disk files. Two databases, namely, RACDB1 and RACDB2, store the database files in two directories of the ASM file system in the diskgroup 'DATADG':

```
+DATADG/RACDB2/
+DATADG/RACDB2/
```

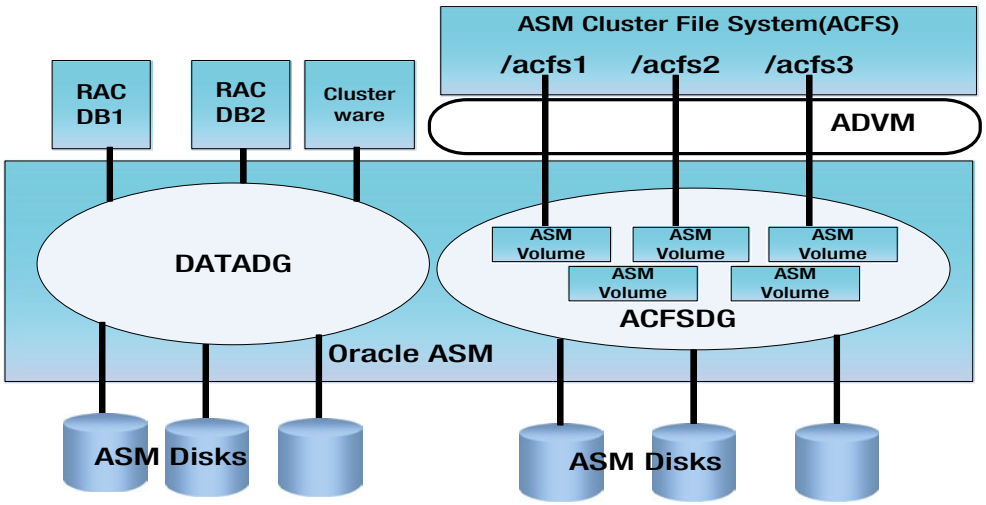


Figure 5-12. *Oracle ASM and Oracle ACFS*

And the Oracle 11gR2 Clusterware stores its OCR/voting disk files in +DATADG/<Clustername>.

The diskgroup ACFSDG is for Oracle ACFS. ASM volumes are created in the ASM diskgroup ACFSDG. For each of these volumes, an ASM device on the operating system is automatically created under /dev/asm. These volumes are managed by Oracle ADVM. Oracle ACFS can be created on these volumes. Oracle ADVM provides the standard device driver interface for ACFS to access these volumes. These ACFS are mounted under given OS mount points; for example, /acfs1, /acfs2, and /acfs3 in this case.

Create ACFS

Before starting to create an ACFS, you have to ensure that several prerequisites are met in your environment.

In Oracle 11gR2, you need to ensure that the required modules are loaded by running the lsmode command. These modules are as follows: oracleasm, oracleoks, and oracleacfs. If they are all loaded, the lsmod command should have the following results:

```
$lsmod
   Module          Size       User by
   Oracleacfs      781732     0
   Oracleadvm      212736     0
   Oracleoks       224992     2 oracleacfs, oracleadvm
   oracleasm       46484      1
```

If these modules are not automatically loaded, you need to load them manually using the acfsload command. You might see an error message like this:

```
[root@k2r720n1 ~]# /u01/app/11.2.0/grid/bin/acfsload start -s
ACFS-9459: ADVM/ACFS is not supported on this OS version: '2.6.32-220.el6.x86_64'
```

This indicates that the version of OS is not supported by ACFS. Refer to MOS note [1369107.1] to check the current ACFS supported on the OS platform. For example, at the time this chapter was written, ACFS was supported for update 3 or later for Oracle Linux 5.x and Red Hat Enterprise Linux 5.x. For Oracle Linux 6.x and Red Hat Enterprise Linux 6.x, it would be necessary to apply 11.2.0.3.3.GI PSU patch for 6.0, 6.1, 6.2 and required to apply 11.2.0.3.4GI PSU patch for 6.3. If the ACFS is not supported in your OS kernel version, the acfsload command will report the error, as shown in the preceding. And in the ASMCA GUI interface, the Volume tab and ASM Cluster File systems tab are grayed out as shown in Figure 5-13. You need to follow the support document's instructions to get the ACFS supported in your OS. For example, since the test cluster is running EL 6.2 with kernel version '2.6.32-220.el6.x86_64', it is required to apply 11.2.0.3.3.GI PSU (Patch 13919095) to get the ACFS supported.

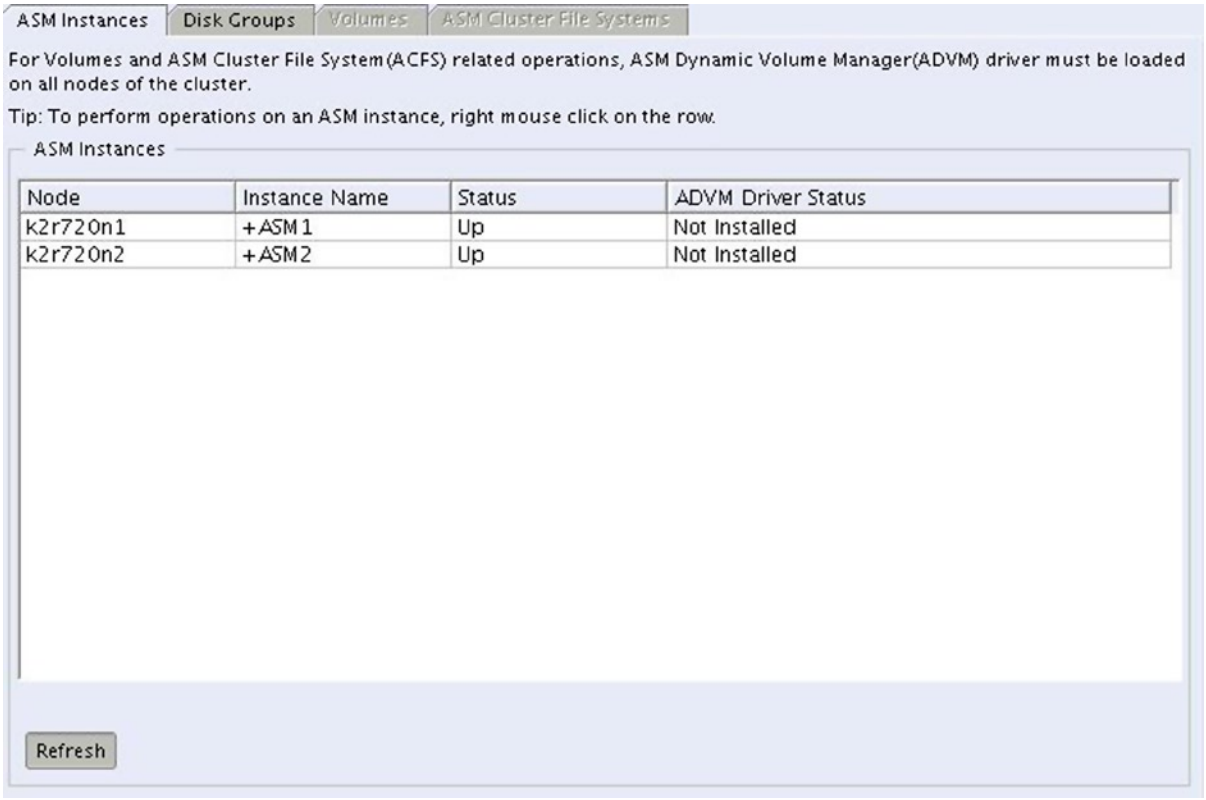


Figure 5-13. *Volume tab and ASM Cluster file systems tab grayed out*

In Oracle Clusterware 12cR1 and Oracle ASM 12cR1, the automatic loading of these modules is supported. For example, these are the related modules automatically loaded on Oracle Linux 6.3 after Oracle 12cR1 Grid Infrastructure installation.

```
# lsmod | grep oracle
          oracleacfs            3053165   2
          oracleadvm            320180    8
          oracleoks             417171    2 oracleacfs,oracleadvm
          oracleasm             53352     1
```

The creation of ACFS starts with the ASM volumes. Once all the prerequisites are met, you need to create ASM volumes for ACFS. ASM volumes can be created in one of the following ways:

- Use ASMCA GUI tool
- User Oracle Enterprise Manager
- Use ASMCMD tool
- Use SQL*Plus

Here is an example of creating an ASM volume acf_vm1 on diskgroup ACFSDG1 using the ASMCMD tool:

```
ASMCMD> volcreate -G ACFSDG1 -s 1g acf_vm1
ASMCMD>
ASMCMD> volinfo -G ACFSDG1 acf_vm1
Diskgroup Name: ACFSDG1

         Volume Name: ACF_VM1
         Volume Device: /dev/asm/acf_vm1-105
         State: ENABLED
         Size (MB): 1024
         Resize Unit (MB): 32
         Redundancy: UNPROT
         Stripe Columns: 4
         Stripe Width (K): 128
         Usage: Mountpath
```

Now you can use the mkfs command to create ACFS based on this volume device /dev/ams/ acf_vm1-105:

```
# mkfs -t acfs  /dev/asm/acf_vm1-105
mkfs.acfs: version                   = 12.1.0.1.0
mkfs.acfs: on-disk version           = 39.0
mkfs.acfs: volume                    = /dev/asm/acf_vm1-105
mkfs.acfs: volume size               = 1073741824
mkfs.acfs: Format complete.
```

Then you can mount the ACFS. The acfsutil registry command can be used to register the ACFS with the ACFS mount registry. Once being registered on the registry, the ACFS mount registry will ensure that the ACFS are mounted automatically.

Create an OS directory as a mount point: /u01/acfs/ asm_vol1

```
mkdir /u01/acfs/asm_vol1
/sbin/acfsutil  registry -a /dev/asm/acf_vm1-105 /u01/acfs/asm_vol1
acfsutil registry: mount point /u01/acfs/asm_vol1 successfully added to Oracle Registry
```

Now you can see the ACFS that is mounted on /901/acfs/asm_vol1.

```
# df -k | grep 'Filesystem \|asm'
Filesystem            1K-blocks      Used      Available  Use%      Mounted on
/dev/asm/acf_vm1-105  1048576        119112    929464     12%       /u01/acfs/asm_vol1
```

Create ACFS for Oracle RAC Home with ASMCA

In an Oracle RAC Database environment, you have the option of storing the Oracle RAC Database software ORACLE_HOME in an ACFS. You can either use ASMCMD to follow the steps just mentioned to manually create the ACFS for Oracle HOME, or you can use the ASMCA shortcut based on GUI to simplify the creation of ACFS for shared Oracle RAC Database home. Assume that you have created an ASM diskgroup ORAHOME for this ACFS. This step usually occurs after the successful configuration of the Grid Infrastructure installation and configuration and before you are ready to install Oracle RAC software. The creation of this ACFS allows you to install Oracle RAC software on this ACFS to have an Oracle RAC home that can be shared by all the RAC nodes.

We use the ASMCA utility to create a dedicated ASM diskgroup as shown in the following diagram (Figure 5-14).

1. As shown in Figure 5-14, start ASMCA and highlight the ASM diskgroup ORAHOME that is the diskgroup for ACFS. Right-click to open a drop-down menu and select "Create ACFS for Database Home" from the menu.

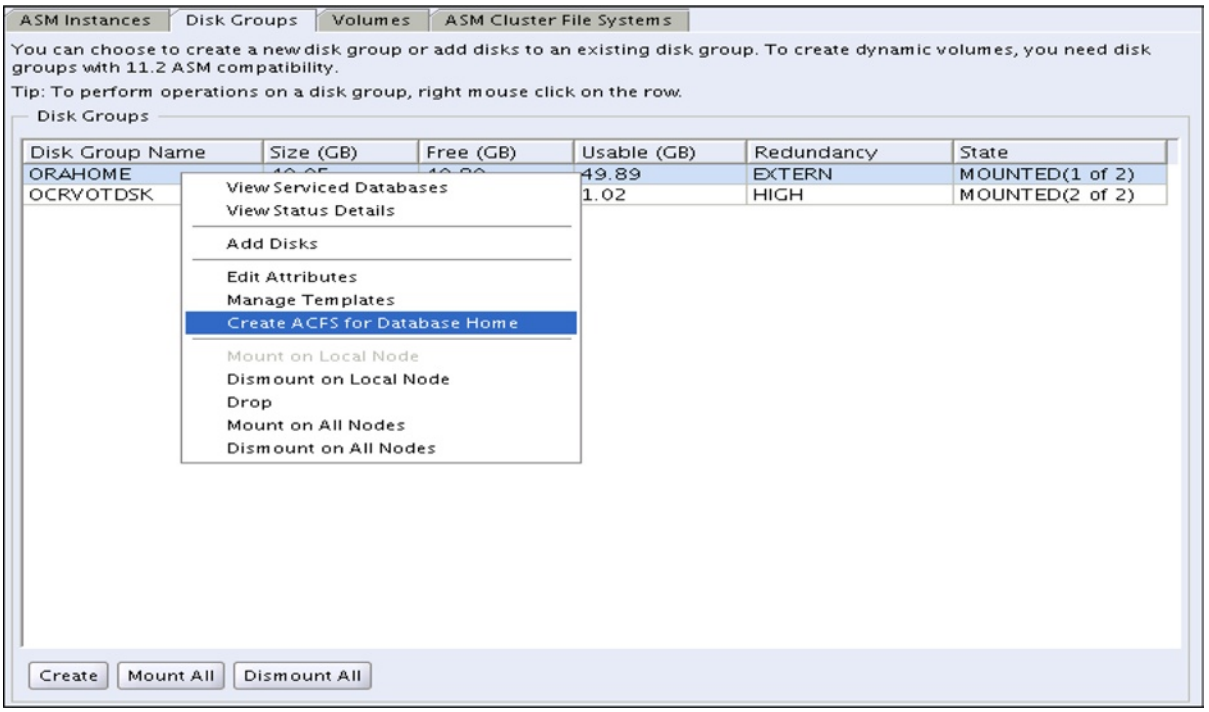


Figure 5-14. *Start "Create ACFS for Database Home" on the ACSMCA menu*

2. Specify the database home's volume name, mountpoint, size, owner name, and owner group (Figure 5-15).

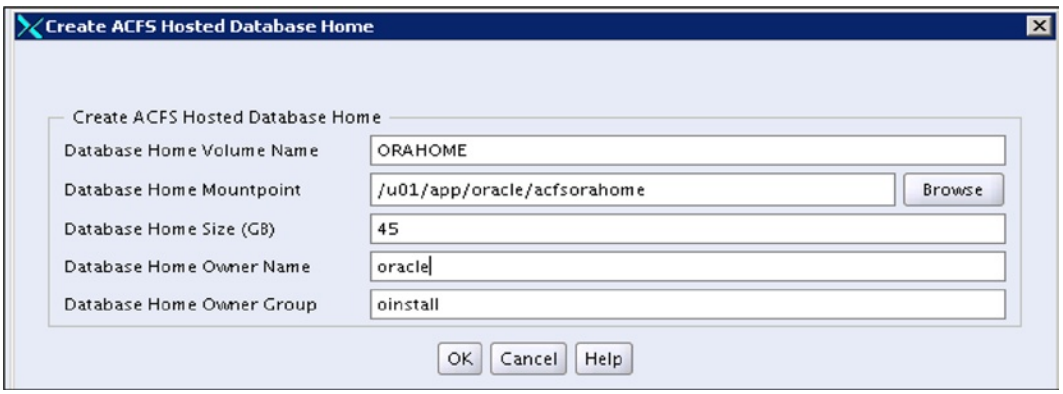


Figure 5-15. *Specify the parameters for Oracle database home volume*

3. Click OK to start the creation process (Figure 5-16).

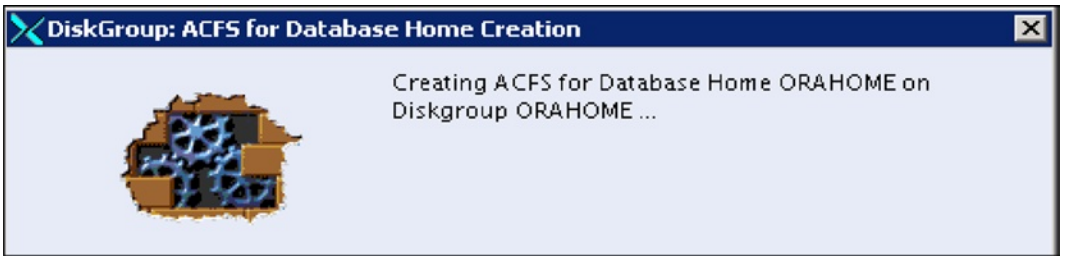


Figure 5-16. *Creating ACFS for database home*

4. Run the acfs_script.sh as the root user on one of the RAC nodes to complete the creation of ACFS Oracle home (Figure 5-17).

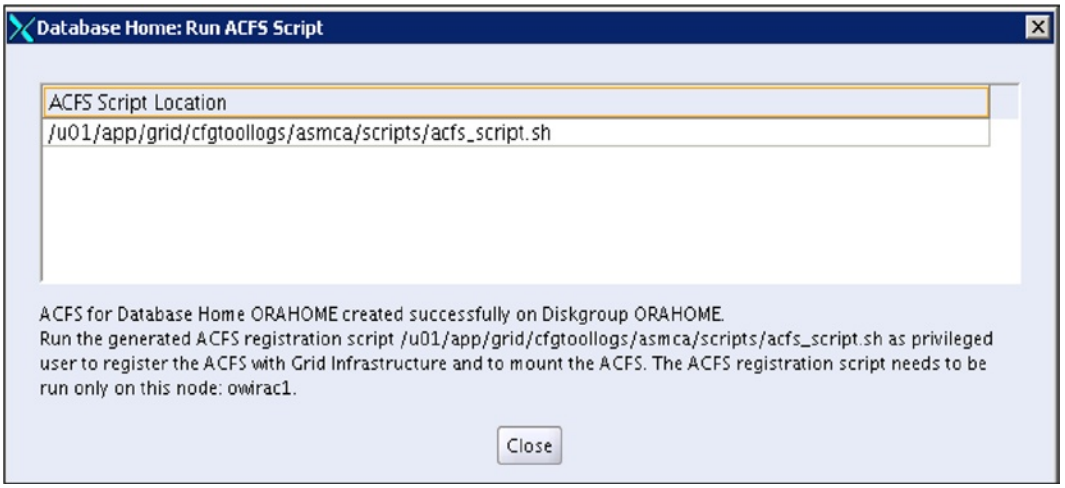


Figure 5-17. *Prompt for acfs_script*

5. Check the ACFS:

Node1:

```
# df -k | grep 'Filesystem \|u01'
Filesystem            1K-blocks Used       Available   Use%    Mounted on
/dev/asm/orahome-10   47185920  159172     47026748    1%      /u01/app/oracle/acfsorahome
```

Node2:

```
#df -k | grep 'Filesystem \|u01'
Filesystem           1K-blocks Used       Available    Use%    Mounted on
/dev/asm/orahome-10   47185920  159172      47026748     1%      /u01/app/oracle/acfsorahome
```

Summary

Shared storage is one of the key infrastructure components for Oracle Grid Infrastructure and the RAC Database. Storage design and configuration can largely determine RAC Database availability and performance. The introduction of Oracle ASM has significantly simplified storage configuration and management. The striping and mirroring features of Oracle ASM also help improve storage I/O performance and reliability against hardware failure.

CHAPTER 6

Application Design Issues

by Riyaj Shamsudeen

Inefficient application design strategy is the root cause of many customer RAC escalations.

Bad design practices causing minor performance issues in a single-instance database are magnified if you convert the database to a RAC database. Failures and cost overruns in many RAC deployment projects can be attributed to application design patterns that are not suitable for RAC. These design flaws are detrimental in single-instance databases, but the harm might be amplified in RAC. For example, poor design of sequence caching will cause only minor performance issues in a single-instance database; however, in a RAC database, that flaw can lead to a completely hung cluster.

In this chapter, I will discuss these design patterns, methods to troubleshoot these issues, and possible strategies for resolving them.

Localized Inserts

Excessive insert concurrency is one of the most common issues faced in a RAC database. Although too much concurrency itself is not an issue, either a flaw in an object design or an invalid use of application affinity can trigger massive performance issues.

In a B-Tree index, values are stored in ascending or descending order of indexed columns. Consider a unique index of the employee_id column in the employee table, populated by an emp_id_seq sequence with a current value of 1,000. An insert into the employee table using that sequence-generated value will store an entry in the rightmost leaf block of the index tree because 1,000 is the current maximum value. Subsequent inserts retrieving values from the emp_id_seq sequence will also populate entries in that leaf block until the leaf block is full. After the leaf block is full, a new block will be added to the index tree and subsequent inserts will populate index entries in the newly added leaf block. Essentially, new inserts into the employee table will populate index entries in the current rightmost leaf block of the index tree. In Figure 6-1, recently inserted values (1,000 to 1,003) are populated in the rightmost leaf block of the index tree.[1]

[1]Note that Figure 6-1 shows only four rows in the leaf block. Usually, hundreds of row entries will exist in a leaf block.

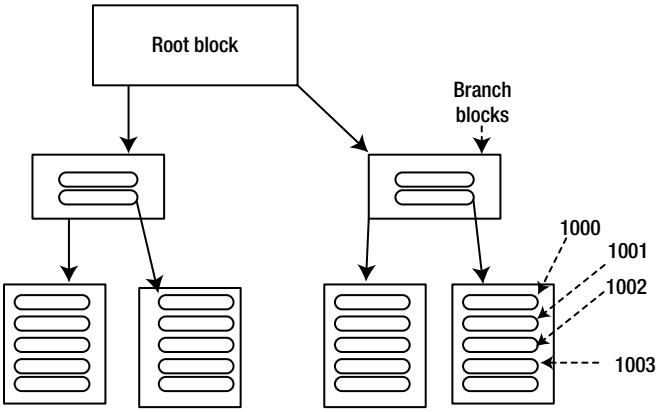


Figure 6-1. *Leaf block contention*

If insert concurrency into the employee table increases, contention on the current rightmost leaf block will increase sharply. Contention on the rightmost leaf block is exacerbated because the leaf block splits when the block becomes full. In a single-instance database, leaf block contention will result in waits for events such as 'buffer busy' waits, 'cache buffers chain' latch waits, and so on. In a RAC database, the current rightmost leaf block must be transferred between the instances, leading to an increase in global cache transfer. Further, sessions will suffer from wait events such as gc buffer busy acquire and gc buffer busy release as global cache locks are acquired and released. The effect of waiting for these global events in a RAC database will be an order of magnitude larger than the single-instance waits, possibly leading to an unusable application. This phenomenon is called *right-hand index growth contention*.

Figure 6-2 shows the output from gv$session and the effect of leaf block contention in a busy database. Thousands of sessions were stuck waiting for gc buffer busy event waits due to index leaf block contention. Waits for buffer busy wait events and ITL contention events are also shown in the output.

INST_ID	SQL_ID	EVENT	STATE	COUNT(*)
4	4jtbgawt37mcd	gc cr request	WAITING	9
3	4jtbgawt37mcd	gc cr request	WAITING	9
3	a1bp5ytvpfj48	gc buffer busy	WAITING	11
4	a1bp5ytvpfj48	gc buffer busy	WAITING	17
4	14t0wadn1t0us	gc buffer busy	WAITING	33
4	gt1rdqk2ub851	gc buffer busy	WAITING	34
4	a1bp5ytvpfj48	buffer busy waits	WAITING	35
2	a1bp5ytvpfj48	gc buffer busy	WAITING	65
1	a1bp5ytvpfj48	gc buffer busy	WAITING	102
2	7xzqcrdrnyw1j	gc buffer busy	WAITING	106
2	7xzqcrdrnyw1j	enq: TX - index c	WAITING	173
1	7xzqcrdrnyw1j	gc buffer busy	WAITING	198
3	7xzqcrdrnyw1j	gc buffer busy	WAITING	247
4	7xzqcrdrnyw1j	gc buffer busy	WAITING	247

Figure 6-2. *Events in a busy database*

Indexed columns populated by current date or current timestamp values can suffer a similar fate because rows inserted at the same time will have the same column values. So, recent values will be populated in the current rightmost leaf block of the index. The issue is monotonically increasing values populating index entries in the rightmost leaf block of the index. Hence, even non-unique indexes can induce right-hand index growth. The root cause of this problem is an invalid application design issue, and unfortunately the effects are magnified in a RAC database.

There are at least four options to resolve a right hand growth index contention issue:

1. Partition the indexes by a hash partitioning method with one or more indexed columns as partitioning keys.[2] With this hash partitioning technique, multiple index trees are created for an index and therefore, contention is distributed among multiple current rightmost leaf blocks of the index partitions. Figure 6-3 shows an index partitioned by a hash with two partitions. Because each partition has its own index tree, the values are spread between two leaf blocks, thereby reducing contention by half. So, if you partition an index by the hash partitioning method with 64 partitions, then the values are distributed among 64 leaf blocks of the index, dividing the contention 64-fold.

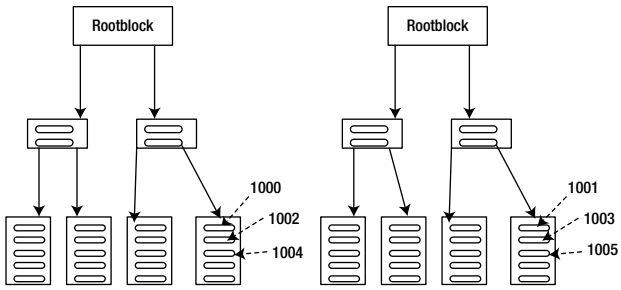


Figure 6-3. *Hash-partitioned index*

Further, a hash-partitioned index supports the index range scan operation. Hence, the ill effects of hash partitioning an index are minimal.

2. Partition the table by hash and recreate indexes as locally partitioned indexes. This technique has advantages similar to hash-partitioned indexes, but the difference is that the table is also partitioned by hash. This technique has an added advantage of reducing the contention in the table blocks. I should caution you that if your queries do not achieve partition pruning,[3] then logical reads can increase because all index partitions must be accessed to search for matching rows. For example, consider the EMP table, hash partitioned by employee_id, with a locally partitioned index on the dept_id column. Because the dept_id column is not part of the partitioning key, predicates specifying the dept_id column must search in all index partitions of the dept_id index. This increase in logical reads will pose a problem only if the SQL statement is executed too frequently. However, the advantages of a partitioned table outweigh this concern.

[2]You need a license for a partitioning option to implement this solution.
[3]Partition pruning is an SQL optimization technique to limit the search to few partitions.

3. Convert the index to a reverse-key index type if you do not have a license for a partitioning option or if your database uses Standard Edition software. Because column values are reversed while they are stored in the index blocks, index entries will be distributed to numerous leaf blocks, relieving the contention. However, because the values are distributed between almost all leaf blocks of the index, it is possible to pollute the buffer cache with these index blocks, causing increased physical reads. Another disadvantage with this option is that reverse key indexes do not support the index range scan operation.

4. Implement software-driven localized sequence access, with the sequence generating a different range of values. For example, you can convert an application to use multiple sequences generating a disjoint range of values instead of a sequence, such as an emp_id_seq1 sequence with a starting value of 1 billion, an emp_id_seq2 sequence with a starting value of 2 billion, and so on. The application will be coded in such a way that sessions connected to instance 1 will access the emp_id_seq1 sequence, sessions connected to instance 2 will access the emp_id_seq2 sequence, and so on. Hence, applications will be inserting values in a different range in each instance, reducing leaf block contention. Disadvantages with this option are that it requires a code change, and it does not resolve the problem completely as leaf block contention can still creep up within an instance.

To learn about methods to identify objects inducing or suffering from gc buffer busy performance issues, see Chapter 10.

Excessive TRUNCATE or DROP Statements

Applications executing excessive numbers of TRUNCATE and DROP commands will not scale in a single-instance database. In a RAC database, global waits triggered by these DDL commands will cause severe performance issues.

The TRUNCATE and DROP commands trigger object-level checkpoints for the table to be truncated or the object to be dropped. In a single-instance database, the object queue in the local buffer cache must be scanned to identify blocks currently in the buffer cache. In a RAC database, object queues in the buffer caches of all instances must be scanned to complete the TRUNCATE and DROP commands. Object checkpoints are triggered by posting remote background processes by means of local background processes. If numerous object checkpoints are triggered concurrently, then these checkpoint requests might be queued, leading to longer wait times to complete the commands. Also, as the size of the buffer cache increases, the amount of work to complete these commands also increases, and so this problem can be magnified in large buffer caches.

■ **Note** An object-level checkpoint also is triggered for dependent objects such as indexes. Hence, tables with numerous indexes can suffer the worst consequences.

The TRUNCATE command is a DDL command, so parse locks (also known as library cache locks) of dependent objects must be broken before the DDL command completes. In a single-instance database, breaking parse locks requires changes to local shared pool objects. However, in a RAC database, where the parsed representation of dependent objects can exist in another instance's shared pool, parsed locks must be broken globally. Further, library cache locks are globalized as Global Enqueue Services (GES) locks, so breaking parse locks requires invalidating GES enqueues. This activity triggers a set of coordination messages between the instance background processes.

I enabled 10046 event trace in my session and truncated a table. The following are a few lines from the SQL*Trace file. These wait events are associated with a coordination mechanism between the instances. You can identify the object details as the object_id is listed in the 'local write wait' event. In this example, the object_id of the truncated table is 89956.

```
 Truncate table t1;
...
 nam='enq: RO - fast object reuse' ela= 976 name|mode=1380909062 2=65598 0=2 obj#=22
...
 nam='enq: RO - fast object reuse' ela= 1022 name|mode=1380909062 2=65598 0=1 obj#=22
 nam='enq: RO - fast object reuse' ela= 478 name|mode=1380909057 2=65598 0=2 obj#=22
...
 nam='local write wait' ela= 270 file#=11 block#=280 p3=0 obj#=89956
 nam='local write wait' ela= 1315 file#=11 block#=281 p3=0 obj#=89956
 nam='local write wait' ela= 1702 file#=11 block#=282 p3=0 obj#=89956
 nam='DFS lock handle' ela= 651 type|mode=1128857605 id1=13 id2=5 obj#=89956
 nam='DFS lock handle' ela= 563 type|mode=1128857605 id1=13 id2=1 obj#=89956
 nam='DFS lock handle' ela= 1492 type|mode=1128857605 id1=13 id2=2 obj#=89956
```

For further discussion about the DFS lock handle mechanism, see Chapter 11.

In most cases, developers use TRUNCATE or DROP commands to store the data temporarily while processing the data. Global temporary tables (GTTs) are the preferred method instead of DDL commands. Rows in GTTs are session specific and not visible to any other session. By default, rows in a GTT are thrown away after a commit. You can also choose to create a GTT with `on commit preserve rows` to preserve rows in the GTT across commits.

As GTT is session specific, it does not suffer from the ill effects of DDL commands. The DELETE command on a GTT does not trigger any checkpoint, and there is no need to invalidate parse locks, either. So, code developed using GTTs will scale better in both a single-instance and a RAC database compared to DDL statements.

I am not advocating that you should never use TRUNCATE or DDL statements in your code. On the contrary, be aware of the ill effects of DDL statements in a RAC database and use the method that best suits your requirements. For example, if you must delete all rows from a big table (100K+ rows) infrequently in your code (or *ad hoc*), then you should use TRUNCATE commands instead of DELETE statements.

A common reason developers shy away from using GTT is that the optimizer can choose an inefficient execution plan for SQL statements accessing GTT. It is common for an application to gather schema-level statistics, thereby collecting statistics on GTT as well. The problem is that the statistics collection process has not populated any rows in those GTTs, and therefore zero rows statistics are populated for the GTT. The optimizer uses special considerations for zero rows statistics leading to inefficient execution plans.

One option to resolve the GTT statistics issue is to remove statistics on GTT and lock the statistics. Dynamic sampling will be triggered if there are no statistics on the table. Another option is to collect statistics after populating a representative number of rows in the GTT, and then lock the table statistics. However, this method assumes that all sessions have a uniform number of rows, which may not be entirely accurate. Release 12c introduces a new feature, session-level private statistics for a GTT, to resolve the statistics issue. You can choose to collect either session-level private statistics and/or shared statistics on GTT. If private statistics are available, the optimizer uses private statistics to choose an optimal execution plan. If not, the optimizer uses shared statistics.

Sequence Cache

Sequences provide unique values, not necessarily strict sequential values. Due to a misconception among developers and designers, however, applications often are designed assuming that sequences will provide strict sequential values.

At the first access to a sequence, sequence values are cached in an instance SGA, up to 20 values by default. Subsequent access to that sequence will retrieve values from the cache until the cache is depleted. After the exhaustion of cached values, the next access to the sequence will cache 20 more values in the instance SGA. The data dictionary table seq$ keeps permanent record of the highest cached value for every sequence in the database. The replenishing of the sequence cache will update the seq$ table, marking a new highest cached value for that sequence. In a single-instance database, updates to a dictionary table require a lock on a row cache.

The problem with the caching sequence value is that in case of instance failure, cached values are permanently lost. For example, consider a sequence with a next value of 1,014 and the maximum cached value of 1,021 recorded in a seq$ table. In an instance crash, values from 1,014 to 1,021 will be permanently lost. If the application code assumes that sequence values will be in a strict sequential order, the loss of cached values can lead to unnecessary application data issues. Also, note that losing sequence values does not imply a data loss scenario, only that cached sequence values are permanently lost. Application designers alter attributes of sequences to ORDER, NOCACHE to impose strict sequential order. The ORDER and NOCACHE attributes have adverse effects if the sequences are accessed very frequently. Every access to the sequence forces an update to the seq$ table. As updates to the seq$ table are performed under the protection of a row cache lock, if many sessions are trying to access sequence concurrently, then the `row cache lock` event can be seen as a major wait event.

■ **Note** Updates to the seq$ table are not the only cause of row cache lock event waits. Further review of details is warranted to identify the root cause. Chapter 11 discusses row cache locks further.

The following few lines from a SQL trace shows that the seq$ table is updated for each access to the sequence if the attributes of the sequence are set to ORDER, NOCACHE. The wait for row cache lock event is also shown in the following.

```
update seq$ set increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,cache=:7,highwater=:8,
audit$=:9,flags=:10,partcount=:11 where obj#=:1

nam='row cache lock' ela= 101 cache id=13 mode=0 request=5 obj#=89956 tim=1364157893629929
```

In the preceding wait line, cache_id is set to 13. Querying v$rowcache view, you can identify the type of rowcache entry. In this example, the update to the seq$ table is the reason to lock a row cache.

```
select type, parameter from v$rowcache where cache#=13;

TYPE        PARAMETER
----------- --------------------------------
PARENT      dc_sequences
```

The lines from a tkprof output of a SQL trace file (see Listing 6-1) show the impact of uncached sequences in a RAC database. About 255 seconds are spent on row cache lock waits in a total run time of 282 seconds. Note that there are no waits for the DFS lock handle event since the sequence attribute is set to NOCACHE. This difference will be relevant when I discuss cached sequences later in this section.

Listing 6-1. Performance of Uncached Sequence

```
INSERT INTO RS.T_GEN_SEQ_02
VALUES
 ( RS.T_GEN_SEQ_02_SEQ.NEXTVAL, LPAD ('Gen',25,'DEADBEEF')

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute   5001      7.71     282.75          3        333      20670        5001
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     5002      7.71     282.75          3        333      20670        5001
```

```
 Event waited on                          Times   Max. Wait  Total Waited
  ---------------------------------------  Waited  ----------  ------------
  row cache lock                            4586        0.76        255.01
  Disk file operations I/O                     7        0.00          0.00
  db file sequential read                      3        0.01          0.03
  gc current block busy                     1064        0.46          7.08
  gc current block 2-way                    2660        0.05          3.36
```

In a RAC database, sequence values can be cached in all instances. This sequence caching leads to retrieved sequence values not correlating with a time component. Consider the following scenario:

1. Session 1 connects to PROD1 instance and accesses a sequence emp_id_seq, retrieving a value of 1,022. Values in the range of 1,022 to 1,041 will be cached in PROD1 SGA for that sequence, assuming a default cache of 20.
2. Another session connecting to PROD2 will retrieve a value of 1,042 from that sequence. Values in the range of 1,042 to 1,061 will be cached in the PROD2 instance SGA.
3. If session 1 connected to the PROD1 instance accesses the sequence again, a value of 1,023 will be returned.

So, the retrieved sequence values sorted by time will be 1,022, 1042, and 1,023, clearly not correlating with a time component.

In a RAC database, if the attributes of a frequently accessed sequence are set to ORDER, NOCACHE, performance problems will be magnified. As uncached sequences trigger an update to a seq$ table for every access, row cache locks are acquired aggressively. In a RAC database, row cache locks are globalized as GES layer locks, so additional GES locks must be acquired before updating dictionary tables. Essentially, a magnification effect comes into play, and a huge amount of waiting for GES locks and row cache lock wait events will be the result.

If you must retrieve sequence values in a strict sequential order, one option is to use ORDER, CACHE attributes of sequences. In RAC, a GES resource is used to maintain the current maximum cached value of a sequence, reducing updates to a dictionary table. Using the attributes ORDER and CACHE is a much better option if you must retrieve values in a strict sequential order. Still, if the resource master instance of the sequence crashes, then there is a possibility of losing cached sequence values. Note that an excessive amount of access to sequences with ORDER and CACHE attributes results in longer waits for the DFS lock handle event.

The test case in Listing 6-1 was repeated with sequence attributes set to ORDER, CACHE 20. The lines from the tkprof output file in Figure 6-2 show that the insert statement completed in about 12 seconds compared to 282 seconds in the case of uncached sequences test case in Listing 6-2.

Listing 6-2. Sequence with ORDER and CACHE

```
INSERT INTO RS.T_GEN_SEQ_02
VALUES
 ( RS.T_GEN_SEQ_02_SEQ.NEXTVAL, LPAD ('Gen',25,'DEADBEEF')

call     count       cpu    elapsed       disk      query    current        rowsf
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.01          0          0          0           0
Execute   5001      0.94      12.60          0        910      16440        5001
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     5002      0.94      12.62          0        910      16440        5001
```

```
Event waited on                          Times   Max. Wait  Total Waited
 ---------------------------------------  Waited  ----------  ------------
 DFS lock handle                            359        0.05          0.64
 enq: HW - contention                         6        0.03          0.09
 buffer busy waits                          130        0.06          0.50
```

Our recommendations for a sequence in a RAC database are as follows:

1. If sequences are heavily accessed, then use a bigger cache value (1,000+). A bigger cache improves performance because access to a cached value is an order of magnitude faster than access to an uncached sequence value. The only drawback is that an instance crash can lose all cached sequence values.
2. If the sequences are accessed infrequently, use the default cache value of 20.
3. If you must retrieve values sequentially for *infrequently* accessed sequences, alter the sequence attributes to `ORDER`, `CACHE` value to avoid ill effects of uncached sequences in a RAC database. Also, alter the cache value of the sequence to 20. This option has less impact than sequences with nocache attributes because sequence values are stored in a GES resource.
4. If you cannot afford to lose any sequence value, then you should consider *only* `ORDER`, `NOCACHE` attributes. You should know that if you alter index attributes to `ORDER`, `NOCACHE` for frequently accessed sequences, it could lead to a hung cluster.

Freelists and ASSM

A freelist maintains a list of blocks available with sufficient free space in a non-ASSM (Automatic Segment Space Management) tablespace. Sessions search freelists while searching for a free block to populate a row piece. Freelists are stored in the segment header block, and if there is a need for excessive concurrent access to freelists, then the segment header block can be a hot block. The segment header block must be transferred between the instances repetitively, causing block contention with possible waits for gc buffer busy events.[4]

You can alleviate freelist block contention (segment header contention) by creating objects with multiple freelist groups. Each freelist group is stored in a block by itself. Further, instances have an affinity to a freelist group and sessions in an instance search in the freelist group with affinity, before searching in another freelist group. So, segment header contention can be alleviated by creating as many freelist groups as instances; that is, if you have four instances in a database, then create objects with a freelist group of 4.

Note that freelists and freelist groups are applicable only to Manual Segment Space Management (MSSM) tablespaces. ASSM tablespaces do not suffer from the contention issue because bitmaps at different levels are used to maintain lists of free blocks. Each instance has affinity to an L1 bitmap, essentially implementing a concept similar to multiple freelist groups.

My recommendation is to avoid using MSSM tablespaces and instead use ASSM tablespaces. You should consider partitioning the table if ASSM is not able to support the concurrency.

[4]Note that this is not a complete description of freelist management. There are actually three freelists: master freelist, transaction freelist, and process freelist. The freelist search algorithm is complicated. Please refer to a MOS paper written by Stephen Heisley titled “Freelist Management.”

Excessive Commits

Executing a COMMIT statement triggers an action to the LGWR (Log writer) background process to write the contents of the log buffer to the log file members. After completion of writing to log file members, LGWR posts waiting foreground (server) processes, marking a successful commit. While the LGWR is performing a log flush, a foreground process accounts wait time to a log file sync wait event. In a single-instance database, excessive commits can overload LGWR.

In a RAC database, CURRENT mode block transfers (and some Consistent Read mode block transfers, too) require log flush, a scheme similar to commit processing. LMS processes must wait for LGWR process to complete log flush before sending blocks to remote instance processes. The LMS process is instrumented to account the wait time for the `gcs log flush sync` event.

The requirement for a log flush is due to the consistency mechanism in the database. Let me explain what would happen if the log flush is not performed for a CURRENT mode transfer. Consider a scenario in which session #1 updated a block in a PROD1 instance but has not committed the changes yet. Session #2 connecting to PROD2 is requesting to update the same block. So, the LMS process will transfer the block from the PROD1 instance to the PROD2 instance in CURRENT mode. Assume the PROD1 instance crashes immediately after the block is transferred to PROD2. Session #1 in PROD1 has a pending transaction that must be rolled back, but the log flush did not happen, so the transaction changes are not recorded permanently in the redo log file. Changes made by session #1 would have been transient and discarded if this is a single-instance database. But the block is transferred to the PROD2 buffer cache with the session #1 changes. Essentially, uncommitted changes are permanent in the block, leading to a block corruption. That's the reason the current mode block transfer requiring a log flush. So, if a block has a change SCN higher than the current commit SCN, then a log flush is triggered before a block is transferred to another node.

A CR block fabrication requires redo change vectors from undo records to be applied to roll back the changes, to create a block consistent with the requested block version. Applying change vectors causes redo generation, and therefore redo records are copied into a log buffer. Hence, if the LMS process generated a redo to construct a CR block copy, then that CR block transfer would require a log flush. However, an underscore parameter controls this behavior, and there is absolutely no reason to consider the underscore parameter as that parameter change can cause data consistency issues.

Delay in LGWR processing can induce delays in LMS processes, leading to a longer wait time for global cache events in other nodes. Other processes trying to access the block in transit will wait for gc buffer busy wait events, leading to a multilayered performance issue.

■ **Note** Performance issues in log file parallel writes can induce massive waits in LMS processing, too. Global cache transfers can be stuck if LGWR is not able to complete log flush faster. LMS processes will be accumulating higher wait time for the gcs log flush sync wait event. It is critically important for LGWR and log file writes to be highly efficient to maintain faster global cache transfers.

There is another inherent danger with excessive commits, which is a phenomenon known as *commit cleanout*. When a transaction commits, not all transactions clean the entries in the Interested Transactions List (ITL) section of a block header. Some transactions do not clean up ITL entries.[5] A subsequent session visiting that data block must clean up the ITL entry by visiting an undo header block to determine the transaction status. In a RAC database, an original transaction might have been initiated in another instance, so commit cleanouts can trigger undo header blocks to be transferred from another instance. As undo header blocks and undo blocks are usually hot blocks, excessive transfer leads to higher global cache workload and downstream performance issues.

Our recommendation is to avoid excessive commits. Avoid autocommit features in various languages. Commit only on logical transaction boundaries.

---

[5]There is no danger to transaction integrity because the transaction table in the undo header block maintains the true status of a transaction.

Long Pending Transactions

Long pending transactions on critical tables can trigger massive amounts of undo construction, too. In an Oracle database, a SQL statement cannot see changes from uncommitted transactions created by another session. SQL statements create a read-consistent view of a block[6] by applying undo records from uncommitted transactions. Further, every session accessing the block must build a read-consistent view of the block.

In a single-instance database, transactions are initiated only in a local instance, so the undo application is the only additional workload. In a RAC database, transactions can be initiated in any instance, so the undo header blocks and undo blocks must be transferred between the instances if the data block is accessed in another instance. This transfer will result in an additional global cache workload. If there is a long pending transaction on a heavily accessed table, this problem can quickly escalate to a cluster hung issue.

Further, blocks may not be transferred immediately to another instance if the block is considered busy. A delay in gc transfer can induce further waits for the gc buffer busy event.

Our recommendations are as follows:

1. Defer long, intrusive transactions on critical tables for a low-activity period. This strategy will reduce the severity of the issue in a RAC database.

2. Update only required columns. Developers write code to delete and reinsert rows instead of updates. Not only is this delete + insert strategy unnecessary, it is redo intensive, which results in a higher number of undo records to be applied for read- consistent block fabrication.

Localized Access

Small sets of blocks accessed from all instances can cause contention-related performance issues. It is a common requirement in many applications to query a request table to identify work items to be processed. For example, manufacturing applications typically query a request table to identify the next work item to process. After processing the work item, the row is updated, marking the work item with a completion status. Typically, these work items are picked up by many such concurrent processes, resulting in concurrent read and write on a few blocks of tables and indexes. The concurrent processing layer in an E-Business suite application also uses a similar strategy to execute the next concurrent request.

The performance issue is exacerbated as new rows are inserted into the table concurrently, too. The following pseudocode explains the application code. Worker programs constantly query the request_table to identify new work items to process. At the end of the processing, rows are updated, marking the work item to a completion status.

```
Loop
        Select request_id into v_req from request_table Where status='P'
        Order by request_id
        For update of request_id skip locked;
        --  Process row
        --  update row mark to C
        Update request_table set  status='C' where request_id=v_req;
End loop;
```

[6]A new feature, _undo_cr, allows construction of a read-consistent row, but it is applicable only to special cases such as accessing a row through a unique index, and so on.

There are a few issues with this type of application design:

1. It is typical of these SQL statements to use index-based access paths. As many processes will be concurrently accessing these index blocks, contention will be centered at fewer index blocks.
2. Also, concurrent processes are interested in recently inserted rows and thus contention will be concentrated at a few table/index blocks.
3. Contention is magnified because cardinality of these indexed columns, such as status column, is very low. Therefore, concurrent worker processes compete against each other, accessing the same block aggressively. In a single-instance database, excessive localized access leads to waits for various events such as buffer busy waits, ITL contention, and so on.

   In a RAC database, worker processes might be connected to all instances. The data blocks of table and indexes are transferred between instances, thereby causing severe performance issues. Numerous waits for events such as gc buffer busy acquire or gc buffer busy release will be visible in the database during the occurrence of this performance issue.
4. If the block is busy undergoing changes, then the global cache block transfer can be delayed, resulting in waits for global cache events in other instances. This delay is controlled by the _gc_defer_time parameter, which defaults to 3 centiseconds.
5. Also, because the rows are locked for update, ITL entries in the block header will show an active transaction. Other processes accessing those blocks must determine the transaction status by accessing the transaction table in the undo header block. Because the original transaction might have been initiated in another instance, the undo header and undo blocks from other instances might need to be transferred. This excessive block transfer leads to complex performance issues.

Therefore, performance problems can be encountered in many layers such as contention in table blocks, index blocks, undo header, and undo block. There are few options to resolve this design issue:

1. If the application performs an excessive amount of localized access, then you should consider applying affinity and keep all workers in the same instance. While this reduces the severity of the issue, it does not completely resolve the issue.
2. Redesign application tables to use hash-partitioned tables and hash-partitioned indexes. Modify the query and execution plan to access local hash partitions.
3. Use the Advanced Queuing feature with multiple queues such that the worker processes read a smaller set of data.
4. Design application so that one process assigns work items to the workers. Worker processes will perform a work item and update the row to completion status.

Small Table Updates

Frequently updated smaller tables can induce block contention. An example is an application that uses a table to keep track of maximum column value instead of using a sequence. The following pseudocode explains this coding practice. The maximum value of the employee_id column is tracked in the emp_seq_table. The query retrieves the max_employee column value, inserts rows into employees table with that value, and then updates the max_employee column, adding one to the value. If this code is executed aggressively, then there can be excessive contention on the emp_seq_table:

```
--Retrieve maximum value;
Select max_employee_id into v_employee_id
from emp_seq_table where column_name='EMPLOYEE_ID';
```

```
-- Insert;
insert into employees values (v_employee_id,name, ...);

-- Update max value;
Update emp_seq_table  set max_employee_id = max_employee_id +1
where column_name='EMPLOYEE_ID';
commit;
```

This is an inefficient application design. In a single-instance database, this design will lead to contention-related wait events such as buffer busy waits and ITL waits. In a RAC database, the application will suffer from more severe RAC-related wait events such as gc buffer busy waits and ITL contention. Further, that block will be transferred aggressively between the instances, increasing program runtime. Use sequences to generate ascending or descending values.

Index Design

Keep fewer indexes on critical, DML-intensive tables. Rows can be populated in any block of a heap table as long as there is sufficient free space in the block. However, index entries must be populated in a specific index block. If there are numerous indexes on a table, and if sessions modify the table from all instances, then index blocks can become hot blocks, leading to aggressive global cache activity. More indexes on a table increase the number of index blocks to modify during inserts/updates to indexed columns. This increased block activity directly translates to an increase in global cache activity.

Bitmap indexes are suitable for read-mostly tables. Excessive DML activity on tables with bitmap indexes increases the size of bitmap indexes sharply and induces row-level locking contention. In a RAC database, bitmap index blocks are transferred between instances, exacerbating the symptoms further.

In addition, the compressed index reduces the number of index blocks. This reduction is achieved by not storing repetitive values. Because the size of the compressed index is smaller than the uncompressed index, the number of buffers needed to support the indexes decreases, thereby reducing global cache activity.

The following best-practice guidelines for indexing design are applicable to both single-instance databases and RAC databases:

1. Reduce the number of indexes on DML-intensive tables. In many cases, a number of indexes can be reduced by rearranging the column in a specific order that matches the application requirement and by modifying the query slightly to match the index design.
2. Avoid adding bitmap indexes on heavily modified tables. Performance issues will be localized to a few components in a single-instance database. However, in a RAC database this performance issue is magnified. I have personally witnessed complete application downtime due to a bitmap index on a DML-intensive table.
3. Consider compressed indexes if the cardinality of the leading column is lower.

Inefficient Execution Plan

Inefficient SQL execution plans (as a result of poorly written SQL statements or invalid statistics) can lead to excessive amounts of logical reads, translating to excessive buffer cache access and excessive physical reads. In a single-instance database, concurrent processes executing these inefficient statements can induce higher physical reads, waits for cache buffers chain latch contention, and so on.

In a RAC database, excessive logical reads can lead to increased global cache activity. Numerous blocks might need to be transferred back and forth between the instances in a busy database. This increased global cache activity will slow down the processes, executing inefficient statements, and also cause performance issues with other application components. SQL statements performing nested loops join with millions of rows in the driving table usually suffer from this type of performance issue. Due to incorrect cardinality estimates, the optimizer might choose nested loops join instead of hash join, leading to numerous lookups of other tables.

Again, RAC acts as a problem magnifier and magnifies performance issues associated with inefficient execution plans. A reliable method of statistics collection is an important part of an application lifecycle in a RAC database. A few customer sites collect statistics on a cloned copy of the production database and import the statistics to a production database, avoiding downtime due to invalid statistics.

Excessive Parallel Scans

Excessive inter-instance parallel scanning can overload the interconnect. In an inter-instance parallel operation, messages between parallel servers are transmitted over the interconnect. If the interconnect is not sized properly, then it is possible to induce global cache latency in the database due to overloaded interconnect.

Our recommendation is that if your application is designed to use parallelism aggressively, measure the private network traffic for PX traffic carefully and size the interconnect hardware to match the workload. For further discussion about parallelism in a RAC database, see Chapter 12.

Also, most parallel scans read blocks directly from disk to PGA. These direct reads trigger object-level checkpoints, causing performance issues. For smaller read-mostly tables, caching an entire table in a KEEP buffer cache and performing full scans might be more efficient.

Further, you can reduce the interconnect traffic for PX messages by localizing all PX servers to the local node using the parallel_force_local parameter. If the parameter is set to TRUE, then all parallel executions initiated from the local node will allocate PX servers from the local node only, avoiding PX interconnect traffic.

Full Table Scans

Full table scans can utilize direct path or conventional mode reads. With conventional mode read, blocks are read into the buffer cache from the disk. Reading the block into buffer cache requires global cache locks.[7] This additional workload affects the application performance in a RAC database. However, blocks read for a full table scan operation are considered to be cold blocks and can be paged out of the buffer cache quickly. Hence, gains from the shared buffer cache are minimal, even if many concurrent processes are scanning the table. To reread the block into the buffer cache, global cache locks must be acquired again. Concurrent sessions from multiple instances compete for the same buffers, increasing waits for global cache events and global cache block transfers. This problem is more acute if the buffer cache is undersized.

A table can be scanned using the direct path read access path too, even if SQL executes serially. This feature is known as *serial direct read*. However, a direct path read access path requires an object-level checkpoint similar to the object checkpoint triggered for parallel SQL statements. Thus, excessive serial direct reads can be cost prohibitive if the buffer cache is very active and large. Events such as write complete waits and DFS lock handle can be seen as major wait events in the database.

With an effective table partitioning scheme, you can reduce the number of blocks to be read from the disk. For example, if the table is accessed using a time component, then partition the table along the natural time order. However, a SQL statement should be written matching the partitioning method. With partition pruning, the full table scan access path will scan only a smaller set of partitions, reducing disk reads and global cache locks.

[7]The dynamic remastering feature is useful. If the table is remastered to a local instance, then the need for additional global cache locks is reduced significantly.

Of course, if you must do full table scans on huge tables, you should use parallelism to improve the performance of those queries.

Application Affinity

Application affinity is an important tool to combat performance impact due to global cache workload delays. Essentially, if an application component (such as multiple threads of a batch process) accesses a few objects aggressively, then modify that application component to connect to one instance. Access to the local buffer cache is an order of magnitude faster than the remote buffer cache access, so application components accessing a few objects aggressively will perform more efficiently if the blocks are cached to the local instance.

An effective application affinity scheme should translate to physical segment-level affinity, not just logical-level separation. For example, a client designed a batch process to execute in multiple instances, with each instance processing data for a disjoint list of organizations; that is, organization 1 to 100 was processed by PROD1, 101 to 200 was processed by PROD2, and so on. Although the batch program logically divides the data processing among multiple instances, this logical separation did not translate to the segment level. All threads of the batch program were accessing the same segments and transferring blocks aggressively between the instances, leading to a sharp increase in global cache events. To avoid this problem, partitioning can be used. In this way, list partitioning by organization results in segment-level affinity; batch processes running on each node accessed a disjoint set of partitions, improving batch process performance.

Creating a service is an excellent choice to implement application affinity, too. For example, if you have two major application groups (such as supply chain and manufacturing applications) in the same database, then you can create a service for each application and separate application workloads to different instances. For example, create two services, SCM and MANUFAC. Assign PROD1 as a preferred instance for the SCM service and PROD2 as a preferred instance for the MANUFAC server. With this service design, supply chain tables will be accessed aggressively in the PROD1 instance and the manufacturing tables will be accessed aggressively in the PROD2 instance, improving block collocation. The dynamic remastering feature should also help by remastering almost all supply chain tables to the PROD1 instance and all manufacturing tables to the PROD2 instance, thereby reducing global cache impact further.

If the database server has enough capacity to handle an application component, then keep all application connections connected to a node. Of course, if one node cannot handle the workload, then you should design affinity at the sub-application component level.

Pipes

Database pipes are single-instance objects and do not spawn multiple instances. If your application uses pipes, then the code must be modified to access pipes locally such that all consumers and subscribers of a pipe connect to an instance. Pipes do not work well in a RAC database, so use of the Advanced Queuing feature is recommended.

Application Change Deployment

Application change deployment should be carefully controlled in a busy RAC database. In a single-instance database, if you modify a heavily used package while the sessions are executing the package, it would lead to severe library cache lock and pin waits.

In a RAC database, modifying heavily used packages (or its dependent objects) can lead to global enqueue waits because library cache locks and pins are globalized as enqueues. Sessions can enqueue on global cache locks, leading to an almost hung database. The database is not actually hung, but queuing causes a serialization because each session must acquire the global cache lock, validate the package, and then proceed. Because the sessions are queued

in the global enqueue layer, even if the package is validated by an initial session, other sessions must acquire global locks, verify the package state, and only then proceed further.[8] This behavior affects application availability severely.

Our recommendation is that you avoid deploying application changes during busy hours or perform application rollout in a controlled maintenance window. Even creating an index on a critical table has the ability to cause major application downtime due to globalized library cache locks and pins.

Similarly, massive data changes should be performed during off hours. Excessive amount of undo block access might be required if user sessions try to access modified blocks.

Summary

Better-designed applications that scale well in a single-instance database will scale even better in a RAC database. With a few simple application design strategies, such as objects designed with scalability in mind, optimal use of sequences, and application affinity, you can give your application a scalability boost. We realize there are some potential contradictions in our recommendations; for example, we recommend decreasing the number of indexes in a table and also decreasing logical reads, which might require more indexes. Instead of following our recommendations unquestioningly, use a balanced approach, understand the reasoning behind the recommendation, and then apply the principles to match your scenario. The ultimate goal is to reduce application wait time and increase the scalability of the application.

[8]You can reduce the dependency chain by creating cascading packages. See, for example, *Expert PL/SQL Practices for Oracle Developers and DBAs* (Apress, 2011), specifically Chapter 15 “Dependencies and Invalidations”, by Arup Nanda.

CHAPTER 7

Managing and Optimizing a Complex RAC Environment

by Syed Jaffar Hussain and Tariq Farooq

Managing any Real Application Cluster (RAC) environment, whether on a small or large scale, comes with its fair share of risks and challenges. At a certain point, balancing out the various components and layers involved in setting up a well-oiled, well-tuned Oracle RAC cluster becomes an art involving a lot of skill and a high level of expertise. This chapter gives you a first-hand shot at tackling some of the problems head-on in an effective and practical manner. There may be some overlap between this chapter and other chapters within certain knowledge streams; this is intentional, as some topics have been re-summarized within this chapter to cover the context of the subject at hand.

The overall goal of this chapter is to present you with ideas, tips, tricks, and methodologies available to manage and optimize large-scale cluster environments for cost-effectiveness, better performance, and easy-to-set-up management. It is essential that you carefully study the pros and cons of every point mentioned in this chapter before you decide to apply any changes into your existing environment.

The following topics have been presented in this chapter to optimize any large-scale cluster environments:

- Comparing the pros and cons of shared vs. non-shared Oracle Homes
- Server pools: Concepts and how to best put them to use
- Planning and designing RAC databases for large-scale environments
- Pros and cons of small- and large-scale cluster environments
- Split-brain scenarios and how to avoid them
- Understanding, debugging, and avoiding node evictions
- Extended distance (stretch) clusters: Synopsis, overview, and best practices
- RAC setup and configuration: Considerations and tips for various OS families
- Oracle Universal Installer (OUI): Setup and configuration—Learning the new way of things
- A quick n' easy approach to RAC Database tuning

Shared vs. Non-Shared Oracle Homes

In a typical cluster deployment, small, medium, or large sized, each node must at least consist of a Grid Infrastructure (GI) and RDBMS home, where the Oracle Clusterware, Automatic Storage Management (ASM), and RDBMS software binaries will be installed. Though the storage requirement for an individual home is between 8 GB and 4 GB, it is best advised to have at minimum a 100-GB file system under which the homes will be deployed to sustain future requirements, such as upgrades and patching, and also to maintain an adequate free space for various cluster and database logs.

When you have a complex cluster environment with a large number of nodes, let's say 20 or more nodes, and if the Oracle Homes on each node are configured locally on a local file system, you not only require a large amount of space to house them—say 2 TB for 20 nodes—managing the environment becomes not only complex but also expensive from a storage cost factor perspective.

The administrative complexity and cost can be cut down by sharing the Oracle Homes across nodes in a cluster. Oracle gives the flexibility to have the software installed once in a shared cluster file system, so that it is shared among all other nodes in a RAC cluster. The key advantage of this approach is not only to reduce the overall installation duration of remote copy operation, but also to drastically reduces the storage requirements for other nodes. You can use any Oracle-certified or -supported shared storage to install the Oracle Homes.

During a fresh cluster installation, when a shared storage location is specified to install the software, OUI automatically detects and adjusts to the environment and bypasses the remote node software copy option, speeding up the installation. In addition, this kind of setup requires less storage for the software installation and potentially reduces the duration when you plan to deploy a patch on the existing environment or plann to upgrade the environment. Adding additional nodes also becomes faster when shared Oracle Homes are in place.

The out-of-place upgrade option introduced recently in 11gR2 (11.2.0.2) won't really hamper the current Oracle Homes much, as it doesn't require a prolonged downtime during the course of the upgrade. Whereas, the patch deployment in a shared home requires a complete cluster-wide downtime in contrast to non-shared Oracle Homes.

Regardless of shared or non-shared Oracle Homes, both settings have their share of strengths and weaknesses. The following highlights some pros and cons of shared Oracle Homes:

Pros:

- Less storage requirement
- Oracle Homes backup becomes faster and takes less storage, as it is just a single copy
- Add node (cloning homes) process takes considerably less time
- Patching, upgrade process will become quicker
- All cluster and database logs across nodes can be put under one roof
- Patch consistency will be guaranteed across nodes

Cons:

- Cluster-wide downtime for patching and upgrade
- Rolling upgrades and patching is not possible
- High-level risk of single point of failure
- If the nodes' OS are incompatible, will result in cluster instability

Above all, you can also configure a mixed environment: both shared and non-shared Oracle Homes in a single environment. When you have such mixed environment, on selected nodes you can implement shared Oracle Homes to take advantage of storage and other benefits talked about a little earlier. The following configuration is possible in a single environment:

- A non-shared Grid home among all the nodes
- A shared ORACLE RDBMS HOME among all the nodes
- A shared ORACLE RDBMS HOME for a group of nodes and a non-shared ORACLE RDBMS HOME for the rest of the nodes

Figure 7-1 depicts non-shared, shared, and mixed OH env settings.

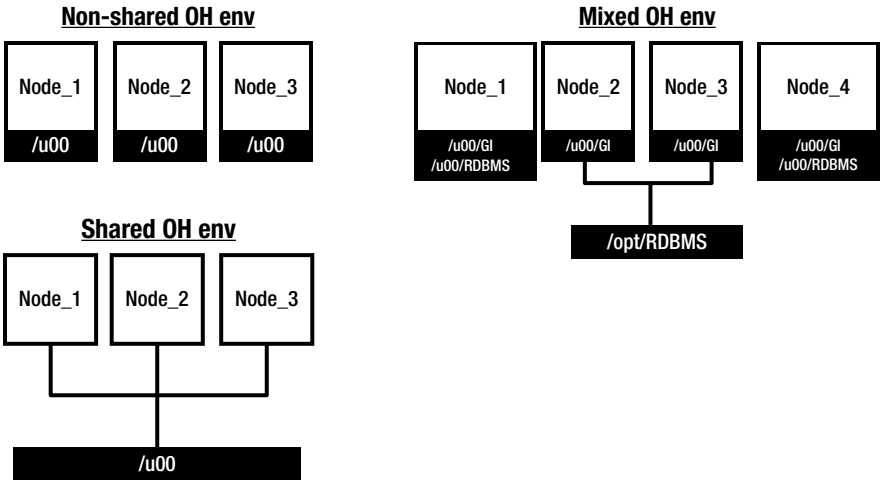


Figure 7-1. *Shared/non-shared/mixed OH environment*

Server Pools

A new feature introduced with 11gR2, server pools, is a dynamic grouping of Oracle RAC instances into abstracted RAC pools, by virtue of which a RAC cluster can be efficiently and effectively partitioned, managed, and optimized. Simply put, they are a logical grouping of server resources in a RAC cluster and enable the automatic on-demand startup and shutdown of RAC instances on various nodes of a RAC cluster.

Server pools generally translate into a hard decoupling of resources within a RAC cluster. Databases must be policy managed, as opposed to admin managed (the older way of RAC server partitioning: pre-11gR2) for them to be organized as/within server pools.

Each cluster-managed database service has a one-to-one relationship with server pools; a DB service cannot be a constituent of multiple server pools at the same time.

server pools and policy management of RAC database instances come in real handy when you have large RAC clusters with a lot of nodes present. However, this technology can be used for policy-based management of smaller clusters as well.

Types of Server Pools

Server pools are classified into two types, both of which are explained in detail as follows:

- System-defined server pools
- User-defined server pools

System-Defined Server Pools

As the name implies, these server pools are created by default automatically with a new installation. Server pools of this type house any/all non-policy-managed databases. System-defined server pools are further broken up into two subcategories:

- GENERIC server pool

In addition to RAC servers assigned to a system-defined server pool, all non-policy-managed (admin-managed) databases are housed within the GENERIC server pool. The attributes of a GENERIC server pool are not modifiable.

- FREE server pool

All unassigned RAC servers automatically get attached to the FREE server pool by default. Common attributes of a server pool (MIN_SIZE, MAX_SIZE, SERVER_NAMES) are not modifiable within the FREE server pool. However, the IMPORTANCE and ACL properties of the FREE server pool can be changed.

User-Defined Server Pools

The attributes of user-defined server pools are listed as follows. This methodology is termed "server categorization" by Oracle.

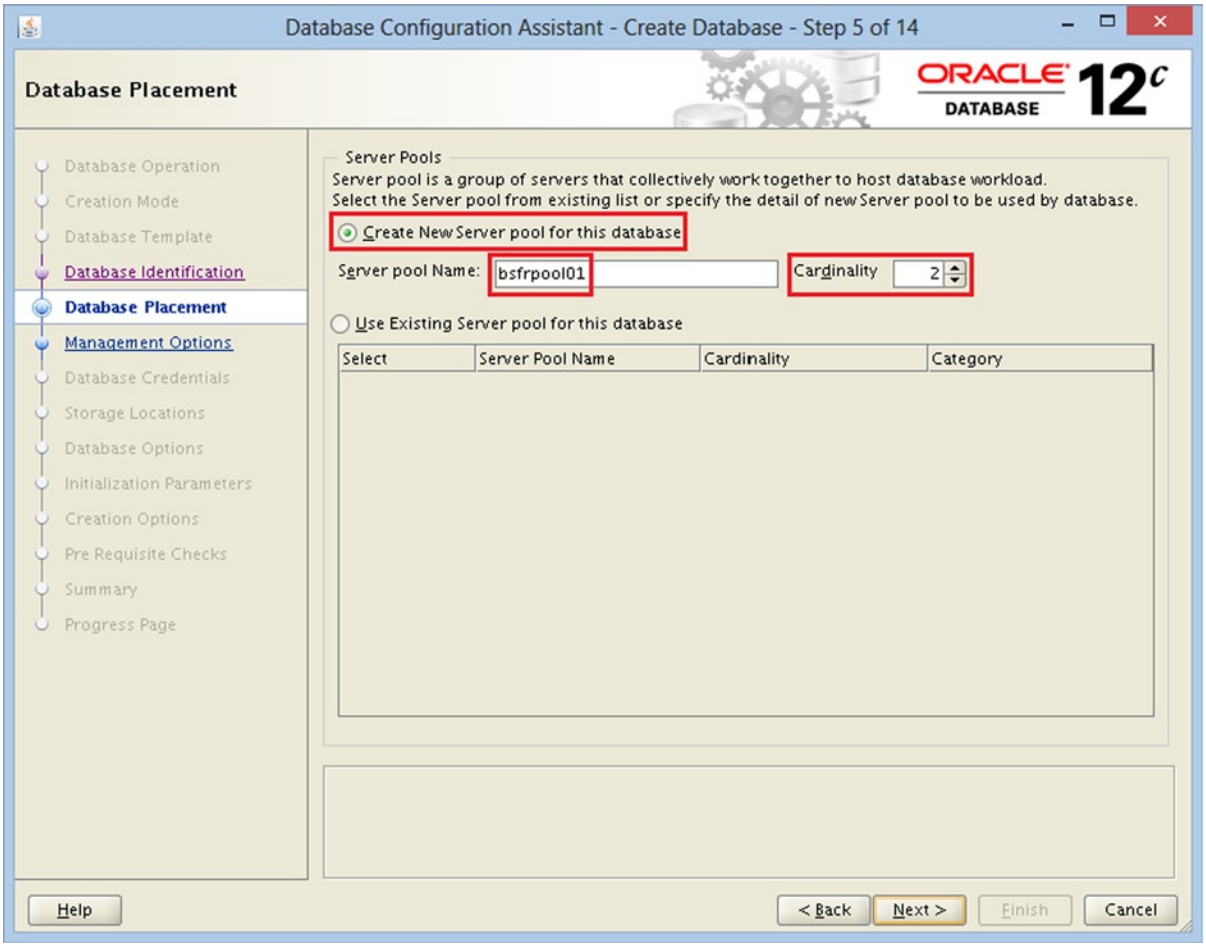
- **NAME:** Define the name of the server pool with this attribute
- **IMPORTANCE:** Comparative importance of server pool. Values allowed in order of importance: 0-1,000
- **MIN_SIZE:** Minimum no. of nodes within a server pool. Default value is 0
- **MAX_SIZE:** Maximum no. of nodes within a server pool. A value of -1 defines a cluster-wide server pool
- **ACL:** Access Control List containing security/users/privileges information about the server pool
- **SERVER_CATEGORY:** Defines server category; this is an attribute mutually exclusive to the SERVER_NAMES attribute; only one can be set at a time (not both)
- **SERVER_NAMES:** A hard coupling of RAC servers to the server pool. This is an attribute mutually exclusive to the SERVER_CATEGORY attribute; only one can be set at a time (not both)
- **ACTIVE_SERVERS:** This is an auto-managed attribute and changes in real time
- **SERVERPOOL:** Name of the server pool
- **EXCLUSIVE_POOLS:** Defines whether the RAC servers are shareable among server pools or not
- **PARENT_POOLS:** Specifies if a server pool is a child of another pool (nested pools)

Creating and Managing Server Pools

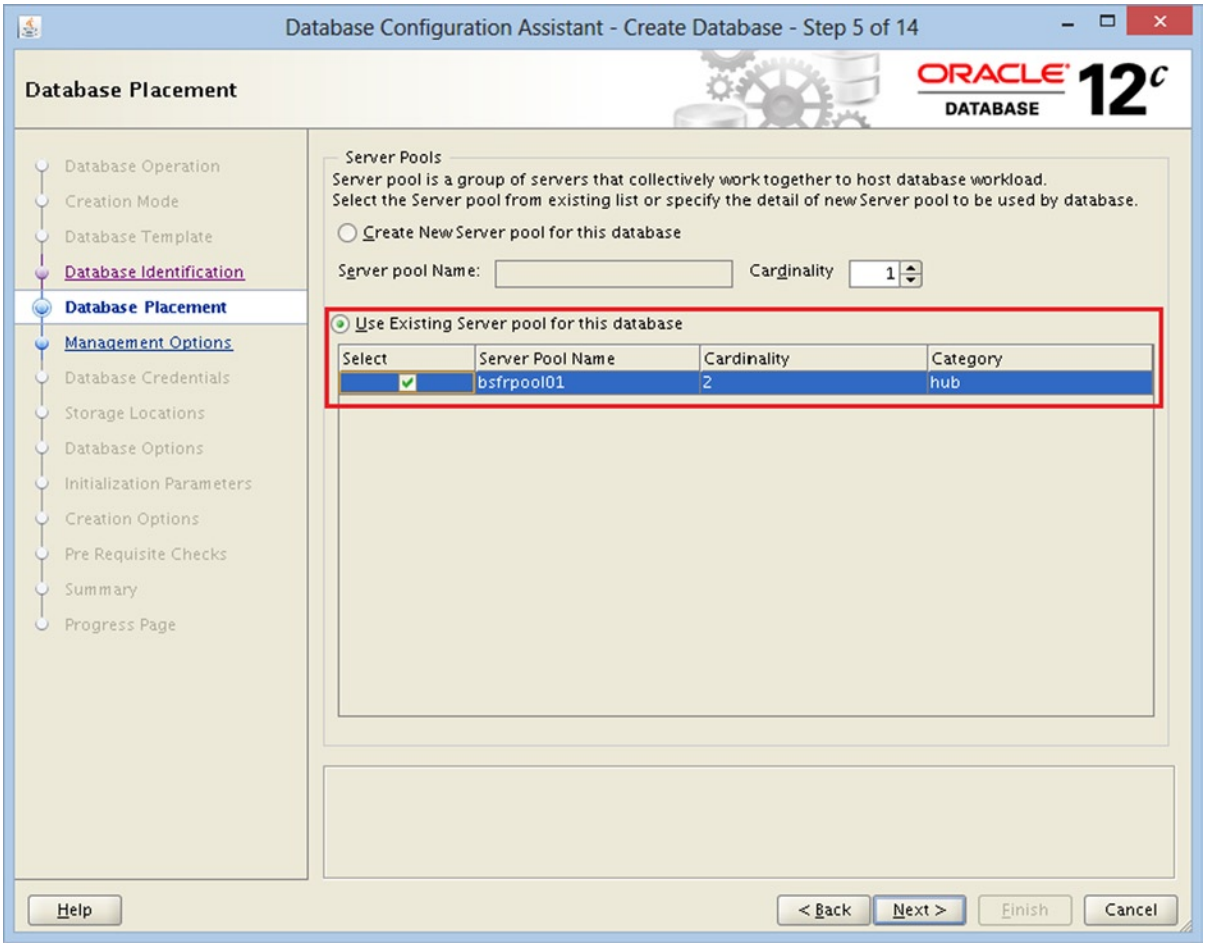
Server pools (with RAC servers attached to them) can easily be defined at database creation time within the Database Configuration Assistant (DBCA) as well as with the SRVCTL and CRSCTL command-line utilities. Additionally, they can be created/managed from Oracle Enterprise Manager (recommended approach).

Some examples of command-line and GUI tool options for adding and managing server pools are listed as follows.

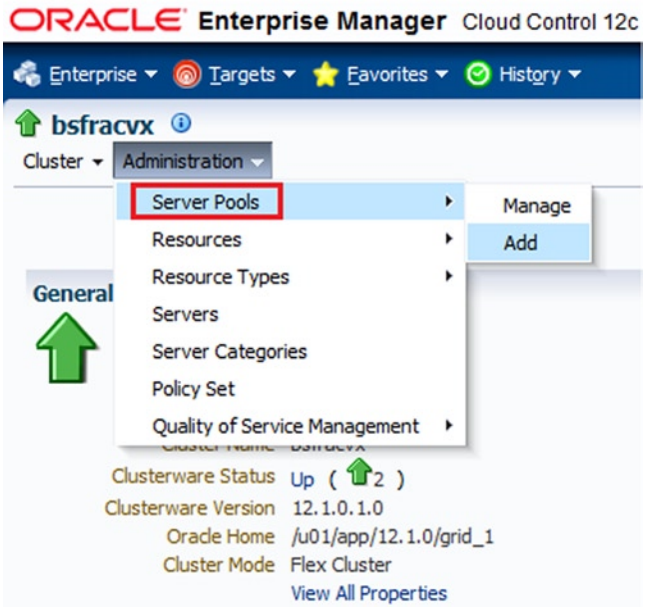
```
$ crsctl add serverpool bsfsrvpool01 -attr MIN_SIZE=1, MAX_SIZE=2, IMPORTANCE=100
$ crsctl delete serverpool
$ crsctl status serverpool -f
$ srvctl config serverpool
```



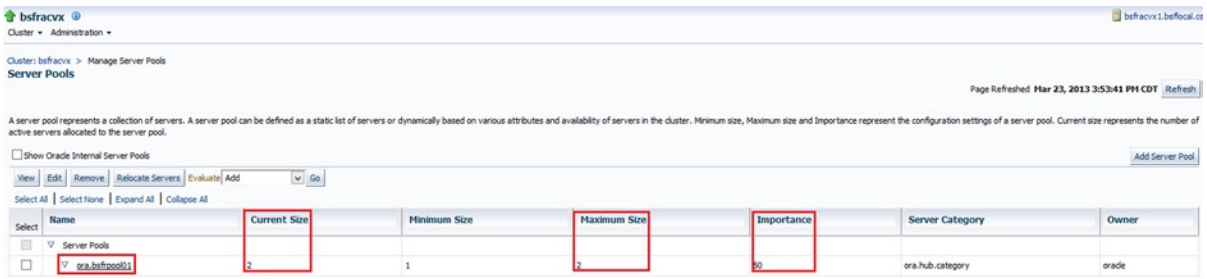
Figures 7-2. *Examples: Creating/attaching to a server pool using DBCA*



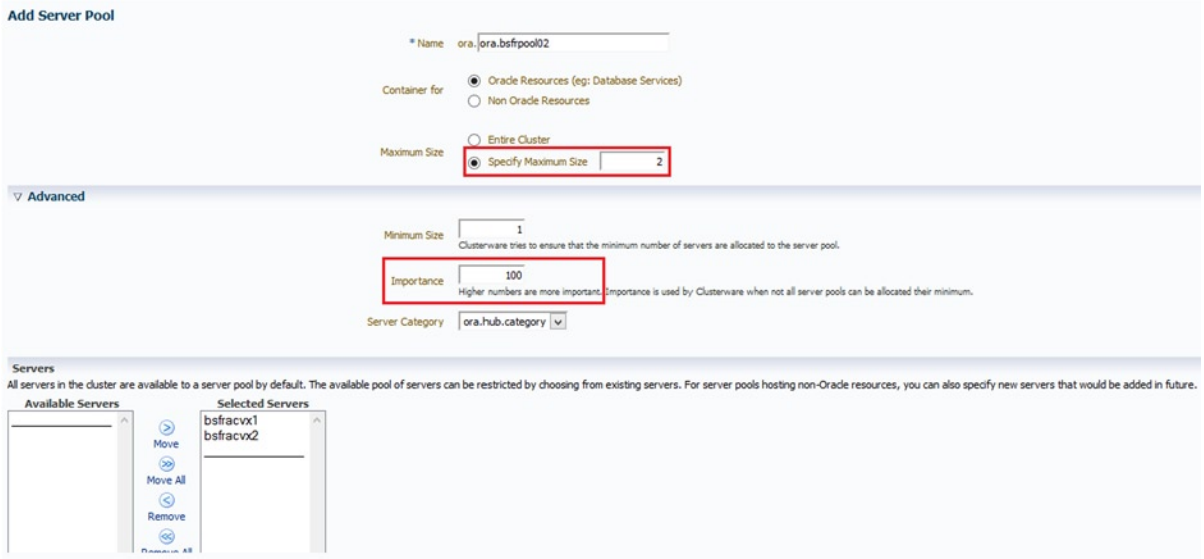
Figures 7-3. *Examples: Creating/attaching to a server pool using DBCA*



Figures 7-4. *Creating and managing server pools within Oracle Enterprise Manager Cloud Control*



Figures 7-5. *Creating and managing server pools within Oracle Enterprise Manager Cloud Control*



Figures 7-6. *Creating and managing server pools within Oracle Enterprise Manager Cloud Control*

Planning and Designing RAC Databases

Before a new RAC database is deployed in a complex/large-scale cluster environment, it is essential that you carefully plan and design the database resources keeping in view the requirements from the application and business points of view. When you plan a new database, you have to consider the core requirements of the application and future business demands, such as increased workload, how to control the resources on the server, etc. When multiple databases are configured on a node, each database has a different requirement for CPU and memory resources. Some databases need more resources, while others need much less. However, there will be no control over the CPU consumption by default to the instances running on the node.

In the subsequent sections, we give you some valuable tips to optimize RAC databases by introducing some of the new features introduced in the recent and current Oracle versions, for example, instance caging and policy-managed database.

Policy-Managed Databases

From 11gR2 onward, you will be given a choice to create two alternative types of RAC database management configuration options during database creation within DBCA: `admin managed` and `policy managed`. Depending on the application and business needs and considering the business workload demands, you can choose the configuration option to create either an `admin-managed` or a `policy-managed` database. The typical admin-managed databases are sort of static and assign to a specific set of nodes in the cluster, their connect configuration as well. You will define your requirements during the course of database or its service creation, which is pinned to particular nodes on which they will be deployed and reside in. Whereas, a policy-based database, on the other hand, is more dynamic in nature and thus will act according to workload requirements. For a policy-managed database, resources are allocated in advance, keeping future requirements in mind, and can be adjusted just-in-time dynamically without much change to the connection configuration.

It is a common factor in a large-scale production RAC setup with a huge number of databases deployed that each database's requirements and use are quite different from those of the other databases in nature. In contrast to a traditional admin-managed database, a policy-managed database has the ability to define the number of instances

and resources needed to support the expected workload in advance. Policy-managed databases are generally useful in a very-large-scale cluster deployment, and the benefits include the following:

- Resources are defined in advance to meet the expected workload requirements.
- When defined with a number, sufficient instances will be initiated to meet the current workload.
- Policy-managed database services are uniform rather than defined in the typical way, which is according to instances preferred and available.
- In general, the objective of a policy-managed database is to remove the hard coding of a service to a specific database instance.

Deploying a Policy-Managed Database

A policy-managed database is configured using the DBCA. At the same time, any preexisting admin-managed database can be effortlessly converted to a policy-managed database in place. Unlike the typical admin-managed database, when you create a new policy-managed database, you will need to specify a server pool on which the instances should be run. You can select a preexisting server pool from the list to configure the RAC database or create a new server pool with the required cardinality (number of servers), as shown in Figure 7-7's `database placement` screenshot.

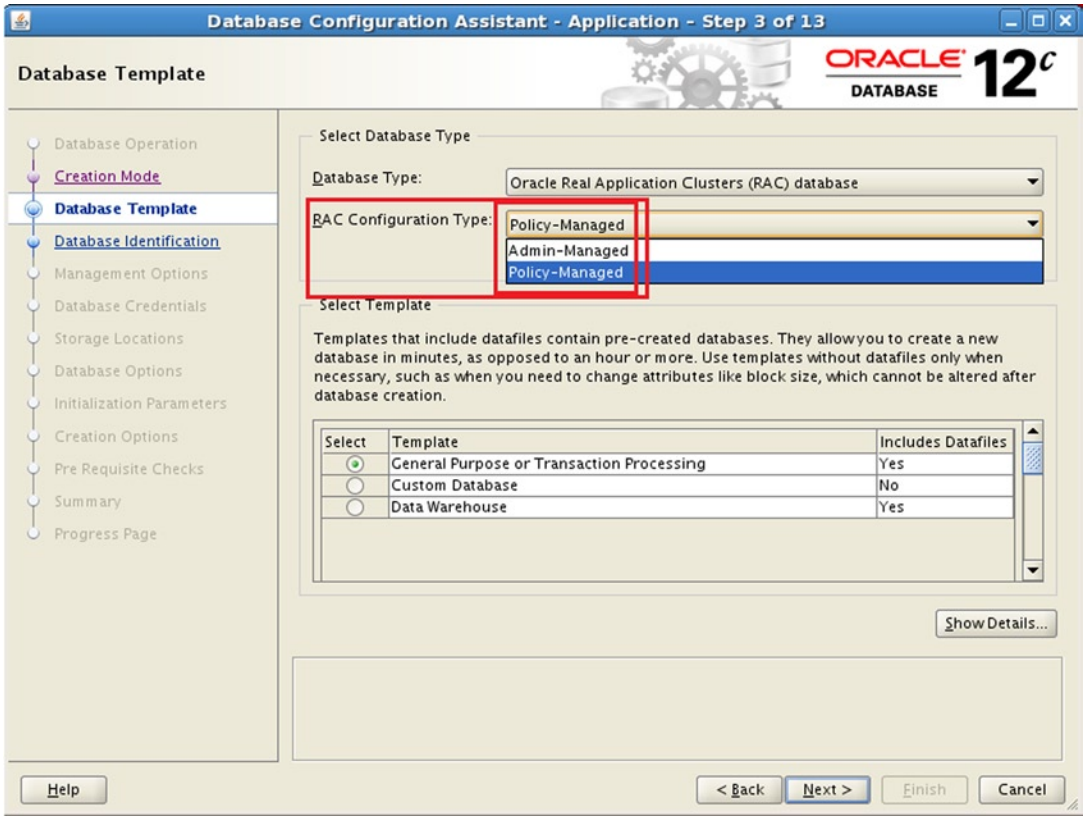


Figure 7-7. *DBCA policy-managed DB configuration*

With regard to the policy-managed database configuration, the RAC configuration type and the selection or creation of a new spool pool is the only different requirement in comparison with a typical admin-based database. The rest of the database creation steps remain unchanged.

Upgrading a Policy-Managed Database

A direct upgrade of a pre-12c admin-managed database to a 12c policy-managed database is not supported; this will have to be a combo upgrade/migration process. The following steps are required:

1. Upgrade the existing pre-12c admin-managed database to 12c.
2. After successful upgrade, migrate the database to policy-managed using the `srvctl modify` command, as demonstrated in the subsequent section.

Migrating an Admin-Managed Database to a Policy-Managed Database

This section provides the essential background required to convert an admin-managed database to a policy-managed database. The following is a practical step-by-step conversion procedure:

1. Verify and save the existing configuration settings of an admin-managed database plus all its services:

```
$ srvctl config database –d <dbname>
$ srvctl status database –d <dbname>
$ srvctl config service –d <dbname>
$ srvctl status service –d <dbname>
```

2. As the GI/cluster administrator (user), create a new server pool specifying MIN/MAX servers to the pool in order to place the policy-managed database. Alternatively, a preexisting/defined server pool can also be used. Use the following example to create a new server pool:

```
$ srvctl add srvpool –serverpool srvpl_pbd –MIN 0 –MAX 4 –servers
node1,node2,node3,node4 - category <name> –verbose

$ srvctl config srvpool–serverpool srvpl_pub–to verify the details
```

3. Stop and migrate the database to the newly created server pool to complete the conversion process:

```
$ srvctl stop database –d <dbname>
$ srvctl modify database –d <dbname> -serverpool srvpl_pbd
```

The preceding srvctl modify command converts an admin-managed to a policy-managed database and places the database into the new policy-managed server pool, named `srvpl_pbd`.

4. Start and verify the database after the conversion using the following examples:

```
$ srvctl start database –d <dbname>
$ srvctl config database –d <dbname>
```

Post-conversion, when you verify the database configuration details, you will notice a change in the services type as "Database is policy managed," instead of "Database is admin managed." Additionally, you will also observe a change, an additional underscore (_) symbol before the instance number, in the instance name in contrast to the previous name.

You also need to run through a set of changes in your environment when the database is managed with OEM Cloud Control 12c containing database services to reflect the change.

5. Since there is a modification in the instance name, you need to rename the existing password file of the database to reflect the new names. Go to the $ORACLE_HOME/dbs directory and use the following example to rename the file:

   ```
   $ mv orapwDBNAME1 orapwDBNAME_1
   ```

6. If the database was registered with OEM Cloud Control 12c, you must change the existing instance name with the new instance name.

7. All admin-managed RAC database-related services are configured defining the PREFERRED and AVAILABLE options, whereas, for policy-managed databases, a service is defined to a database server pool, defining either SINGLETON or UNIFORM options. When a service is configured to UNIFORM option, it will be set to all instance, whereas the SINGLETON option will be set to a single instance. Use the following example to modify an existing database service:

   ```
   $ srvctl modify service -d dbname -s service_name -serverpool srvpl_pdb -cardinality
   UNIFORM|SINGLETON
   ```

■ **Note** We strongly recommend using the SCAN IPs to connect to the database to avoid connection issues when a database is moved between the nodes defined in the server pool.

Instance Caging

When multiple instances are consolidated in a RAC DB node, resource distribution, consumption, and management become a challenge for a DBA/DMA. Each instance typically will have different requirements and behavior during peak and non-peak business timeframes. There is a high probability that a CPU thirsty instance could impact the performance of other instances configured on the same node. As a powerful new feature with Oracle 11gR2, with instance caging, DBAs can significantly simplify and control an instance CPUs/cores overall usage by restricting each instance to a certain number of CPUs. Large-scale RAC environments can greatly benefit with this option.

Deploying Instance Caging

Instance caging can be deployed by setting two database initialization parameters: the cpu_count and the resource manager on the instance. The following procedure explains the practical steps required to enable instance caging on the instance:

```
SQL> ALTER SYSTEM SET CPU_COUNT=2 SCOPE=BOTH SID='INSTANCE_NAME';

SQL> ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = default_plan|myplan SCOPE=BOTH SID='INSTANCE_NAME';
```

The first SQL statement restricts the database instance's CPU resource consumption to 2, irrespective of the number of CPUs present on the local node; the instance CPU resource use will be limited to a maximum of 2 CPU

power. If the database is a policy-managed database and tends to swing between nodes, you need to carefully plan the `cpu_count` value considering the resource availability on those nodes mentioned in the server pool.

The second SQL statement enables the resource manager facility and activates either the `default plan` or a user defined resource plan. An additional background process, `Virtual Scheduler for resource manager (VKRM)`, will be started upon enabling resource manager. To verify whether the `CPU caging` is enabled on an instance, use the following examples:

```
SQL> show parameter cpu_count
SQL> SELECT instance_caging FROM v$rsrc_plan WHERE is_top_plan = 'TRUE';
```

When instance caging is configured and enabled, it will restrict the CPU usage for the individual instances and will not cause excessive CPU usage.

After enabling the feature, the CPU usage can be monitored by querying the `gv$rsrc_consumer_group and gv$rsrcmgrmetric_history` dynamic views. The views will provide useful inputs about the instance caging usage. It provides in minute detail the CPU consumption and throttling feedback for the past hour.

To disable instance caging, you simply need to reset the `cpu_count` parameter to 0 on the instance.

■ **Note** `cpu_count` and `resource_manager_plan` are dynamic initialization parameters, which can be altered without having instance downtime. Individual instances within a RAC database can have different `cpu_count` values. However, frequent modification of the `cpu_count` parameter is not recommended.

Figure 7-8 shows how to partition a 16-CPU/core box (node) using instance caging to limit different instances on the node to a specific amount of CPU usage.

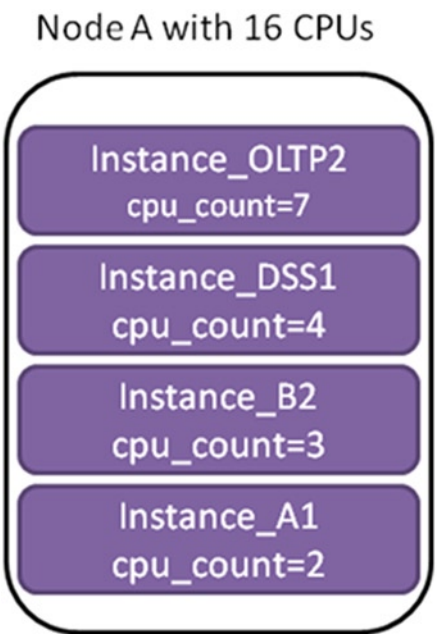


Figure 7-8. *Example instance caging a 16-CPU/core box for consolidated RAC databases*

Small- vs. Large-Scale Cluster Setups

It's an open secret that most IT requirements are typically derived from business needs. To meet business requirements, you will have either small- or large-scale cluster setups of your environments. This part of the chapter will focus and discuss some of the useful comparisons between small- and large-scale cluster setups and also address the complexity involved in large cluster setups.

It is pretty difficult to jump in and say straightaway that whether a small-scale cluster setup is better than a large-scale cluster setup, as the needs of one organization are totally dissimilar to those of another. However, once you thoroughly realize the benefits and risks of the two types of setup, you can then decide which is the best option to proceed with.

Typically, in any cluster setup, there is a huge degree of coordination in the CRS, ASM, and instance processes involved between the nodes.

Typically, a large-scale cluster setup is a complex environment with a lot of resources deployment. In a large-scale cluster setup, the following situations are anticipated::

- When the current environment is configured with too many resources, for example, hundreds of databases listeners, and application services, etc., across the nodes, there is a high probability that it might cause considerable delay starting up the resources automatically on a particular node upon node eviction. Irrespective of the number of resources configured or were running on the local node, on node reboot, it has to scan through the list of resources registered in the cluster, which might cause delay in starting things on the node.
- Sometime it will be time consuming and bit difficult to gather the required information from various logs across all nodes in a cluster when Clusterware related issues confronted.
- Any ASM disk/diskgroup-related activity requires ASM communication and actions across all ASM instances in the cluster.
- If an ASM instance on a particular node suffers any operational issues, other ASM instances across the nodes will be impacted; this might lead to performance degradation, instance crash, etc.
- When shared storage LUNS are prepared, it must be made available across all cluster nodes. For any reason, if one node lacks ownership or permission or couldn't access the LUN(disk), the disk can't be used to add to any diskgroup. It will be pretty difficult to track which node having issues.
- Starting/stopping cluster stack from multiple nodes in parallel will lock GRD across the cluster and might cause an issue bringing up the cluster stack subsequently on the nodes.
- If you don't have an optimal configuration in place for large-scale cluster implementation, frequent node evictions can be anticipated every now and then.
- It is likely to confront the upper limit for the maximum number of ASM diskgroups and disks when a huge number of databases is deployed in the environment.

On the other hand, a smaller cluster setup with a lesser number of nodes will be easy to manage and will have less complexity. If it's possible to have multiple small ranges of cluster setups in contrast to a large-scale setup, this is one of the best options, considering the complexity and effort required for the two types.

Split-Brain Scenarios and How to Avoid Them

Split-brain scenarios are a RAC DBA or DMA's worst nightmare. They are synonymous with a few interchangeable terms: node evictions, fencing, STONITH (Shoot the Node in the Head), etc.

The term split-brain scenario originates from its namesake in the medical world: split-brain syndrome, which is a consequence of the connection between the two hemispheres of the brain being severed, results in significant impairment of normal brain function. Split-brain scenarios in a RAC cluster can be defined as functional overlapping of separate instances, each in its own world without any guaranteed consistency or coordination between them. Basically, communication is lost among the nodes of a cluster, with them being evicted/kicked out of the cluster, resulting in node/instance reboots. This can happen due to a variety of reasons ranging from hardware failure on the private cluster interconnect to nodes becoming unresponsive because of CPU/memory starvation issues.

The next section covers the anatomy of node evictions in detail.

Split-brain situations generally translate into a painful experience for the business and IT organizations, mainly because of the unplanned downtime and the corresponding impact on the business involved. Figure 7-9 shows a typical split-brain scenario:

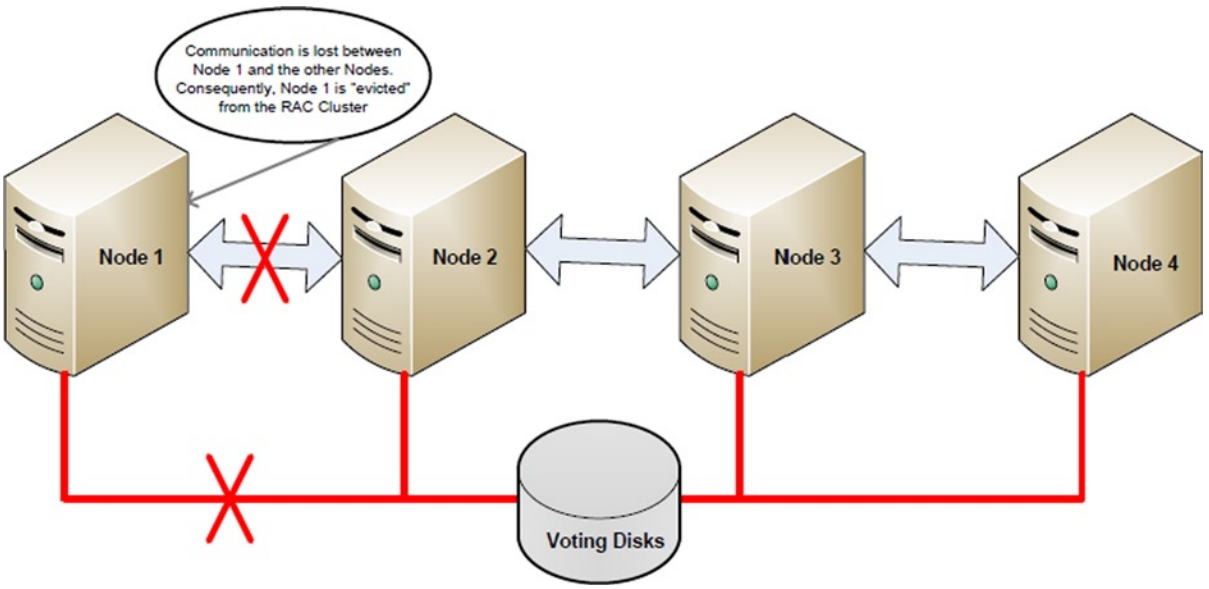


Figure 7-9. *Typical split-brain scenario within an Oracle RAC cluster*

Here are some best practices that you can employ to eliminate or at least mitigate potential split-brain scenarios, which also serve as good performance tuning tips and tricks as well (a well-tuned system has significantly less potential for overall general failure, including split-brain conditions):

- Establish redundancy at the networking tier: redundant switches/Network Interface Cards (NICs) trunked/bonded/teamed together; the failover of the componentry must be thoroughly tested, such that the failover times do not exceed communication thresholds in place for RAC split-brain scenarios to occur.
- Allocate enough CPU for the application workloads and establish limits for CPU consumption. Basically, CPU exhaustion can lead to a point where the node becomes unresponsive to the other nodes of the cluster, resulting in a split-brain scenario leading in turn to node evictions.

This is the most common cause of node evictions. Place a cap to limit CPU consumption and usage through sophisticated and easy-to-use workload management technologies like DBRM, instance caging, cluster-managed database services, etc.

- Allocate enough memory for the various applications and establish limits for memory consumption. Automatic Memory Management comes in handy, as it puts limits on both SGA and Program Global Area (PGA) areas. With 11g, if you are using Automatic Shared Memory Management(ASMM) on HUGEPAGES within the Linux family of OS (AMM is not compatible with HUGEPAGES), this can prove to be a bit of a challenge, especially if you are dealing with applications that tend to be ill-behaved. With 12c, a new parameter called PGA_AGGREGATE_LIMIT has been introduced to rectify this problem, which caps the total amount of PGA that an instance uses.

- Employ/deploy DBRM along with IORM (if you are operating RAC clusters on Exadata). In the absence of resource consumption limits, a single rogue user with one or more runaway queries (typically in the ad-hoc query world) can run away with all your resources, leaving the system starved for CPU/memory in an unresponsive state; This will automatically lead to node evictions/split-brain scenarios occurring repeatedly. After the instances/nodes have been rebooted, the same jobs can queue again with the same behavior being repeated over and over again. This point has been alluded to in the preceding points as well.

- Set up and configure instance caging (CPU_COUNT parameter) for multi-tenant database RAC nodes; monitor and watch out for RESMGR:CPU quantum waits related to instance caging, especially when instances have been overconsolidated, for example, within an Exadata environment. If RESMGR:CPU quantum waits are observed, the dynamic CPU_COUNT parameter can temporarily be increased to relieve the pressure points in a RAC instance, provided enough CPU is available for all the instances within the RAC node.

- Ensure that any kind of antivirus software is *not* active/present on any of the RAC nodes of the cluster. This can interfere with the internal workings of the LMS and other RAC processes and try to block normal activity by them; in turn, this can result in excessive usage of CPU, ultimately being driven all the way to 100% CPU consumption, resulting in RAC nodes being unresponsive and thereby evicted from the cluster.

- Patch to the latest versions of the Oracle Database software. Many bugs have been associated with various versions that are known to cause split-brain scenarios to occur. Staying current with the latest CPU/PSUs is known to mitigate stability/performance issues.

- Avoid allocating/configuring an excessive no. of LMS_PROCESSES. LMS is a CPU-intensive process, and if not configured properly, can cause CPU starvation to occur very rapidly, ultimately resulting in node evictions.

- Partition large objects to reduce I/O and improve overall performance. This eases the load on CPU/memory consumption, resulting in more efficient use of resources, thereby mitigating CPU/memory starvation scenarios that ultimately result in unresponsive nodes.

- Parallelization and AUTO DOP: Set up/configure/tune carefully. Turning on Automatic Degree of Parallelism (AUTO DOP—PARALLEL_DEGREE_POLICY= AUTO) can have negative consequences on RAC performance, especially if the various PARALLEL parameters are not set up and configured properly. For example, an implicit feature of AUTO DOP is in-memory parallel execution, which qualifies large objects for direct path reads, which in the case of ASMM (PGA_AGGREGATE_TARGET) can translate into unlimited use of the PGA, ultimately resulting in memory starvation; nodes then end up being unresponsive and finally get evicted from the cluster. High settings of PARALLEL_MAX_SERVERS can have a very similar effect of memory starvation. The preceding are just a few examples, underscoring the need for careful configuration of PARALLELIZATION init.ora parameters within a RAC cluster.

Understanding, Debugging, and Preventing Node Evictions

Node Evictions—Synopsis and Overview

A node eviction is the mechanism/process (piece of code) designed within the Oracle Clusterware to ensure cluster consistency and maintain overall cluster health by removing the node(s) that either suffers critical issues or doesn't respond to other nodes' requests in the cluster in a timely manner. For example, when a node in the cluster is hung or suffering critical problems, such as network latency or disk latency to maintain the heartbeat rate within the internal timeout value, or if the cluster stack or clusterware is unhealthy, the node will leave the cluster and do a fast self-reboot to ensure overall cluster health. When a node doesn't respond to another node's request in a timely manner, the node will receive a position packet through disk/network with the instruction to leave the cluster by killing itself. When the problematic node reads the position (kill) pockets, it will evict and leave the cluster. Thereby, the evicted node then will then perform a fast reboot to ensure a healthy environment between the nodes in the cluster. A fast reboot generally doesn't wait to flush the pending I/O; therefore, it will be fenced right after the node reboot.

Debugging Node Eviction

Node eviction is indeed one of the worst nightmares of RAC DBAs, posing never-ending challenges on every occurrence. Typically, when you maintain a complex or large-scale cluster setup with a huge number of nodes, frequent node eviction probabilities are inevitable and can be anticipated. Therefore, node eviction is one of the key areas to which you have to pay close attention.

In general, a node eviction means unscheduled server downtime, and an unscheduled downtime could cause service disruption; frequent node eviction will impact an organization's overall reputation too. In this section, we will present you with the most common symptoms that lead to a node eviction, and also what are the crucial cluster log files in context to be verified to analyze/debug the issue to find out the root cause for the node eviction.

Since you have been a cluster administrator for a while, we are pretty sure that you might have confronted a node eviction occurrence at least once in your environment. The generic node eviction information logged in some of the cluster logs sometimes doesn't actually provide a right direction for the actual root cause of the problem. Therefore, you need to gather and refer to various types of log file information from cluster, platforms, OS Watcher files, etc., in order to find the root cause of the issue.

A typical warning message, outlined hereunder, about a particular node being being evicted will be printed in the cluster `alert.log` of a surviving node just few seconds before the node eviction. Though you can't prevent the node eviction happening in such a short span, it will bring the warning message to your attention so that you will be informed about which node is about to leave the cluster:

```
[ohasd(6525)]CRS-8011:reboot advisory message from host: node04, component: cssmonit, with time
stamp: L-2013-03-17-05:24:38.904
[ohasd(6525)]CRS-8013:reboot advisory message text: Rebooting after limit 28348 exceeded; disk
timeout 28348, network timeout 27829,
last heartbeat from CSSD at epoch seconds 1363487050.476, 28427 milliseconds ago based on invariant
clock value of 4262199865
2013-03-17 05:24:53.928
[cssd(7335)]CRS-1612:Network communication with node node04 (04) missing for 50% of timeout
interval. Removal of this node from cluster in 14.859 seconds
2013-03-17 05:25:02.028
[cssd(7335)]CRS-1611:Network communication with node node04 (04) missing for 75% of timeout
interval. Removal of this node from cluster in 6.760 seconds
2013-03-17 05:25:06.068
[cssd(7335)]CRS-1610:Network communication with node node04 (04) missing for 90% of timeout
interval. Removal of this node from cluster in 2.720 seconds
```

```
2013-03-17 05:25:09.842
[cssd(7335)]CRS-1601:CSSD Reconfiguration complete. Active nodes are node01,node02,node03.
2013-03-17 05:37:53.185
[cssd(7335)]CRS-1601:CSSD Reconfiguration complete. Active nodes are node01,node02,node03.
```

The preceding extract is from the node01 `alert.log` file of 4-node cluster setup, which indicates about node04 eviction. You may discover from the preceding input that the first warning appeared when the outgoing node missed 50% of timeout interval, followed by 75 and 90% missing warning messages. The reboot advisory message represents the component details that actually initiated the node eviction. In the preceding text, it was the `cssmmonit` that triggered the node eviction due to lack of memory on the node, pertaining to the example demonstrated over here.

Here is the continuous extract from the `ocssd.log` file from a surviving node, node01, about node04 eviction:

```
2013-03-17 05:24:53.928: [    CSSD][53]clssnmPollingThread: node node04 (04) at 50% heartbeat fatal,
removal in 14.859 seconds
2013-03-17 05:24:53.928: [    CSSD][53]clssnmPollingThread: node node04 (04) is impending reconfig,
flag 461838, misstime 15141
2013-03-17 05:24:53.938: [    CSSD][53]clssnmPollingThread: local diskTimeout set to 27000 ms,
remote disk timeout set to 27000, impending reconfig status(1)
2013-03-17 05:24:53.938: [    CSSD][40]clssnmvDHBValidateNCopy: node 04, node04, has a disk HB,
but no network HB, DHB has rcfg 287, wrtcnt, 77684032, LATS 4262516131, lastSeqNo 0, uniqueness
1356505641, timestamp 1363487079/4262228928
2013-03-17 05:24:53.938: [    CSSD][49]clssnmvDHBValidateNCopy: node 04, node04, has a disk HB,
but no network HB, DHB has rcfg 287, wrtcnt, 77684035, LATS 4262516131, lastSeqNo 0, uniqueness
1356505641, timestamp 1363487079/4262228929
2013-03-17 05:24:54.687: [    CSSD][54]clssnmSendingThread: sending status msg to all nodes
2013-03-17 05:25:02.028: [    CSSD][53]clssnmPollingThread: node node04 (04) at 75% heartbeat fatal,
removal in 6.760 seconds
2013-03-17 05:25:02.767: [    CSSD][54]clssnmSendingThread: sending status msg to all nodes
2013-03-17 05:25:06.068: [    CSSD][53]clssnmPollingThread: node node04 (04) at 90% heartbeat fatal,
removal in 2.720 seconds, seedhbimpd 1
2013-03-17 05:25:06.808: [    CSSD][54]clssnmSendingThread: sending status msg to all nodes
2013-03-17 05:25:06.808: [    CSSD][54]clssnmSendingThread: sent 4 status msgs to all nodes
```

In the preceding, the `clssnmPollingThread` of CSS daemon process thread, whose responsibility is to scan/verify the active nodes members, periodically reports about the node04 missing heartbeat and also represents the timestamp of node eviction.

The following text from the `ocssd.log` on node01 exhibits the details about the node being evicted, node cleanup, node leaving cluster, and rejoining sequence:

```
2013-03-17 05:25:08.807: [    CSSD][53]clssnmPollingThread: Removal started for node node04 (14),
flags 0x70c0e, state 3, wt4c 02013-03-17 05:25:08.807: [CSSD][53]clssnmMarkNodeForRemoval: node 04,
node04 marked for removal
2013-03-17 05:25:08.807: [    CSSD][53]clssnmDiscHelper: node04, node(04) connection failed, endp
(00000000000020c8), probe(0000000000000000), ninf->endp 00000000000020c8
2013-03-17 05:25:08.807: [    CSSD][53]clssnmDiscHelper: node 04 clean up, endp (00000000000020c8),
init state 5, cur state 5
2013-03-17 05:25:08.807: [GIPCXCPT][53] gipcInternalDissociate: obj 600000000246e210
[00000000000020c8] { gipcEndpoint : localAddr '
2013-03-17 05:25:08.821: [    CSSD][1]clssgmCleanupNodeContexts():  cleaning up nodes, rcfg(286)
```

```
2013-03-17 05:25:08.821: [    CSSD][1]clssgmCleanupNodeContexts():  successful cleanup of nodes
rcfg(286)
2013-03-17 05:25:09.724: [    CSSD][56]clssnmDeactivateNode: node 04, state 5
2013-03-17 05:25:09.724: [    CSSD][56]clssnmDeactivateNode: node 04 (node04) left cluster
```

Node Evictions—Top/Common Causes and Factors

The following are only a few of the most common symptoms/factors that lead to node evictions, cluster stack sudden death, reboots, and status going unhealthy:

- Network disruption, latency, or missing network heartbeats
- Delayed or missing disk heartbeats
- Corrupted network packets on the network may also cause CSS reboots on certain platforms
- Slow interconnect or failures
- Known Oracle Clusterware bugs
- Unable to read/write or access the majority of the voting disks (files)
- Lack of sufficient resource (CPU/memory starvation) availability on the node for OS scheduling by key CRS daemon processes
- Manual termination of the critical cluster stack daemon background processes (`css, cssdagent, cssdmonitor`)
- No space left on the device for the GI or /var file system
- Sudden death or hang of CSSD process
- `ORAAGENT/ORAROOTAGENT` excessive resource (CPU, MEMORY, SWAP) consumption resulting in node eviction on specific OS platforms

Gather Crucial Information

Consult/refer to the following various trace/log files and gather crucial information in order to diagnose/identify the real symptoms of node eviction:

- `alert.log`: to determine which process actually caused the reboot, refer to the cluster `alter.log` under `$GI_HOME/log/nodename` location. The alert log provides first-hand information to debug the root cause of the issue. Pay close attention to the component to determine important information. If the component shows `cssmoint` or `cssdagent`, then the node is evicted due to resource unavailability for OS scheduling. Either the CPU was 100% clocked for a long time period or too much swapping/paging took place due to insufficient memory availability. If it shows `cssagent`, then it could be due to network issues
- `ocss.log`: If the node eviction happens due to network failure or latency, or voting disk issues, refer to `ocss.log` file under `$GI_HOME/log/nodename/cssd` location
- `cssmonit/cssagent_nodename.lgl`, depending on the OS you are on, either in `/etc/oracle/lastgasp` or `/var/adm/oracle/lastgasp`
- `oracssdmonitor/oracssdagent_root`, under `$GI_HOME/log/nodename/agent/ohasd` location
- In addition to the preceding, refer to OS-specific logs

- Also, it's important to refer to the OS watcher logs to identify resource consumption on the node during or just before node eviction
- OS Watcher log files—All OS Watcher log files are stored under the archive log directory under OS Watcher directory. It contains significant information about resource utilization, such as CPU, memory, top output, swap (page in/out), etc., information. The statistical information recorded in the log files help in identifying if the system was overloaded or starving for resources (CPU, memory)

Rebootless Stop/Fencing

With the incorporation of `rebootless fencing` behavior with 11gR2, a node actually under certain circumstances won't reboot completely; rather, it will attempt to stop the GI on the node gracefully to avoid complete node eviction. For example, when the GI home or `/var` file system runs out of space on the device, `cssd/crsd,evmd` processes become unhealthy, in contrast to pre-11gR2 behavior, and this prevents a total node reboot scenario. Restarting the individual stack resources after resolving the underlying issues in the context is recommended to bring the cluster stack back to a healthy situation. However, if the `rebootless fencing` fails to stop one or more resources, the `rebootless fencing` will fail and the `cssdagent` or `cssdmonitor` will favor and trigger complete node eviction and will perform a fast reboot. In this context, a warning message of resource failure will be written to the `alert.log` file.

Extended Distance (Stretch) Clusters—Synopsis, Overview, and Best Practices

Extended distance (stretch) clusters (Figure 7-10) are a very rare species and justifiably so. RAC clusters are generally architected to reside within a single geographical location. Stretch clusters are generally defined as RAC clusters with nodes spanning over multiple data centers, resulting in a geographically spread-out private cluster interconnect.

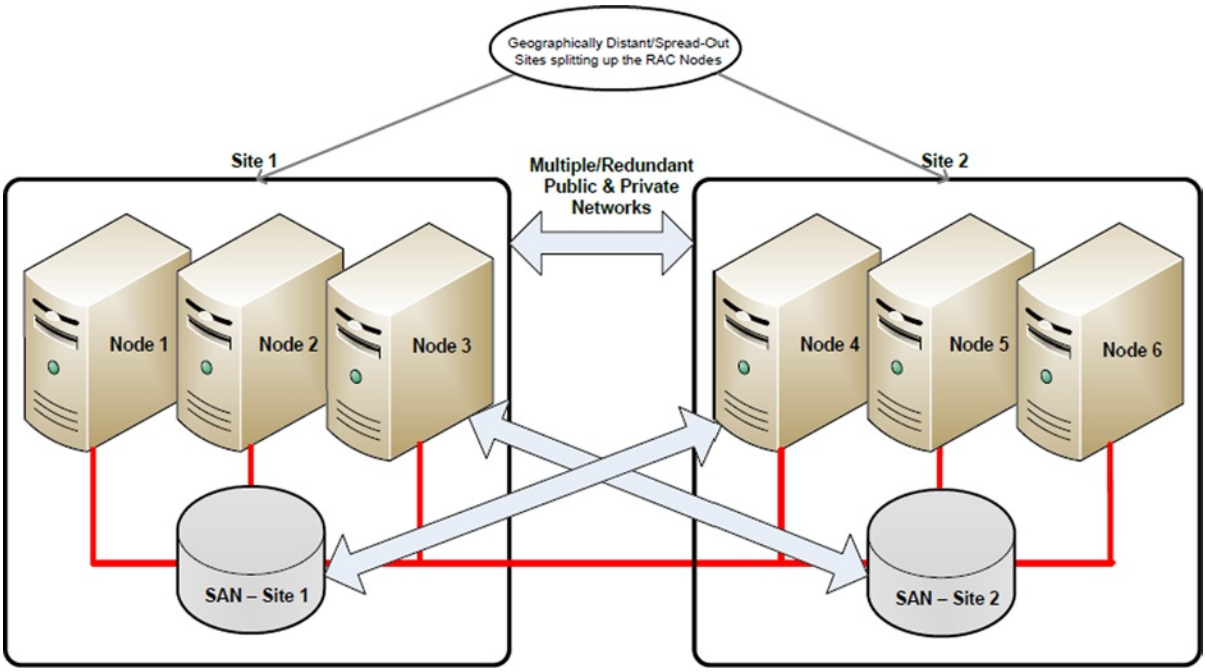


Figure 7-10. *Example configuration: Extended distance (stretch) RAC cluster*

Basically, stretch clusters are used for a very specific reason: Building Active-Active RAC clusters, spread over a geographically spread-out location. However, due to network latency and cost issues, Active-Active Multi-Master Golden Gate is a much more viable option than extended distance clusters. It is important to consider that stretch clusters *are not* and *should not* be used as a replication/Disaster Recovery technology; Data Guard is the recommended technology for DR/failover purposes.

Extended Distance (Stretch) Clusters: Setup/Configuration Best Practices

Following are good design best practices critical to the smooth operation of extended distance clusters:

- Implement/configure redundancy at the public network layer
- Dense wavelength division multiplexing (Dark Fibre) is the general technology of choice for implementing a dedicated networking tier for stretch clusters, as it provides the high-speed low-latency network channel needed for the private cluster interconnect
- Implement and configure full redundancy at the private cluster interconnect networking layer
- The private cluster interconnect must be on the same subnet across all sites/nodes
- The public network must be on the same subnet across all sites/nodes
- Implement/configure full redundancy at the storage fabric layer
- Set up/configure ASM for normal redundancy for maintaining a local copy of the database at all sites of the RAC cluster
- Implement/configure preferred mirror disk reads (ASM_PREFERRED_READ_FAILURE_GROUPS parameter) within ASM so that the disks nearest (local) to the particular nodes are utilized
- Implement the voting disk on a 3rd geographical site on NFS; this can be done over a WAN
- Do not implement extended distance clusters over distances of more than 50-70 miles

Following are the pros/benefits/advantages of extended distance clusters:

- Fast site-level failover protection solution among geographically spread-apart nodes
- Active-Active configuration: full consumption/usage of all involved hardware
- Scalability/High Availability RAC features not limited to a single geographical location
- ASM data mirroring: multiple copies of the same data in geographically distant locations
- Fast: good overall performance

By the same token, extended distance clusters are limited in their capabilities by the following disadvantages/cons:

- High costs associated with the need for high-speed networks
- A dedicated redundant high-speed network is needed for the public/private network and storage tiers. General rule of thumb: the more the geographical distance, the more the network latency
- Generally/usually, after 50-100 miles, the latency with most current technologies starts to gets prohibitive for the smooth operation of an extended distance (stretch) cluster
- No DR protection from large-scale geographical disasters due to the latency limitations imposed by the need for a high-speed dedicated network infrastructure

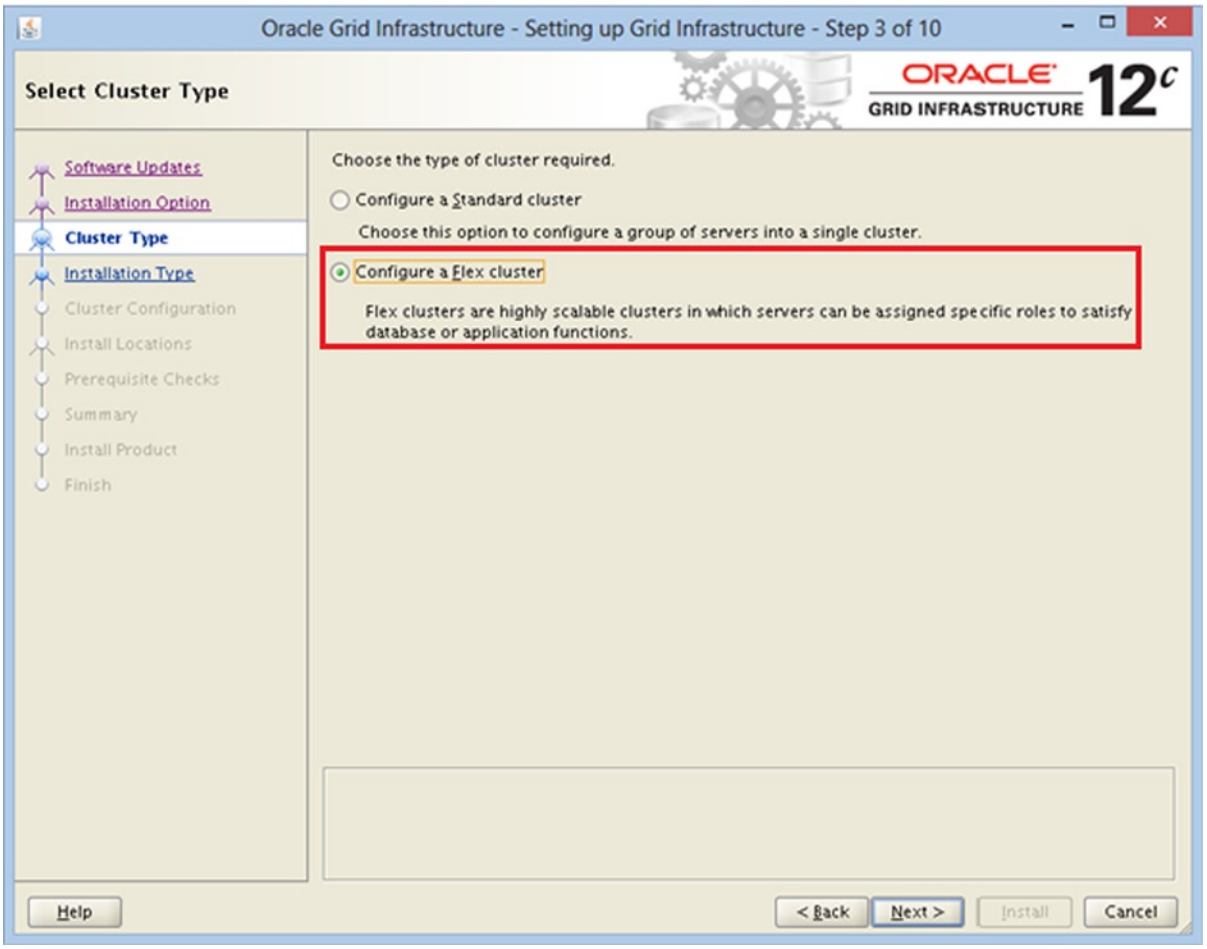
- More space due to "normal redundancy" ASM mirroring (equivalent to RAID10) required on both ends. This is best practice for setting up extended distance clusters
- Network latency, due to geographical separation, if not configured/kept in check properly, can have a crippling impact on the overall performance of a stretch RAC cluster

Setup and Configuration—Learning the New Way of Things

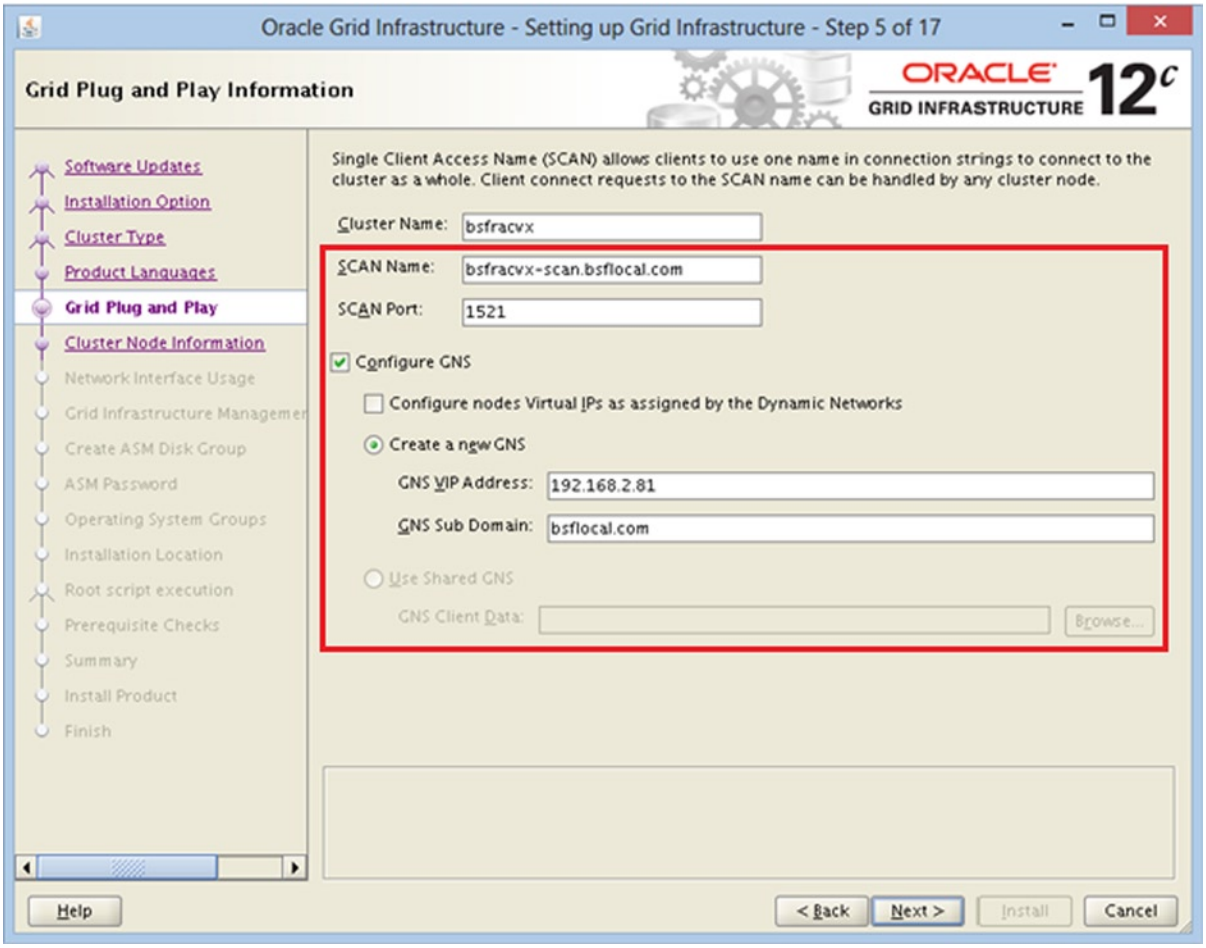
OUI

OUI has come quite a ways in its long and distinguished history. The new installer grows more powerful, intuitive, and easy to use as a integral tool with each new version; that helps with quite of bit of automation in setting up and configuring an Oracle RAC cluster according to industry-standard best practices.

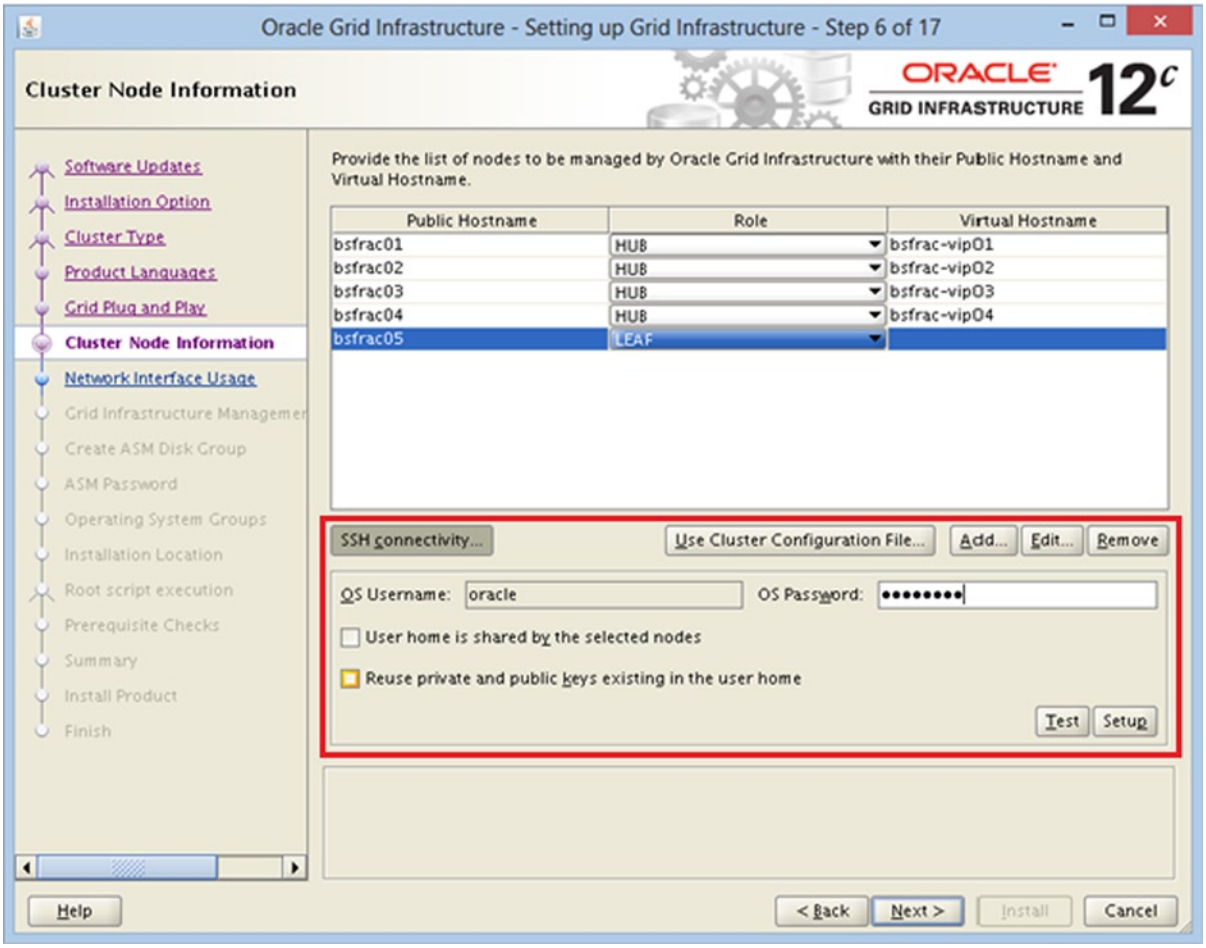
Shown in Figures 7-11, 7-12, 7-13, and 7-14 are some examples of recent enhancements in OUI.



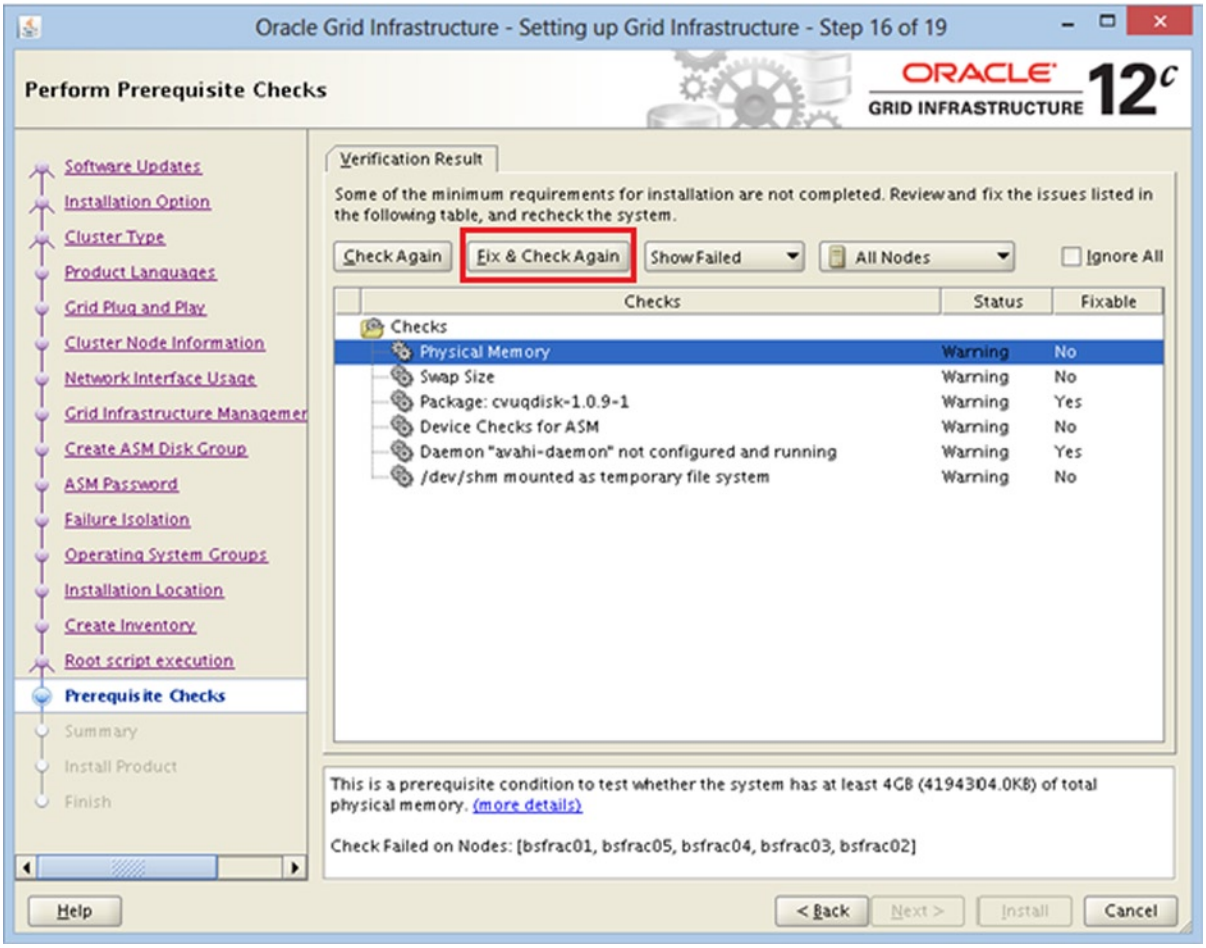
Figures 7-11. *Examples of recent enhancements/features added to the OUI for setting up a RAC cluster*



Figures 7-12. *Examples of recent enhancements/features added to the OUI for setting up a RAC cluster*



Figures 7-13. *Examples of recent enhancements/features added to the OUI for setting up a RAC cluster*

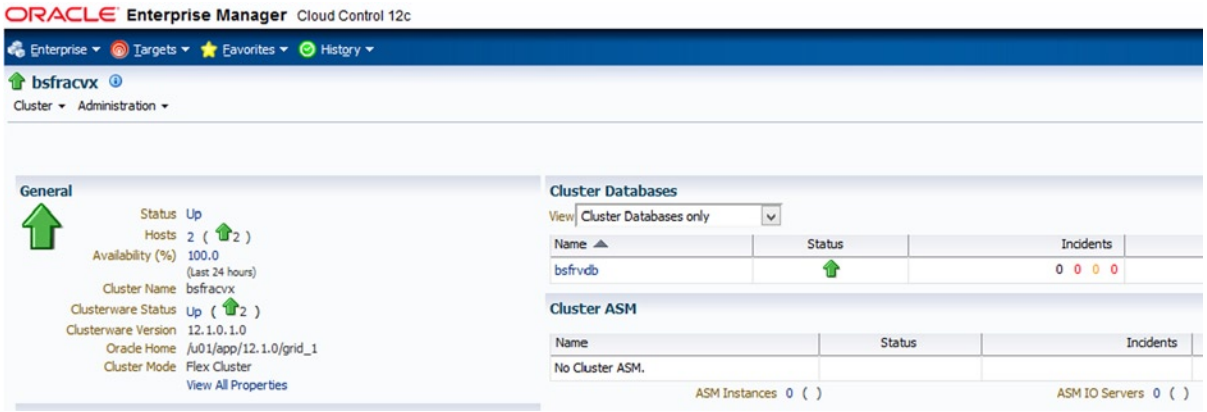


Figures 7-14. *Examples of recent enhancements/features added to the OUI for setting up a RAC cluster*

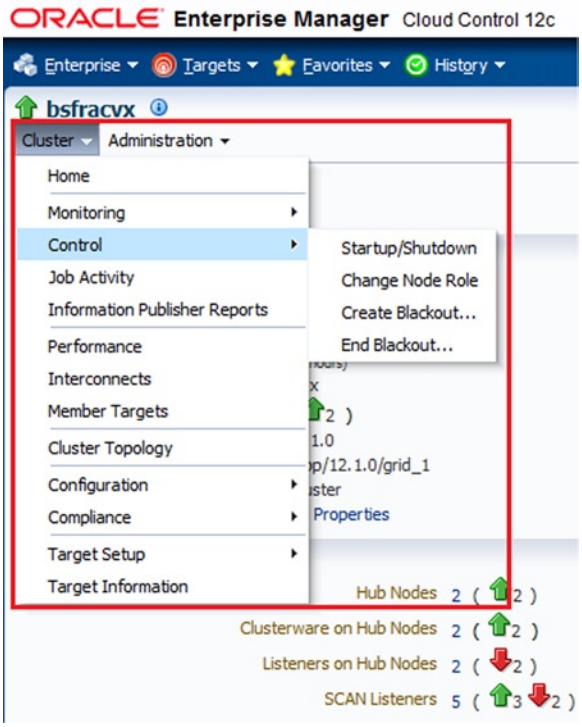
Oracle Enterprise Manager Cloud Control 12c

Oracle Enterprise Manager is the end-to-end tool for configuring, managing, and monitoring RAC clusters. Features like conversion of Oracle databases from single-instance to RAC in Oracle Enterprise Manager Cloud Control 12c are some of the many powerful features that should be leveraged, especially when databases need to be managed or configured en masse.

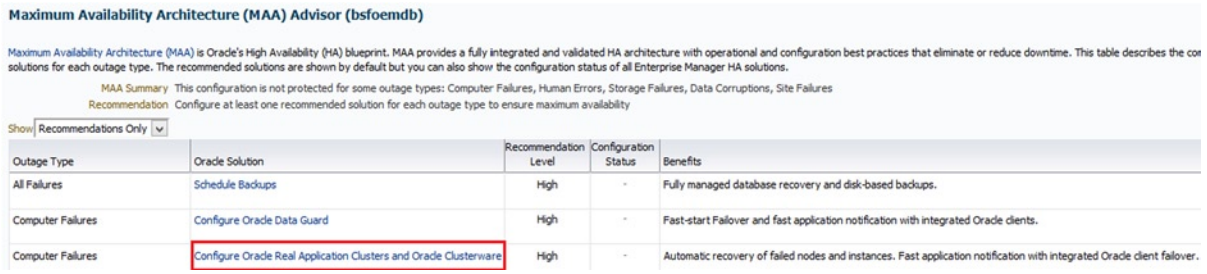
It is important to note that Oracle Enterprise Manager along with RAC and ASM formulate the Database-as-a-service cloud paradigm. This underscores the significance of OEM within the RAC ecosystem as a whole: in other words, they integrally complement each other to constitute the Oracle Database Cloud.



Figures 7-15. *Example activity: Configuring/managing RAC within Oracle Enterprise Manager Cloud Control 12c*



Figures 7-16. *Example activity: Configuring/managing RAC within Oracle Enterprise Manager Cloud Control 12c*



Figures 7-17. *Example activity: Configuring/managing RAC within Oracle Enterprise Manager Cloud Control 12c*

RAC Installation and Setup—Considerations and Tips for OS Families: Linux, Solaris, and Windows

There may be some overlap in the material presented in this section with other chapters. The intent is to summarize proven and well-known tips, tricks, and best practices in setting up and configuring Oracle RAC clusters in the various variants of popular OS families of Linux, Solaris, and Windows. The list is long and expansive and therefore, not everything can be covered in detail; salient bullet points follow.

Linux:

- Use HUGEPAGES especially for large Oracle RAC DBs (according to Oracle documentation, any Oracle Database with an SGA >= 8 GB qualifies as a large database). HUGEPAGES is the general rule of thumb for effective and efficient usage of memory within the Linux OS family
- Configure/set up ASMM, as it is the only Oracle Database Server Automatic Memory Management technology that is compatible with HUGEPAGES
- Install all prerequisite packages before installing GI/RAC
- Create/configure the minimum required OS users/user groups
- Set up/configure X Windows along with X11 forwarding so that graphical output can be redirected for common utilities, for example, DBCA, ASMCA, OUI, NETCA, etc.
- Set up/configure UDEV for ASM disks
- Configure/set up/utilize asynchronous I/O

Solaris (SPARC):

- Employ/deploy Oracle VM for SPARC, an effective virtualization approach which allows you to easily virtualize/partition the physical servers for RAC
- Check/install the minimum version of firmware on Sun servers
- Create/configure the required OS users/user groups
- Set up/configure the recommended/required UDP/TCP parameters for the kernel
- Set up/configure X forwarding over SSH for graphical redirection of common Oracle utilities, for example, DBCA, OUI, etc.
- Create/configure the minimum required OS users/user groups
- Guest LDOMs should be utilized as Oracle RAC nodes

- Set up/configure/utilize Network Time Protocol (NTP) or Cluster Time Synchronization Service (CTSS) for time synchronization across all nodes of the RAC cluster
- No I/O, control domains, or service domains should be utilized as Oracle RAC nodes
- Set up/configure the recommended shell limits and system configuration parameters
- Run CLUVFY at various stages of the RAC setup/configuration process

Windows:

- Needless to say, 64-bit version of Windows is an absolute must; this is a basic-level recommendation, part of RAC101. Windows server 2008 R2 SP2 is a good starting point
- Create/configure the required OS users/user groups
- Assign/configure Local security policies to the OS "Oracle" user
- Create the required OS "environment variables" for setting up RAC
- Set up/configure cluster nodes as "application servers"
- Synchronize the clocks on all RAC nodes
- Set up/configure regional/language settings
- Set up the order of the NICs such that they are set up/bound so that the "PUBLIC NIC" precedes the "PRIVATE NIC"
- Enable/configure auto mounting of ASM disks
- Set up/configure ASM disks as RAW
- Disable write caching on all ASM disks
- Configure/set up/utilize asynchronous I/O
- Create/set up/configure extended partitions on all ASM disks and then create logical drives within them
- Use non-shared (local) Oracle Homes on each node; this allows rolling-fashion patching/upgrades
- Disable/stop the "Distributed Transaction Coordinator" Windows service on all RAC nodes
- Make abundant use of CLUVFY at each stage of the RAC setup/configuration process
- Configure/set up HyperThreading
- For large databases, set up/configure Windows Large Pages on OS version >= Windows server 2008 R2 SP2
- Leverage/utilize Performance Monitor, Process Explorer, and Event Log Windows utilities for troubleshooting/monitoring purposes
- Any/all other miscellaneous OS-level environment preparation steps

RAC Database Performance Tuning: A Quick n' Easy Approach

OEM12c is a leap forward in sophisticated DB management, monitoring, and administration—a great tool for quick, easy, and intuitive performance tuning of a RAC cluster. This section is an overview, and further details are presented in other chapters.

OEM Cloud Control facilitates and accelerates identification and fixing of DB bottlenecks, problematic SQL statements, locks, contention, concurrency, and other potential crisis situations, including monitoring/logging into hung databases.

Holistic graphical analysis can quickly and easily help a RAC DBA/DMA to identify performance bottlenecks at the cluster/instance level, resulting in lower incident response times.

The 3 A's of Performance Tuning

The 3 A's of performance tuning in a RAC Database, all of which are easily accessible in quite a bit of drill-down detail within OEM12c, are listed in the following:

- AWR (Automatic Workload Repository)
- ADDM (Automatic Database Diagnostic Monitor)
- ASH (Active Session History)

The preceding are supplemented by the following new/recent features within OEM12cR2:

- Real-time ADDM
- Compare real-time ADDM reports
- ASH analytics
- Top activity
- SQL monitoring

Some of these much-needed tools/features are incorporated within the diagnostics and tuning pack, which is essential for enhanced and effective monitoring, troubleshooting, and performance tuning the Oracle Database Server family including RAC.

The frequency of slow times in a RAC database is way higher than downtime: the perfect tool to attack this problem is OEM12cR2 ➤ Performance Home Page, which, as the heart and soul of the visual monitoring/analysis process/methodology, gives you a clear and easy-to-interpret overview of the RAC DB health in real time as well as from a historical perspective.

OEM 12c R2 ➤ Performance Home Page

The Performance Home Page within OEM12cR2 is all about graphs, charts, and colors (OR pictures)! This is an intuitive starting point for effective and easy-to-learn performance tuning of the Oracle Database Server family including RAC.

Tips/tricks/methodologies:

- Familiarize yourself with the graphs and learn to understand them in detail over time.
- Get accustomed to your DB's baselines during peak activity timeframes.
- Know what the colors represent and the overall global picture that they want to communicate to you as a DBA.

Some examples of powerful and easy-to-use performance tuning features within OEM 12cR2 are shown in Figures 7-18 and 7-19.

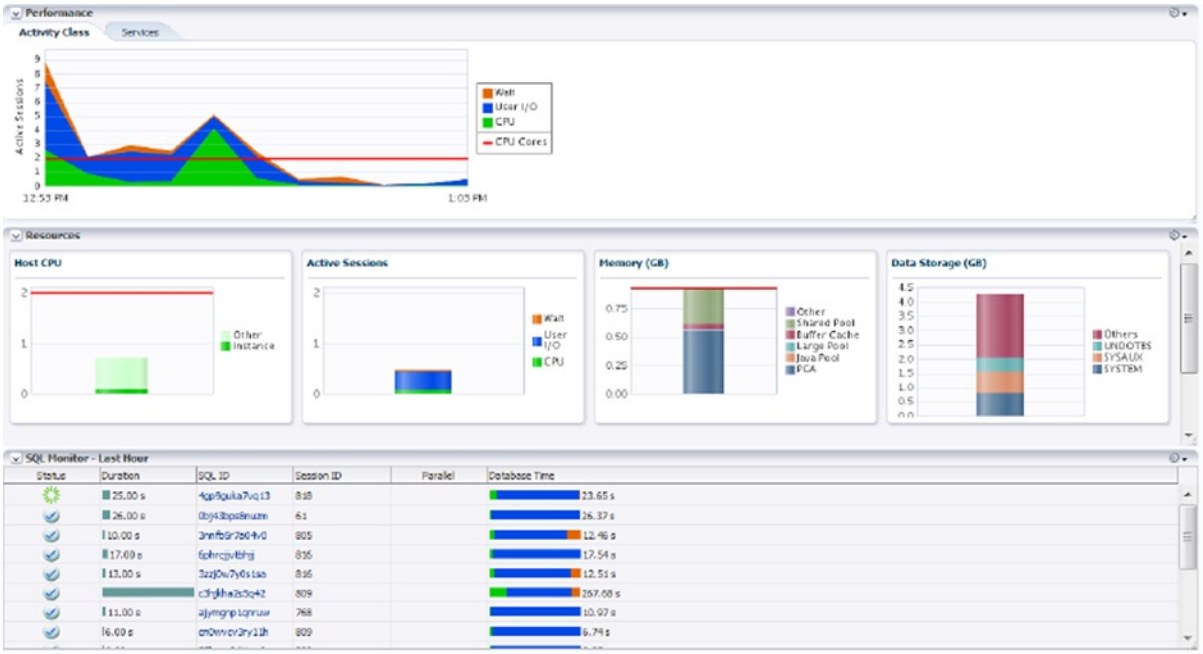


Figure 7-18. OEM 12c ➤ Targets ➤ Select RAC DB ➤ Home Page *gives performance, resources, and SQL summary*

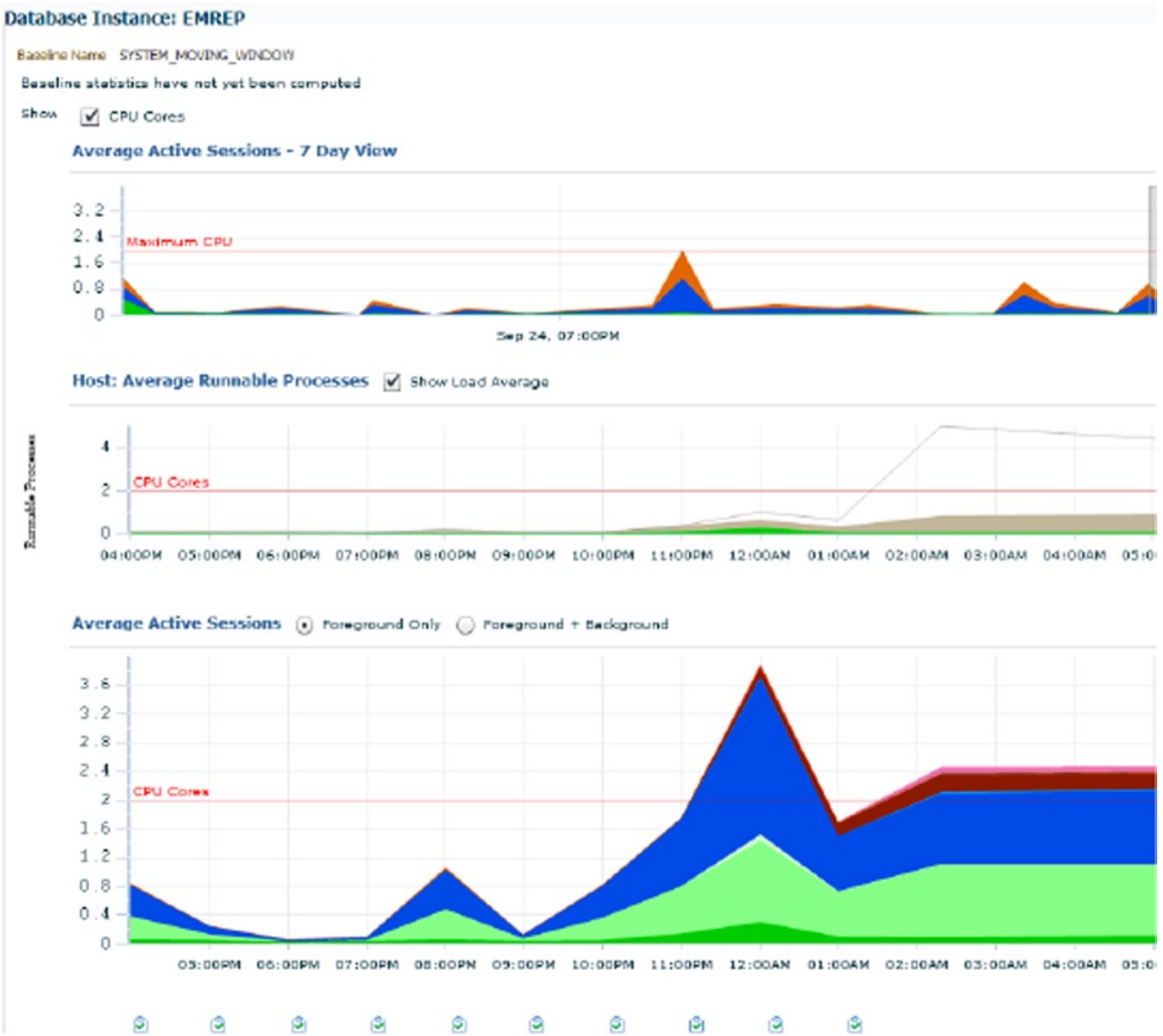


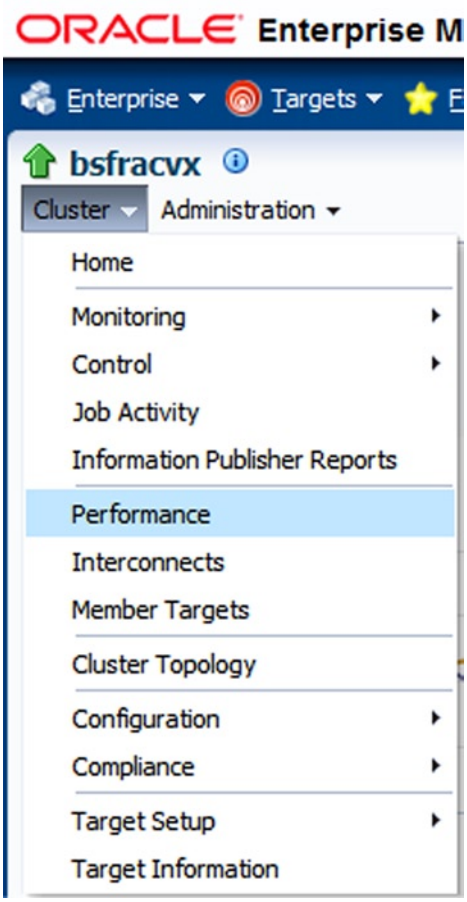
Figure 7-19. Performance Home Page: *The stargate of Oracle RAC (and non-RAC) DB tuning in OEM Cloud Control*

"Performance" Menu ➤ "Performance Home" homepage:

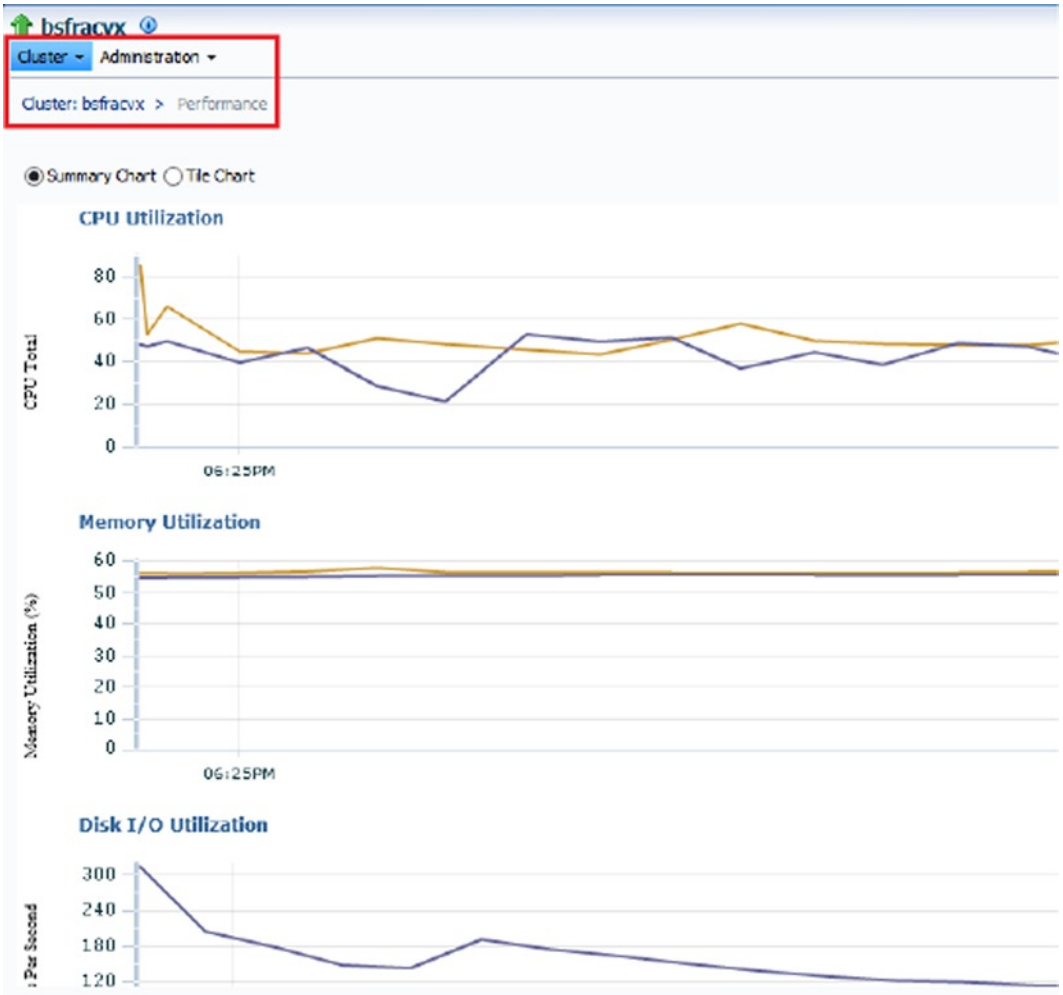
- A very structured and useful organization of performance-centric links/resources.
- Significant improvement over 10g/11g OEM Grid Control.
- Will ask you to log in with SYSDBA or DBSNMP monitoring credentials.

Understanding the colors on the various graphical depictions and the information/severity level they represent is very important to this approach: generally, the more the reddish the color, the more trouble it is for the RAC DB.

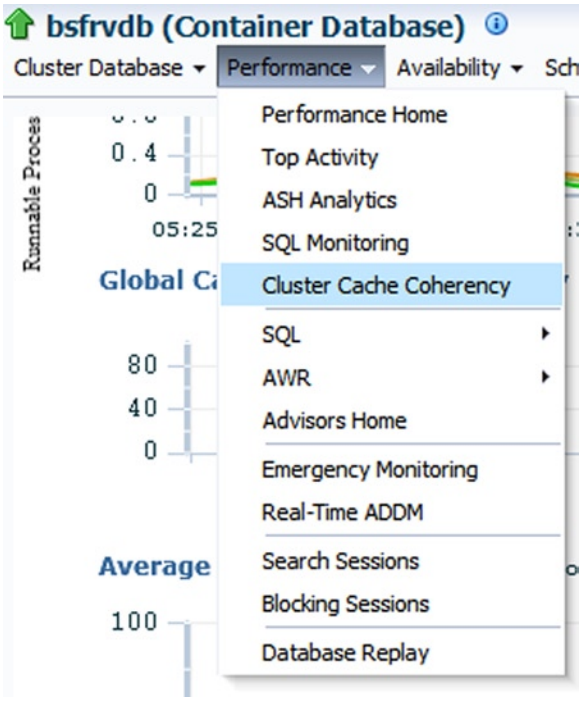
- Greenish/blueish colors.
- Grayish colors.
- Brownish colors.
- Reddish colors.



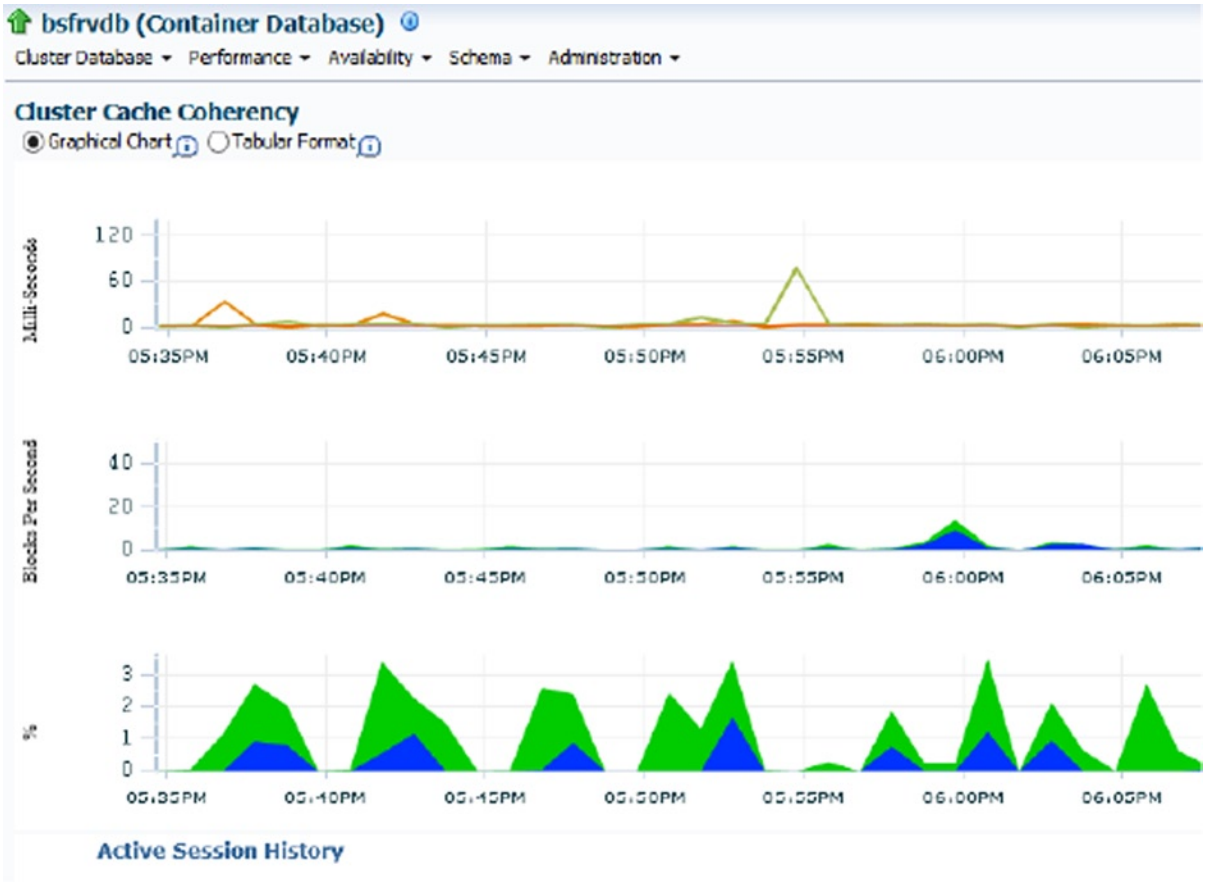
Figures 7-20. Cluster ➤ Performance: *CPU, memory, and disk utilization within OEM12c*



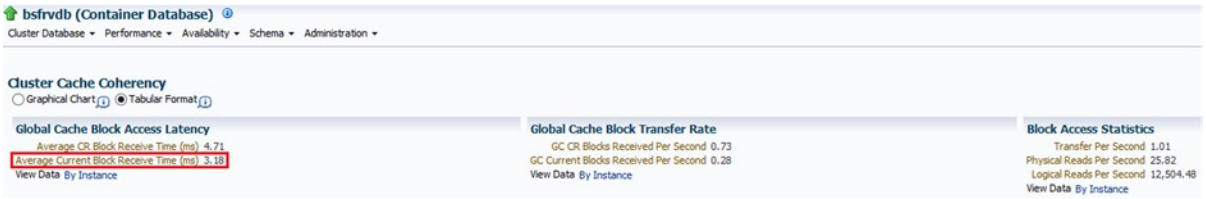
Figures 7-21. Cluster ➤ Performance: *CPU, memory, and disk utilization within OEM12c*



Figures 7-22. Cluster cache coherency link/menu item: *The main link/entry page for RAC cluster monitoring/tuning*



Figures 7-23. Cluster cache coherency link/menu item: *The main link/entry page for RAC cluster monitoring/tuning*



Figures 7-24. Cluster cache coherency link/menu item: *The main link/entry page for RAC cluster monitoring/tuning*

The preceding examples are just some of many powerful features for a quick n' easy way of performance tuning RAC clusters within OEM12cR2.

Summary

RAC lies at the most complex end of the Oracle Database Server family spectrum and can be challenging to get one's arms around, especially for large-scale setups. Managing and optimizing any RAC cluster require a considerable amount of skill, experience, and expertise. This chapter covered most essential knowledge streams related to easy, policy-based, and easy-to-learn management of Oracle RAC clusters. From node evictions to split-brain scenarios, tips/tricks for various OS families, server pools and stretched clusters, and a whole lot more, a wide and expansive range of topics was covered in this chapter to efficiently and effectively cope with/ease the daily load of a RAC DBA/DMA.

CHAPTER 8

Backup and Recovery in RAC

by Syed Jaffar Hussain

Oracle Real Application Clusters (RAC) minimizes the impact of unplanned instances outages with its high-availability infrastructure. Nevertheless, it remains a crucial and very challenging task for DBAs to safeguard business-critical data from database failures and ensure its continuous availability. Data loss may occur in the form of any hardware, software, or disk failures or unintentional human error. As a result, the DBA must develop and test comprehensive strategies for database backup and recovery operations. More importantly, a key factor to consider is the ability to restore and recover the database within business-agreed time limits (such as a service-level agreement, or SLA) to ensure quick database accessibility to the end users.

It is important to emphasize that there are no substantial differences in the ways in which database backup and recovery operations are performed in clustered and nonclustered environments. This chapter focuses on leveraging Recovery Manager (RMAN) advantages in the RAC environment to develop optimal backup and recovery procedures. This chapter also explains the concepts of instance and crash recovery mechanics in RAC environments and demonstrates various Oracle Cluster Registry (OCR) file recovery scenarios.

RMAN Synopsis

RMAN is an Oracle client tool that was first introduced with Oracle v8 to perform a wide range of database backup and recovery operations. With RMAN, DBAs can develop comprehensive, reliable, and secure database backup and recovery procedures. RMAN ships with Oracle RDBMS software and is automatically installed upon Oracle database software installation for no additional cost. The DBA can later adjust the RMAN configuration according to database needs. RMAN simplifies all Oracle database backup and recovery procedures, and the use of RMAN is highly recommended by Oracle.

RMAN offers the following key benefits:

- A wide range of flexible backups: tablespace, datafile, image copy, incremental, fast increment, encrypted, full database, and so on.
- Support for various levels of recovery scenarios: block-level, complete, incomplete, point-in-time, tablespace, table, and so on.
- Management of cold and hot database backups.
- A command-line interface (CLI) and Oracle Enterprise Manager (OEM) GUI interfaces.
- Fast incremental backups.
- Compressed and encrypted backup support.
- Ability to detect and repair data block corruptions.

- Multisection backups to improve large data file backups.
- Undo backup optimization.
- Support for backups to disk or tape devices.
- Validation of existing backup consistency.
- Cross-platform database transportation.
- Simplified backup and recovery procedures.
- Easy to duplicate a database, construct a standby database, and clone the database.
- Close coordination with the Oracle database engine.
- Recovery operation can be tested without actually being performed.
- Capacity to record 7 days of database backup and recovery operations action in the database control file by default.
- Support for backup and various recovery scenarios of 12c new container and pluggable databases.
- Storage snapshot optimization for third-party technologies.
- With 12c, a table or table partition can be recovered from RMAN backups.

Figure 8-1 represents a typical RMAN connection workflow.

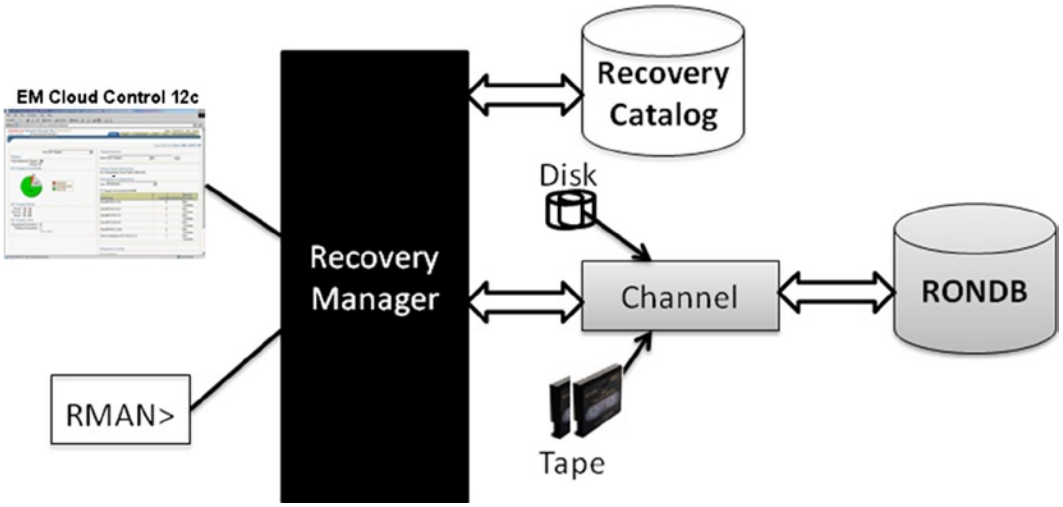


Figure 8-1. *RMAN connection workflow*

Media Management Layer

To be able to perform database backup and recovery operations to and from tape, a media management library (MML) should be configured and must be integrated with RMAN. Oracle Secure Backup (OSB) provides a media manager layer and supports RMAN backup and recovery operations to tape on the majority of operating systems (OS). Similarly, there are a few third-party vendor media management solutions available for the same purpose, such as Symantec NetBackup, EMC NetWorker Module for Oracle, IBM Tivoli Data Protection for Oracle (TDPO), and HP Data Protector (DP).

Once the media manager software is successfully installed, it needs to be tightly integrated and must interact with RMAN. After the configuration, ensure that an RMAN can interact without any issues, and you need to configure or set the SBT channels in RMAN pointing to appropriate media manager software. The following examples of different MML options illustrate the backup run level configuration settings with RMAN:

OSB:

```
RMAN> run{
 ALLOCATE CHANNEL ch00 DEVICE TYPE sbt
    PARMS 'SBT_LIBRARY=/<dir_location>/lib/libobk.so,
    ENV=(OB_DEVICE=drive1,OB_MEDIA_FAMILY=datafile_mf)';
.......
Release channel ch00;
}
```

Symantec NetBackup:

```
RMAN> run{
 allocate channel ch00 type 'SBT_TAPE';
 allocate channel ch01 type 'SBT_TAPE';
SEND 'NB_ORA_CLIENT=host_name,NB_ORA_POLICY=RONDB_FULL_LEVEL0,NB_ORA_SERV=netbackup_master_server_name';
..
release channel ch00;
release channel ch01;
}
```

IBM TDPO:

```
        RMAN> run{
ALLOCATE CHANNEL ch00 type 'sbt tape' parms 'ENV=(TDPO_OPTFILE=/opt/tivoli/tsm/client/tdpo.opt)';
;
..
release channel ch00;
release channel ch01;
}
```

You must ensure that the MML is configured successfully without any issues; otherwise, you will run into troubles executing backup/recovery operations. Similarly, you can also define static MML tape configuration in the RMAN configuration.

Online Backup and Recovery Prerequisites

As a prerequisite to perform a wide range of online database backup and recovery operations, with or without RMAN, the database in the context must be configured in archivelog mode with a few additional recovery-related database initialization parameters. The following procedure configures a RAC database in archivelog mode and also sets the database recovery–related initialization parameters:

1. Define a user-specified, upper-limit size to the recovery destination:

   ```
   SQL> alter system set db_recovery_file_dest_size=1g SCOPE=BOTH SID='*';
   ```

2. Specify a shared recovery destination path (in this example, an ASM diskgroup) in which archivelogs from all instances of a RAC database will be stored:

```
SQL> alter system set db_recovery_file_dest = '+DG_RONDB_ARCH' SCOPE=BOTH SID='*';
```

■ **Note** In a RAC database, the settings must remain consistent across all instances.

To enable archivelog mode:

1. Shut down the database first:

```
# srvctl stop database –d RONDB –o immediate
```

2. Mount the first instance either from the SQL*Plus prompt or by using the srvctl command, as follows:

```
SQL> startup mount;
# srvctl start database –d RONDB –o mount
```

Unlike pre-11gR2, you don't need to off/on the `cluster_database` parameter to switch between the archive/noarchive modes.

1. Switch the database mode to archivelog and start the database using the following examples:

```
SQL> alter database archivelog;
SQL> alter database open;
```

2. To start the rest of the instance, use one of the following examples:

```
# srvctl start database –d RONDB
# srvctl start instance –d RONDB –I RONDB2
```

Once the preceding settings are made, connect to `SQL*Plus` and verify the database log mode and recovery parameters using the following commands:

```
SQL> SELECT log_mode FROM v$database;
SQL> archive log list
```

The outcome should be similar to the following:

```
SYS @RONDB1 >archive log list
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            USE_DB_RECOVERY_FILE_DEST
Oldest online log sequence     10
Next log sequence to archive   12
Current log sequence           12
```

From the preceding output, you can see that the database log mode is in Archive Mode now, and the archivelog location is set to Flash Recovery Area.

Non-RAC vs. RAC Database

Before we really jump in and get started on backup and recovery concepts in the RAC environment, let's examine the architectural differences between a clustered and a nonclustered database.

There are no substantial architectural differences between a single instance and RAC database except that multiple instances access a single physical database in the RAC environment. Moreover, every instance has its own undo tablespace, a set of individual redo groups, and thereby generates instance-specific archived logs. No additional or special skills are required to manage database backup and recovery for a RAC database. In contrast, there is a significant difference in the way instance, media, or crash recoveries are performed in a RAC database. When an instance is involved in any sort of recovery operation, and if it requires the redo/archive log generated by the other instance in the context, the instance that is performing the recovery must have access to the other instance redo/archive logs to successfully complete the recovery. Therefore, it is strongly advised to put the redo/archive logs at a shared location so that all instances of the RAC database can have access.

Figure 8-2 depicts the architectural differences between a standalone and a RAC database.

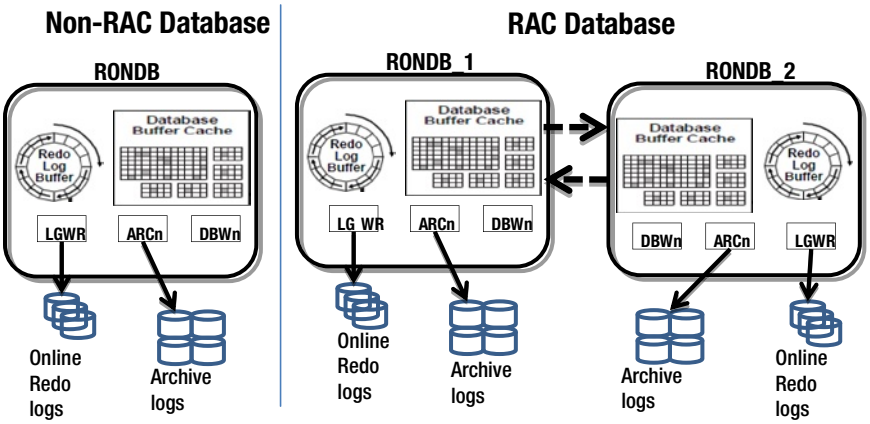


Figure 8-2. *RAC vs. non-RAC architecture*

Shared Location for Redo and Archive Logs

As stated earlier, each instance in a RAC database typically consists of its own set of redo logs and generates individual archive logs. The redo group and archive logs in a RAC database are solely identified by a distinct thread number to avoid ambiguity. For example, a redo member and archive log of instance 1 will be identified as thread_1.

Despite each instance having absolute control over its own redo and archive logs, it is essential that all instances have the privilege to read each other's redo and archive logs to successfully perform various RAC database recovery tasks, such as instance recovery, crash recovery, and media recovery. It is a requirement for a RAC database to have the redo logs across all instances on a shared destination, also preferably archive logs, to ensure accessibility between the instances.

Oracle strongly suggests making use of the Fast Recovery Area (FRA), which is a centralized location for all backup-related and recovery-related files. The FRA could be configured on an ASM diskgroup, Cluster Filesystem, or Network File System by configuring the two dynamic database initialization parameters. When there is no FRA specified during database creation process, the FRA will be automatically set under the Oracle base directory.

When no shared spfile is used, the following procedure needs to run across all instances and must contain the same values. If a shared spfile is configured, you need only to run the commands from one instance:

- Set a user upper-limit size to the recovery destination:

```
SQL> alter system set db_recovery_file_dest_size=10G scope=both;
```

- Specify the recovery path destination:

```
SQL> alter system set db_recovery_file_dest='+DG_RONDB_ARCH' scope=both;
```

Figure 8-3 illustrates the concept of a centralized recovery area in a RAC database.

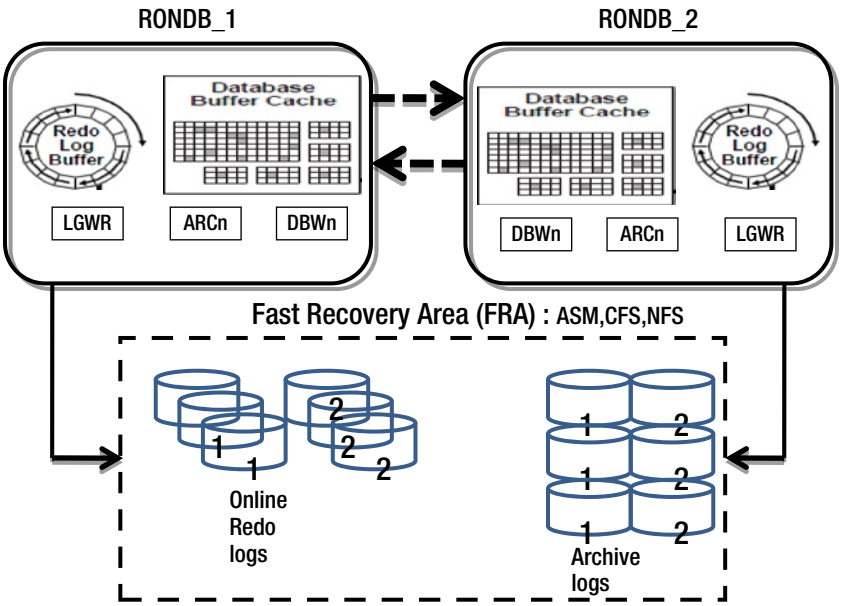


Figure 8-3. *FRA settings*

Following are a few useful database dynamic views that render a better understanding of FRA usage and other specific details:

```
SQL> SELECT * FROM V$RECOVER_AREA_USAGE;
SQL> SELECT * FROM V$FLASH_RECOVERY_AREA_USAGE;
SQL> SELECT * FROM V$RECOVERY_FILE_DEST
```

Snapshot Control File Configuration

Beginning with 11.2.0.2, for any form of database control file backup operations, Oracle no longer acquires the controlfile enqueue lock. This change requires snapshot control file accessibility to all instances in a RAC database, in contrast to a single-instance database, to successfully complete a manual or auto control file backup operation. When the snapshot control file is not accessible or doesn't exist across instances, the backup of control file operations, including RMAN backups, will fail with an '`ORA-00245: control file backup operation failed`' error.

To prevent an ORA-00245 error, the snapshot control file must be placed on a shared location or on ASM DISKGROUP, ensuring that each instance of RAC database has the read and write ability on the file.

The following procedure explains the process to configure the snapshot control file on an ASM DISK GROUP:

```
RMAN> CONFIGURE SNAPSHOT CONTROLFILE NAME TO '+DG_RONDB/snapcf_RONDB.f';
```

Upon configuration, the following message will be displayed at RMAN prompt:

```
using target database control file instead of recovery catalog
new RMAN configuration parameters:
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '+DG_RONDB\snapcf_RONDB.f;
new RMAN configuration parameters are successfully stored
```

To display snapshot control file configuration details, connect to an RMAN prompt and execute one of the following commands:

```
RMAN> show all;
RMAN> show snapshot controlfile name;
```

Multiple Channels Configuration for RAC

On a RAC database, it is feasible to have multiple channels allocated and scattered over multiple instances to scale up the backup and recovery workload. This section explains how to configure parallelism to enable load balancing for a preallocated (automatic) or runtime RMAN channel.

Parallelism for Load Balancing

Parallelism must be configured in RMAN prior to making use of multichannel configuration for balancing the backup and restore workload over multiple instances. The following example configures the parallelism and enables the load balancing option, also overwriting any pre-existing parallelism configuration settings:

```
RMAN>  configure device type disk parallelism 2;

new RMAN configuration parameters:
CONFIGURE DEVICE TYPE DISK PARALLELISM 2 BACKUP TYPE TO BACKUPSET;
new RMAN configuration parameters are successfully stored

RMAN> show all;            -- shows RMAN configuration settings

RMAN configuration parameters for database with db_unique_name RONDB are:
........
CONFIGURE DEVICE TYPE DISK PARALLELISM 2 BACKUP TYPE TO BACKUPSET; <<<<<
```

Persistent Multichannel Configuration

To scale an RMAN backup workload over multiple instances, you need to allocate multiple channels directing the connection to an individual instance. The allocation of channels can be a preconfigured or just a one-time (dynamic) allocation.

The following example configures a preconfigured (auto) and instance-specific connection. RMAN will utilize the preconfigured (auto) channels to perform the job when no channels have been allocated within the backup and recovery operations.

```
RMAN> configure channel 1 device type disk connect 'sys/password@RONDB_1';
RMAN> configure channel 2 device type disk connect 'sys/password@RONDB_2';
```

The RONDB_1 and RONDB_2 TNS configurations used in the preceding example are defined in instances 1 and 2, respectively, and contain pointers to individual instances. Ensure that the TNS configuration does not use a service name that is configured to multiple instances and that the load balancing option is turned off.

All RMAN backups followed after the configurations will automatically allocate two channels over instances 1 and 2, and the workload will be balanced across instances.

Runtime (One-Time) Multichannel Configuration

Similarly, multiple channels also can be defined dynamically just for the duration of backup runtime by specifying them within the RMAN backup script, as follows:

```
run
{
               allocate channel 1 device type disk connect='sys/password@rondb_1';
               allocate channel 2 device type disk connect='sys/password@rondb_2';
               ...
}

run
{
               allocate  channel 1 device type disk connect='sys/password@RONDB_RMAN';
......
}
```

When a backup is initiated with multiple channels, you will see a message similar to the following:

```
Starting backup at 13-NOV-12
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=102 instance=RONDB_2 device type=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: SID=16 instance=RONDB_1 device type=DISK
```

In the first example, two channels are dynamically allocated on two different instances using a predefined TNS connect string, specific to each instance, and the TNS connection does not have the load balancing option enabled. With this approach, you can control which channel should go to which instance, rather than relying on automatic load balancing algorithms.

Similarly, in the second example, two channels are dynamically allocated using a TNS or service connection with load balancing options turned on. This approach banks on the load balancing algorithm and allocates channels on instances according to a predefined formula.

Moreover, the life of the dynamic channel is strictly limited to the runtime. Once the backup operations get completed, the channels are no longer valid.

During a database backup, if the instance on which the backup was initiated terminates unexpectedly, the backup job won't fail; instead, the other channel from an active instance would take over the job and complete the backup. Assuming that, a backup operation with two channels has been initiated from instance 1 and the instance has

terminated while the backup is in progress. In this context, the other active channel, from instance 2, would take over the job and complete the backup with the following RMAN message:

```
channel ORA_DISK_1: starting piece 1 at 13-NOV-12
RMAN-03009: failure of backup command on ORA_DISK_2 channel at 11/13/2012 17:17:11
RMAN-10038: database session for channel ORA_DISK_2 terminated unexpectedly
channel ORA_DISK_2 disabled, job failed on it will be run on another channel
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-03009: failure of backup command on NIL channel at 11/13/2012 17:17:11
RMAN-10038: database session for channel NIL terminated unexpectedly

channel ORA_DISK_2: starting piece 1 at 13-NOV-12
RMAN-03009: failure of backup command on ORA_DISK_2 channel at 11/13/2012 17:29:20
RMAN-10038: database session for channel ORA_DISK_2 terminated unexpectedly
channel ORA_DISK_2 disabled, job failed on it will be run on another channel
```

■ **Note** The database status (OPEN or MOUNT) must remain constant across all nodes during a backup or the backup will fail.

Additionally, the backup destination must be accessible to all the instances involved in the backup operation. If the backup location is inaccessible or doesn't exist to any particular instance involved in the backup operation, the backup will fail with 'unable to create file' errors. This is particularly significant when the backups are taken to a disk location. Similarly, when backing up the database to tape, ensure that the tape and media manager driver are configured properly on all the nodes involved in the backup.

Figure 8-4 depicts the architecture for multichannel disk or tape RMAN configuration.

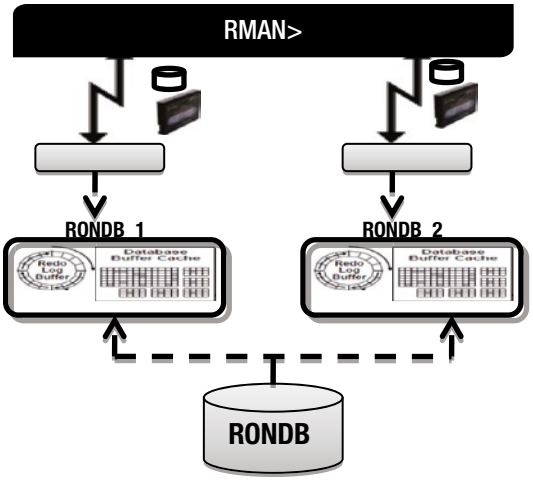


Figure 8-4. *RMAN TAPE connection workflow*

Parallelism in RAC

Enforcing parallelism can sometimes produce fruitful results, including enhancing the performance of SQL queries, scaling up typical database manageability activities, and accelerating RMAN backup and recovery operations. In this section, we discuss the significance of parallelism and exactly how it helps RMAN backup, restore, and recovery operations in a RAC database.

When a database backup or restore operation is pioneered with multiple channels, Oracle determines and sets the degree of parallelism based on the number of allocated channels for that job; thus, any restore or recovery command (applying incremental backups) will span a corresponding number of slave processes to get the job done quickly. Moreover, for media recovery tasks, such as applying archive logs, Oracle sets the parallelism automatically based on the number of CPU resources on the node. In contrast, the PARALLEL hint with RECOVER statement will limit and overwrite the default degree of parallelism. Last but not least, when there are a huge number of archive logs to be applied as part of the recovery, parallelism indeed helps speed up the recovery time.

There have been some bugs reported of late with regards to parallel media recovery due to a few OS-related and storage-related issues. These can be avoided by disabling parallelism by specifying NOPARALLEL in conjunction with the RECOVER command.

Instance/Crash Recovery in RAC

A recovery operation on an Oracle database can be classified into two phases: media and instance/crash recovery. A database instance or crash recovery is needed when an instance starts up after an abnormal stop. Typically, when an instance fails or ends abruptly, it leaves the data files in an inconsistent state and also sets the current online redo group flag in an open state. In a RAC database, when one or more instances fail, the surviving instance does the instance recovery on behalf of the failed instances. When all instances of a RAC database fail, the instance that comes up first does the crash recovery on behalf of other instances. An instance recovery is automatically done by the Oracle System Monitor (SMON) background process without intervention from the DBA.

The prime objective of an instance/crash recovery is to ensure database consistency right after an instance failure. The SMON process in that context reads the online redo logs and applies the redo changes to the appropriate data blocks into their respective data files. An instance/crash recovery is done in two phases: rolling forward and rolling back. During the rolling-forward phase, all committed transactions (data blocks) that are not written yet to the concerned data files will be applied to the proper data files. In the rolling-back phase, all uncommitted transactions occurring before the instance crash will be rolled back, reading the undo segments to guarantee database consistency.

Instance Recovery

Though instance recovery mechanisms remain identical in nonclustered and clustered environments, the way that instance/crash recovery steps are carried out in a RAC database is slightly dissimilar from single-instance database instance recovery mechanics.

Contrary to a stand-alone database, when a dead (failed) instance is detected by a surviving instance in a RAC database, the SMON of the surviving instance initiates and does the instance recovery for the failed instance. Likewise, when multiple instances of a RAC database fail, even if the failure is of all instances except one, the surviving instance is responsible to perform recovery steps for all the failed instances. Figure 8-5 illustrates an instance recovery mechanism in RAC by a surviving node.

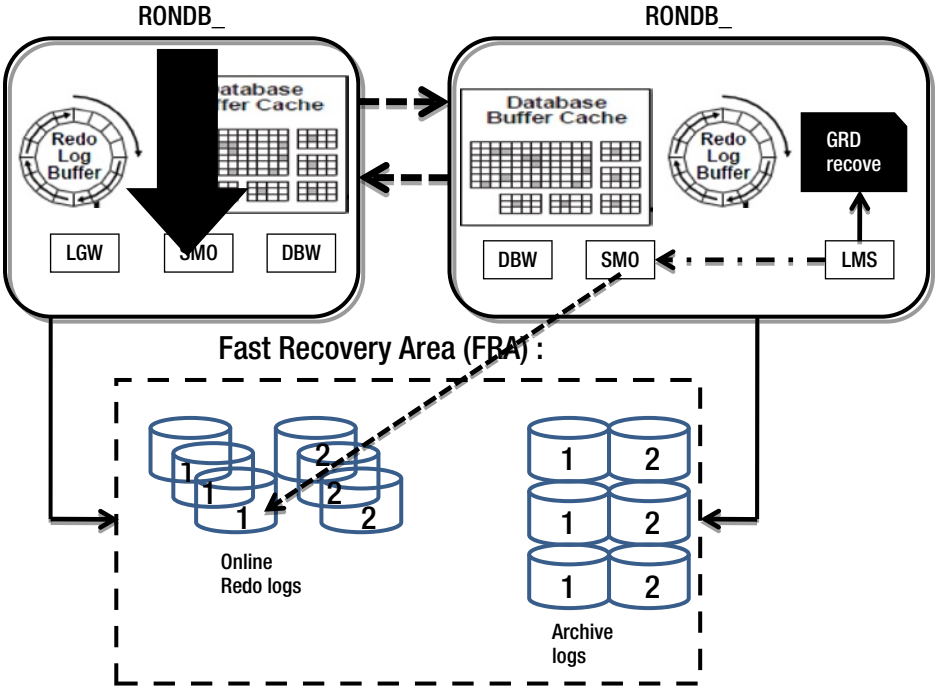


Figure 8-5. *RAC database FRA settings*

Crash Recovery

In the event of a RAC database having an all-instances failure, the instance that comes up first will perform the crash recovery on behalf of the database. As already explained in the section 'Shared Location for Redo and Archive Logs,' ensure that all instances are able to access each others' redo logs in order to perform the recovery operations.

Parallel Recovery

In this section, we will explain and provide some guidelines to enhance the recovery duration, key parameters, and other settings that reflect improving the recovery time significantly to ensure quick database or instance accessibility.

The duration required for an instance or crash recovery plays a pretty important role in instance and database accessibility to the end user for general usage. It is indeed essential to consider appropriate actions and steps to enhance the recovery time.

On a multiple CPU resource-based node, Oracle automatically sets the optimal degree of parallelism and does the instance recovery in parallel to run the recovery operations. You can also make use of the `RECOVERY_PARALLISM` and `PARALLEL_MIN_SERVERS` initialization parameters to enable parallelism. When the `RECOVERY_PARALLISM` parameter value is set to 2 or higher but can't be higher than the `PARALLEL_MAX_SERVER` value, Oracle automatically sets the degree of parallelism for instance/crash recovery operations. Set the parameter `RECOVERY_PARALLELISM` value either to 0 or 1 on a multi-CPU-based server to disable parallel recovery option.

Ensure that your platform supports async I/O and is enabled. It can also be enabled by setting the DISK_ ASYNCH_IO init parameter value to TRUE. The async I/O plays a key role in improving instance recovery during the first phase (first-pass log read) of recovery.

Another way of improving the instance recovery time is to have an adequate DEFAULT CACHE size of the DB BUFFER CACHE SIZE, as instance recovery utilizes about 50% of the DEFAULT CACHE size from the actual default

buffer cache size. You can verify whether the DEFAULT CACHE size is adequate or not. If the DEFAULT CACHE size is too low, you can see the related error message in the alert.log.

Monitor and View Instance Recovery Details

Instance recovery operation details are logged in the alert logs as well as in the SMON trace file. Refer to the alert.log file to find out the details, such as when the recovery began and when it was completed. For additional details on recovery, you can also refer to the SMON trace file.

Furthermore, the ESTD_CLUSTER_AVAILABILE_TIME column in the `GV$INSTANCE_RECOVERY` dynamic view shows the amount of time (in seconds) that the instance will be frozen. Therefore, the longer the time, the longer will be the estimated instance recovery.

Instance/Crash Recovery Internals in RAC

To provide a better understanding of the way that an instance recovery is performed in a RAC database, we constructed a small test scenario. As part of the test case, the following tasks were carried out on instance 1:

- A new table created
- A few records inserted and committed
- All records deleted with 'delete * from table_name'
- A couple of records inserted
- Instance 1 was aborted from the other SQL window

The following recovery action is performed by a surviving instance SMON background process for the dead instance:

1. Surviving instance acquires the instance recovery enqueue.
2. After Global Cache Services (GCS) are remastered, the SMON then reads the redo logs of the failed instance to recognize the resources (data blocks) that are needed for the recovery.
3. Global Resource Directory will be frozen after acquiring all necessary resources.
4. At this stage, all data blocks, except that needed for recovery, become accessible.
5. SMON starts recovering the data blocks identified earlier.
6. Immediately after the recovery, the individual data blocks become accessible.
7. All uncommitted transactions are also rolled back to maintain the consistency.
8. Once all the data blocks are recovered, the instance is fully available for end users.

Here is a walkthrough of the alert.log of the surviving instance that does the recovery for the failed instance. Most of the action is recorded in the context of instance recovery; you can refer to the log file to understand how things have been performed by the instance. All of the previously explained steps can be seen in the alert.log:

```
Reconfiguration started (old inc 12, new inc 14) <<<<<
List of instances:
 2 (myinst: 2)
 Global Resource Directory frozen      <<<<<
 * dead instance detected - domain 0 invalid = TRUE <<<<<
```

```
 Communication channels reestablished
 Master broadcasted resource hash value bitmaps
 Non-local Process blocks cleaned out
Sun Nov 18 11:39:19 2012
 LMS 0: 0 GCS shadows cancelled, 0 closed, 0 Xw survived
 Set master node info
 Submitted all remote-enqueue requests
 Dwn-cvts replayed, VALBLKs dubious
 All grantable enqueues granted
 Post SMON to start 1st pass IR      <<<<<
Sun Nov 18 11:39:22 2012
minact-scn: Inst 2 is now the master inc#:14 mmon proc-id:24545 status:0x7
minact-scn status: grec-scn:0x0000.00000000 gmin-scn:0x0000.00384a18 gcalc-scn:0x0000.00384a24
minact-scn: master found reconf/inst-rec before recscn scan old-inc#:14 new-inc#:14
Sun Nov 18 11:39:22 2012
Instance recovery: looking for dead threads
Beginning instance recovery of 1 threads
 Submitted all GCS remote-cache requests <<<<<
 Post SMON to start 1st pass IR
 Fix write in gcs resources
Reconfiguration complete
 parallel recovery started with 2 processes <<<<<
Started redo scan
Completed redo scan
 read 53 KB redo, 68 data blocks need recovery
Sun Nov 18 11:39:27 2012
Warning: recovery process P000 cannot use async I/O
Sun Nov 18 11:39:27 2012
Warning: recovery process P001 cannot use async I/O
Started redo application at
 Thread 1: logseq 20, block 42789 <<<<<
Recovery of Online Redo Log: Thread 1 Group 2 Seq 20 Reading mem 0 <<<<
  Mem# 0: +DG_LINQ/rondb/onlinelog/group_2.286.795350031
Completed redo application of 0.03MB
Sun Nov 18 11:39:27 2012
Completed instance recovery at
 Thread 1: logseq 20, block 42842, scn 3709030
 67 data blocks read, 68 data blocks written, 53 redo k-bytes read
Thread 1 advanced to log sequence 21 (thread recovery)
Redo thread 1 internally disabled at seq 21 (SMON)
```

Also, refer to the SMON trace file for an in-depth understanding of instance recovery.

How to Back Up and Restore Archive Logs

When an instance maintains a local destination in which to put its archive logs separately, then a separate channel must be allocated for each instance to back up the archive logs from the destination. However, when archive logs are placed at a shared location, and multiple channels are allocated, all allocated channels will then back up the archived logs. The same rule applies when deleting archive logs.

To restore one or more archive logs manually from a particular instance, the instance THREAD number must be specified in conjunction with the RESTORE ARCHIVE LOG command, as demonstrated in the following:

```
RMAN> restore archivelog logseq=200 thread 1;

              RMAN> restore archivelog ass 'from' word logseq=100 until
              logseq=110 thread 2;
```

The first example shows how to restore a specific archive log, and the second example shows how to restore a range of archived logs from a particular instance. When a THREAD clause is not specified, the following RMAN error will be encountered:

```
RMAN-20242: specification does not match any archived log in the repository
```

The following example demonstrates how to restore a set of archived logs from different instances at one time:

```
RMAN> run
{
 configure channel 1 device type disk connect='sys/password@rondb_1';
 configure channel 2 device type disk connect='sys/password@rondb_2';
 restore archivelog from logseq 14 until logseq 16 thread 1;
 restore archivelog from logseq 65 until logseq 70 thread 2;
}
```

Real-World Examples

The objective of this section is to provide working examples for some of the frequently used RMAN RAC scenarios. We don't aim to demonstrate a step-by-step procedure; we'll focus only on those core details that are suitable for RMAN in RAC environment.

Best Database Backup Strategies

In the following sections, we discuss some of the industry standard backup strategies.

Configure fast incremental backups: You need to connect to the database and enable the Block Change Tracking (BCT) option for optimized fast incremental backups on the database. Ensure that the BCT file is stored on a shared location so that all instances have the read and write ability on the file. The command provided enables the fast incremental backup option on the database:

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING;
```

After enabling the BCT feature, let's verify it by querying the `v$block_change_tracking` dynamic view:

```
SQL> SELECT * FROM v$block_change_tracking;
```

■ **Note** When BCT is configured, ensure that either the BCT file is available or the feature is disabled before opening the database after completing the database restore and recover procedure on the same host or on a new host.

Now it is time to develop a script for full database level 0 backup to initiate a base backup for subsequent incremental backups. The following script will have two channels allocated to TAPE, will back up the database along with the archive logs, and will also remove the archive logs after the backup:

```
--- rondb_full_db_bkp.sh
--- RONDB RMAN ONLINE FULL DATABASE BACKUP, LEVEL 0

rman{
 ALLOCATE CHANNEL ch00 TYPE 'SBT_TAPE' CONNECT sys/password@rondb_1;
 ALLOCATE CHANNEL ch01 TYPE 'SBT_TAPE' CONNECT sys/password@rondb_2;
SEND 'NB_ORA_CLIENT=host_name,NB_ORA_POLICY=RONDB_FULL_LEVEL0,NB_ORA_SERV=netbackup_master_server_name';
BACKUP DATABASE INCREMENTAL LEVEL=0 FORMAT 'u%ROMDB_FULL_LEVEL_0.bkp' plus
               ARCHIVELOG DELETE INPUT;
RELEASE CHANNEL ch00;
RELEASE CHANNEL ch01;
}
```

Now we will have an incremental cumulative database level 1 backup script. The following script performs an incremental cumulative backup to back up a database and archive logs together. Of course, the logs will be removed after the backup:

```
--- rondb_incr_db_bkp.sh
--- RONDB RMAN ONLINE INCREMENTAL DATABASE BACKUP, LEVEL 1

rman{
ALLOCATE CHANNEL ch00 TYPE 'SBT_TAPE' CONNECT sys/password@rondb_1;
               ALLOCATE CHANNEL ch01 TYPE 'SBT_TAPE' CONNECT sys/password@rondb_2;
SEND 'NB_ORA_CLIENT=host_name,NB_ORA_POLICY=RONDB_FULL_LEVEL0,NB_ORA_SERV=netbackup_master_server_name';
BACKUP DATABASE INCREMENTAL LEVEL=1 CUMULATIVE FORMAT
               'u%RONDB_FULL_LEVEL_0.bkp' plus ARCHIVELOG DELETE INPUT;
RELEASE CHANNEL ch00;
RELEASE CHANNEL ch01;
}
```

Having developed a full and incremental database backup, we typically schedule full database level 0 backup once a week followed by the incremental backup for the rest of the week. You can schedule the jobs through netbackup, in the crontab on a Unix platform, and so on.

If the application is highly OLTP and the database is tending to generate a huge number of archived logs, you may also develop a separate script scheduled at regular intervals to back up the archived logs and remove them after backup to avoid a database hang situation due to a lack of space left for archiving.

You should alter the preceding scripts to suit your environment and schedule the backup policy as per your organizational standards.

Restore a RAC Database to a Single-Instance Database

Notwithstanding clustered or nonclustered, the database restore procedure remains the same. However, the following additional post–database restore steps need to be carried out on the single-instance database after opening the database:

Proceed with disabling additional instances thread and remove redo groups.

```
SQL> SELECT instance,thread#,status,enabled FROM v$thread;
SQL> SELECT instance,group# FROM v$log WHERE thread=<thread_number>;
SQL> SELECT tablespace_name,contents FROM dba_tablespaces WHERE contents = 'UNDO';
```

Upon determining the thread number, redo groups, and undo tablescapes for the additional instances, now it is time to remove them from the database. The following set of SQL statements will achieve this:

```
SQL> ALTER DATABASE disable THREAD <thread_number>;
SQL> ALTER DATABASE drop logfile group <group_number>;
SQL> DROP TABLESPACE <undo_instance_specific> including contents and datafiles;
```

Assuming that a RAC database with three instances restored to a single-instance database, you need to follow the preceding procedure for instances 2 and 3 in order to have a clean single-instance database.

Configure RAC Standby Database

In the following example, we demonstrate how to configure a standby RAC database with two instances to a RAC primary database with two instances. The primary RAC database name used in the context is RONDB, and the standby RAC database is named RONDBS. Prior to getting into the real action, we will assume that the same version, platform, Clusterware, and Oracle RBDMS binaries are configured and operational at the standby location. The objective is to explain the procedure to set up a standby RAC database; the data guard configuration and the switchover/failover mechanism, however, is not part of our objective.

Once the standby environment is ready, run through the following tasks on the standby site:

- Create and mount necessary ASM DISKGROUPS across the nodes on which RAC standby instances are going to be placed.
- Configure a listener for the standby instances and a TNS connection pointing to the primary database instance on the standby nodes.
- After preparing SPFILE and the password file, and creating all necessary directory structures, start up the standby instance in NOMOUNT state: `SQL> STARTUP NOMOUNT.`
- Initiate the active standby database configuration procedure from the standby instance RMAN:

```
# rman connect target sys/password@prim auxiliary /
RMAN> duplicate target database for standby from active database
spfile
        parameter_value_convert 'RONDB','RONDBS'
        set db_unique_name='RONDBS'
        set fal_client='PRIM'
        set fal_server='STDBY'
        set standby_file_management='AUTO'
        set log_archive_config='dg_config=(RONDBS,RONDB)'
                      set log_archive_dest_1='service=RONDBS ASYNC
                      valid_for=(ONLINE_LOGFILE,PRIMARY_ROLE) db_unique_name=RONDBS';
```

- Once the standby database is successfully configured, create standby redo logs on the standby instance of the same number (plus one additional group) and the same size as the primary database redo logs. It is highly recommended to have an additional standby redo group for each instance, in contrast to the primary database instance redo groups count. Presume that a primary instance has three redo groups and each redo member is sized 100M; therefore, we will have four standby redo groups for each standby instance with the same redo log member size.

Run the following SQL statements on the standby instance:

```
   SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 1
GROUP 11 SIZE 100M,
GROUP 12 SIZE 100M,
GROUP 13 SIZE 100M,
GROUP 14 SIZE 100M;

   SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 2
GROUP 15 SIZE 100M,
GROUP 16 SIZE 100M,
GROUP 17 SIZE 100M,
GROUP 18 SIZE 100M;
```

- Now it is time to turn on auto-managed recovery and start applying logs in real time. Run the following command on the standby instance:

  ```
  SQL> alter database recover managed standby database using current logfile
   disconnect from session;
  ```

- Finally, add the second standby instance and configure standby database details in the OCR. Run through the following set of statements from the command prompt:

  ```
  # srvctl add database -d RONDBS -o /u00/app/oracle/product/12.1.0/db_1 -r PHYSICAL_STANDBY
  # srvctl add instance -d RONDBS_1 -n rac1
  # srvctl add instance -d RONDBS_2 -n rac2
  ```

- Configure a TNS connection pointing to the first standby instance and then run through the following steps on the primary database, from the first instance:

  ```
  SQL> ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(RONDB,RONDBS)' SCOPE=BOTH;
  SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=<ARCHVIE_LOCATION>
  VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=RONDB'
  SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2=SERVICE=<RONDBS_TNS>
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)  DB_UNIQUE_NAME=RONDBS'
  ```

Manage RMAN with OEM Cloud Control 12c

As mentioned earlier, RMAN can be managed and invoked through a command-line prompt interface or through an OEM (also referred to as OEM Cloud Control 12c). We will run through some of the OEM screenshots here to manage and schedule a database backup and recovery operation through OEM.

- Log in to OEM Cloud Control 12c using the predefined login credentials (Figure 8-6).

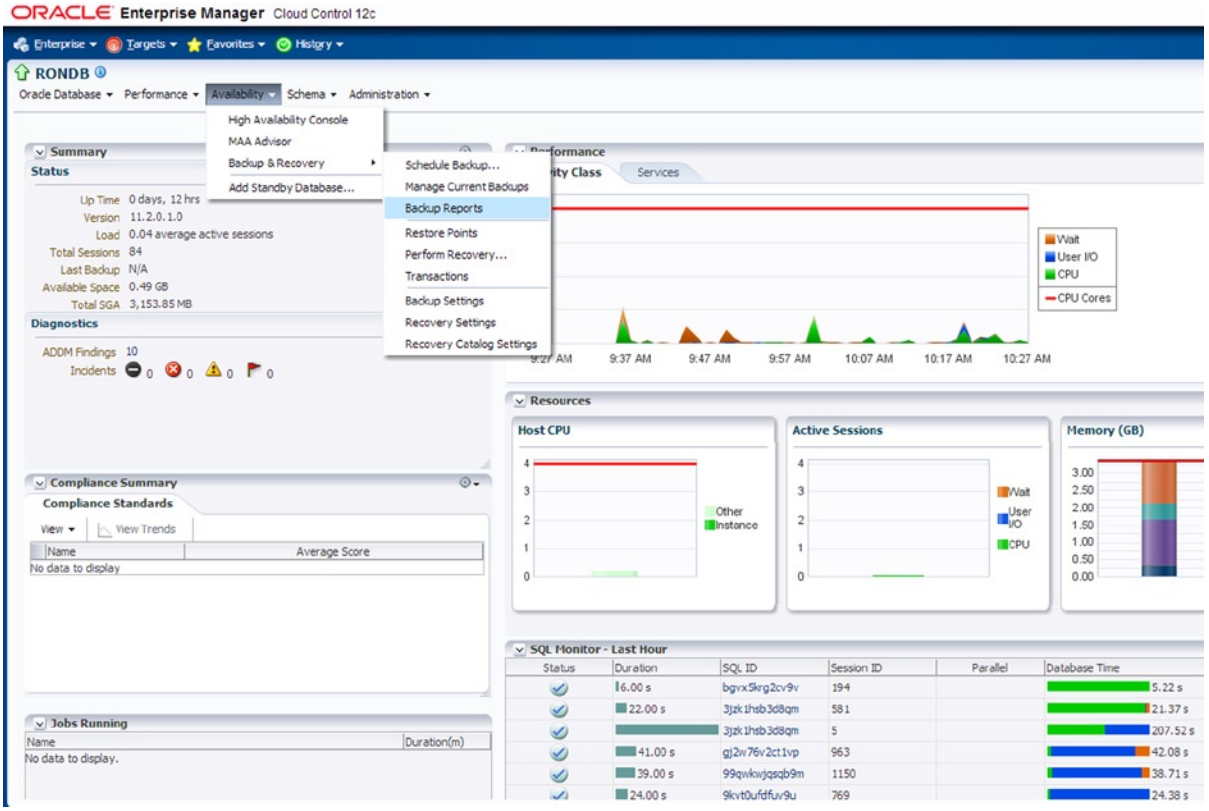


Figure 8-6. *OEM Cloud Control 12c availability screen*

- From the **Targets** drop-down list, select the **Databases** option.
- Click the desired database name from the list displayed under the Databases list.
- Go to the **Backup & Recovery** setting from the **Availability** drop-down list, which will display the following options: `Schedule Backup, Manage Current Backups, Backup Reports, Restore Points, Perform Recovery, Transactions, Backup Settings, Backup/Recovery/Catalog settings.`

To configure Device and Tape settings, such as parallelism, backup location, backup type: default, compression or image copy, and to set tape settings, go to the **Device** tab under **Backup Settings**, as shown in Figure 8-7.

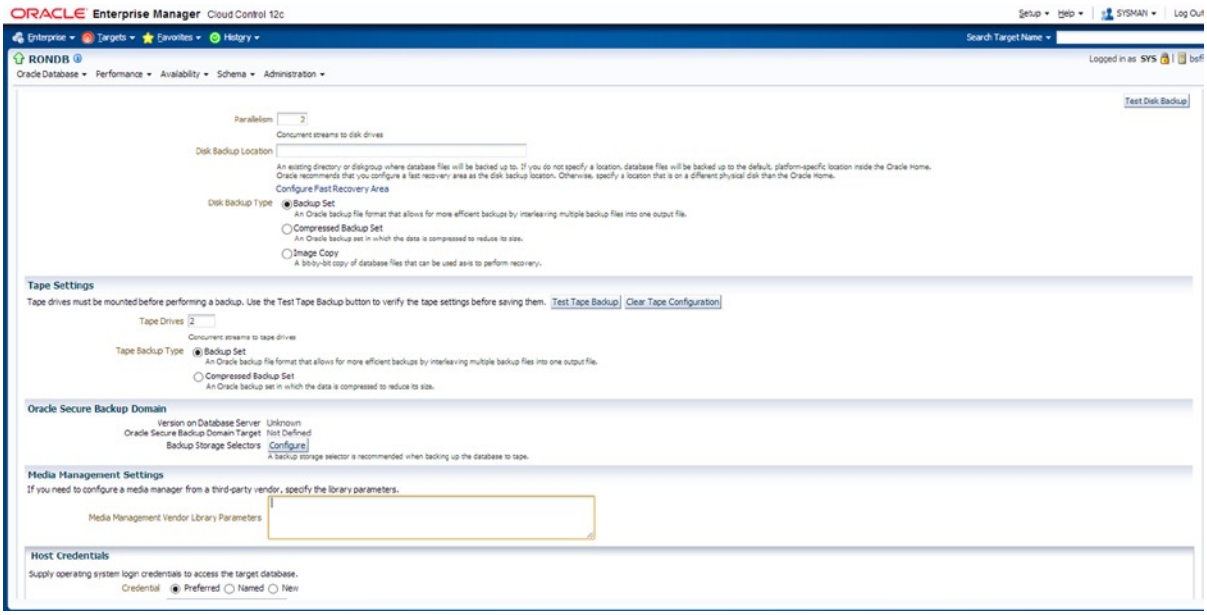


Figure 8-7. *OEM Cloud Control 12c Backup_setting_device screen*

The **Backup Set** tab provides options to adjust the backup piece size that is useful while backing up a large database and wanting to control the backup piece size. Additionally, the compression algorithm gives choices to set different levels of algorithm; that is, `BASIC`, `LOW`, `MEDIUM`, and `HIGH`, subject to separate licensing. When the **Optimizer for Load** choice turned on, it controls compression processing by optimizing CPU usage, as shown in Figure 8-8.

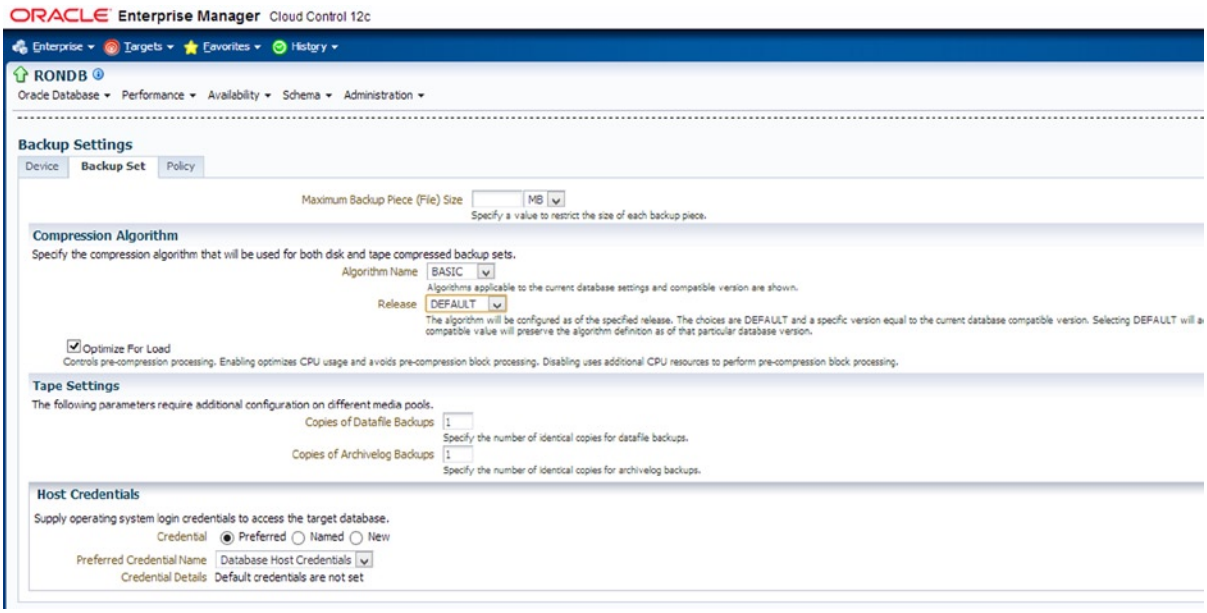


Figure 8-8. *OEM Cloud Control 12c Backup_setting_backup_set screen*

The **Policy** tab (Figure 8-9) offers the following settings:

- Controlfile, SPFILE auto backup option on any structural changes and with every database physical backup. In addition, you can specify the default destination for these auto backups.
- Flexibility to skip the read only and offline datafiles that were backed up once.
- Can turn on BCT feature to take faster incremental backups.
- Tablespaces can be excluded as part of the complete database backups.
- You can also set the retention policy for backups and archive logs according to your requirements.

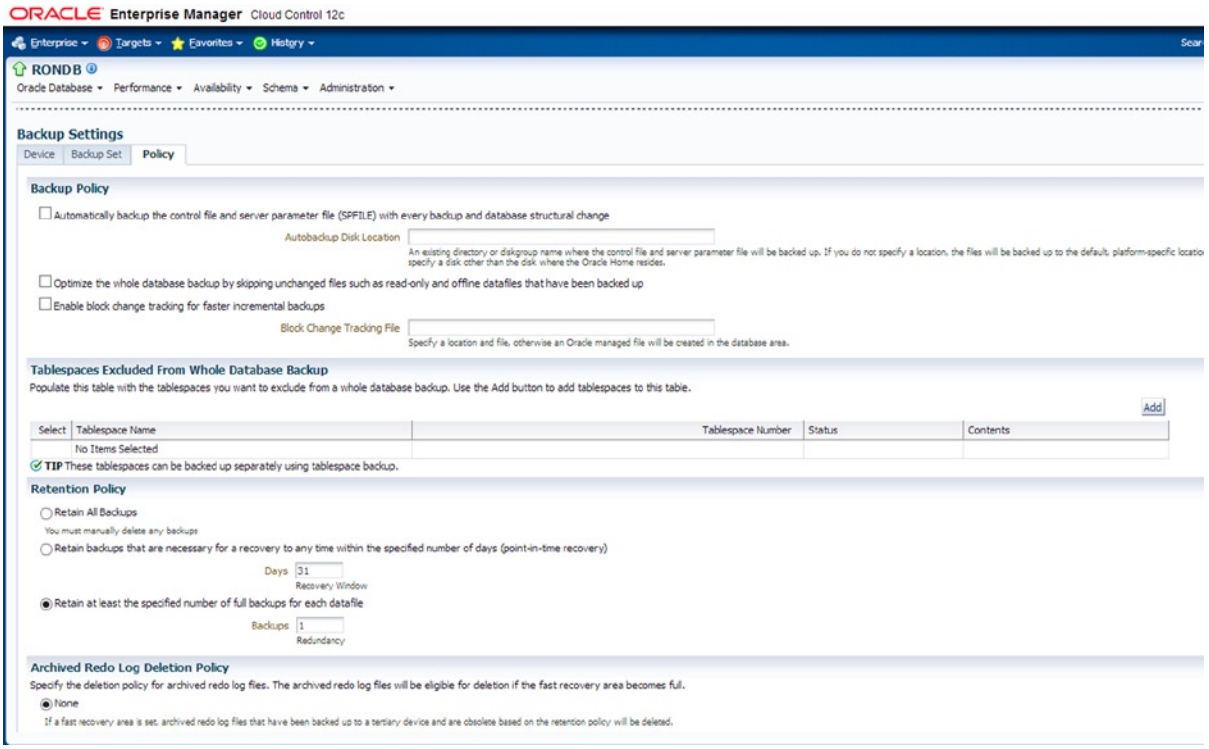


Figure 8-9. *OEM Cloud Control 12c Backup_setting_policy screen*

Click the **OK** button at the bottom right to save all backup settings done in the preceding steps.

The **Recovery Settings** comes with options to adjust the Mean Time To Recovery (MTTR) values to enable the fast-start checkpoint feature, which helps during instance/crash recovery. In addition, FRA settings can be managed and adjusted. Click the Apply initialization parameter option to save changes in SPFILE only. To apply the change with immediate effect, click the Apply button displayed at the bottom right. For all details, see Figure 8-10.

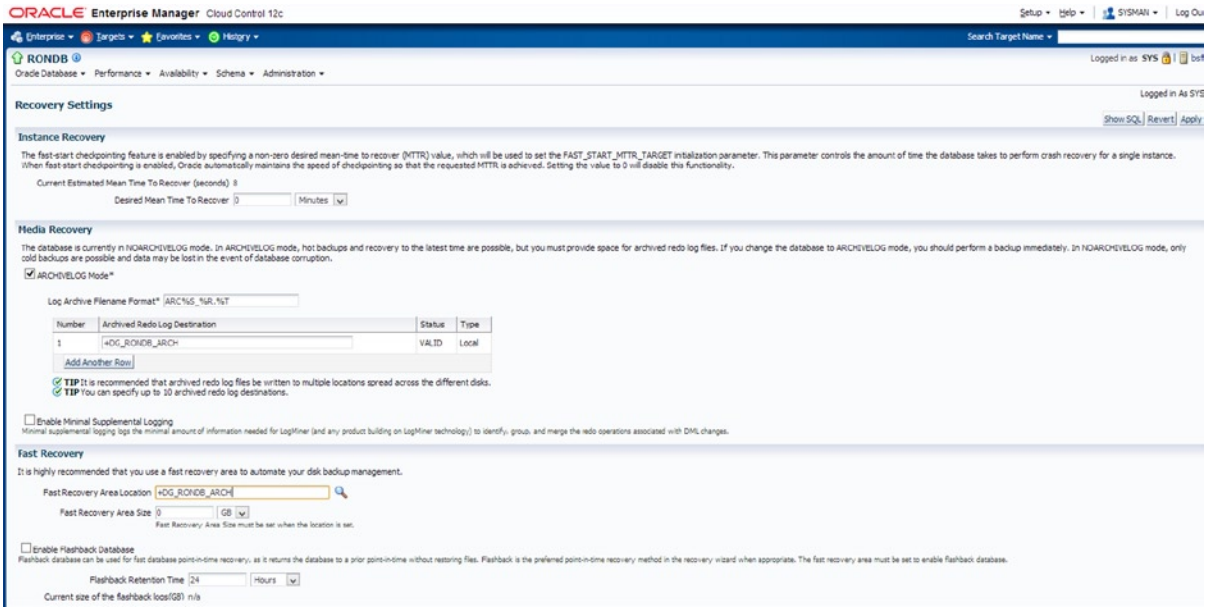


Figure 8-10. *OEM Cloud Control 12c recovery settings screen*

Schedule Backup (Figure 8-11) offers automated and customized backup strategy settings. The customized backup lets you schedule the backup for full database, tablespaces, datafiles, archived logs, and so on, whereas the automated option provides auto backup management.

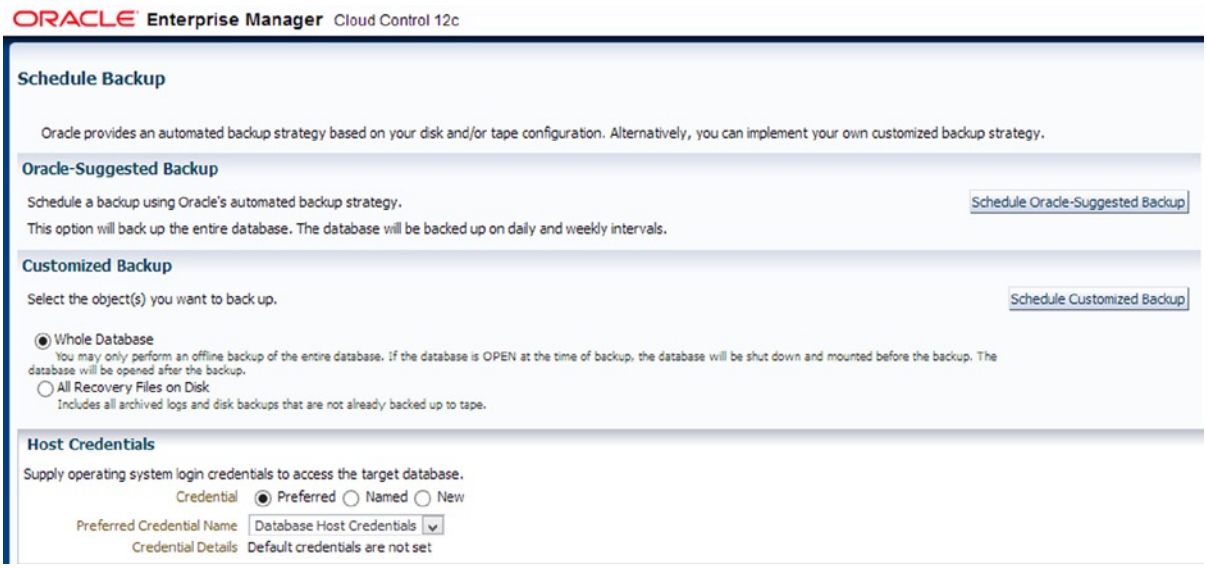


Figure 8-11. *OEM Cloud Control 12c schedule backup screen*

The **Backup Sets** option under the **Manage Current Backups** section provides detailed reports to existing backups that include backup piece completion time, where the backup piece is placed, and the status. Simultaneously, there are options to run through crosscheck, delete, and validate the obsolete, expired, and valid backup pieces. Have a look at Figure 8-12.

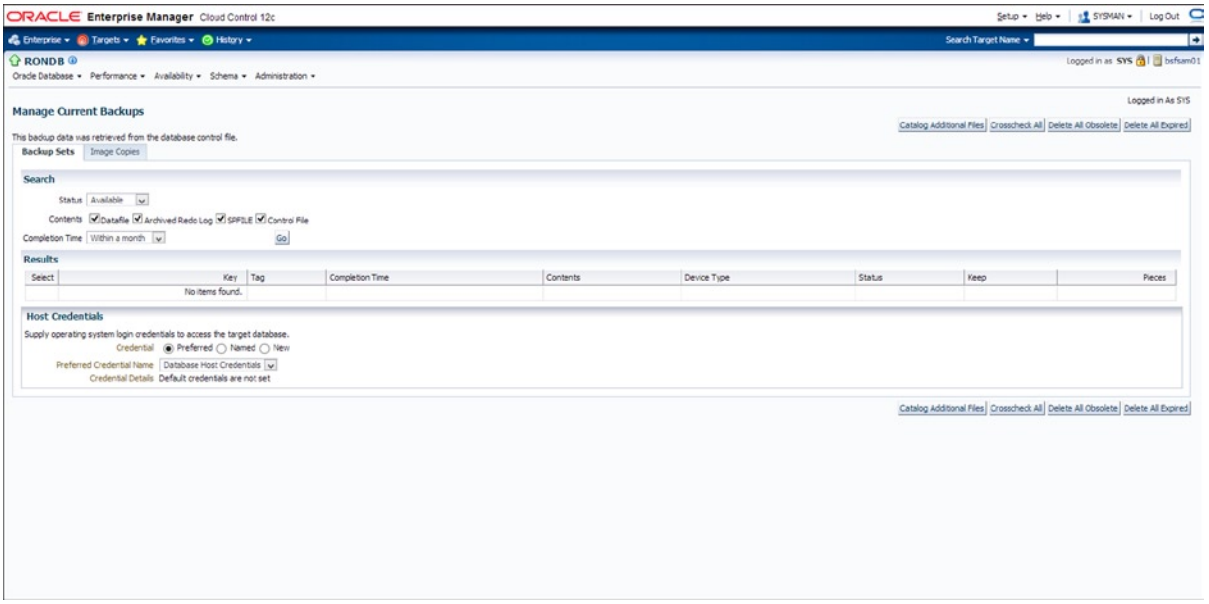


Figure 8-12. *OEM Cloud Control 12c manage current backups screen*

The **Perform Recovery** section provides various recovery scenarios: complete database, datafiles, tablespaces, tables, archived logs, etc. Recovery operation types include complete recovery and point-in-time recovery. Look at Figures 8-13 and 8-14.

Figure 8-13. *OEM Cloud Control 12c manage recovery_fulldb_settings screen*

Figure 8-14. *OEM Cloud Control 12c manage recovery_table_settings screen*

OCR recovery

OCR is beyond a doubt one of the most critical components of Oracle Clusterware, and its uninterrupted availability is necessary to the cluster resources function. Keeping in mind this criticality, Oracle offers several options to protect the OCR file from physical or logical corruptions, unintentional human errors, and single points of failure. The OCR file is automatically backed up every 4 hours by Oracle Clusterware and can also be backed up manually on demand. To avoid a single point of failure, consider multiplexing the file up to a maximum of five copies.

You really need to understand and be aware of all possible methods to protect and recover OCR from different failures. In this section, we shall highlight various OCR recovery scenarios.

Let's verify the existing OCR details. For that, you need to log in as root and execute the following to view OCR details:

```
# ocrcheck
```

Logical corruption verification will be skipped if the preceding command is executed as a nonprivileged user.

To view the OCR file name and path, use the following command:

```
# ocrcheck -config
```

Automatic/manual backup details are listed using the following command:

```
# ocrconfig -showbackup
```

The following are a few OCR restore procedures that can be used in the event of all OCR files being either corrupted or lost.

Scenario 1: The following demonstrates a procedure to restore OCR from autogenerated OCR backups:

As the root user, get the auto backup availability details using the following command:

```
# ocrconfig –showbackup
```

Stop Clusterware on all nodes using the following command as root user:

```
            # crsctl stop crs [-f]

-- Use the –f option to stop the CRS forcefully in the event that the crs stack couldn't be
stopped normally due to various errors.
```

As root user, restore the most recently valid backup copy identified in the preceding step. Use the following restore example:

```
# ocrconfig –restore backup02.ocr
```

Upon restore completion, as root user, bring up the cluster stack and verify the OCR details subsequently on all nodes using the following commands:

```
# crsctl start crs
# cluvfy comp ocr –n all -verbose
```

Scenario 2: The following demonstrates a procedure to recover OCR in an ASM diskgroup, assuming that the ASM diskgroup got corrupted or couldn't be mounted:

After locating the most recent valid automatic or manual OCR backup, stop the cluster on all nodes. To stop the cluster, use the command mentioned in the previous procedure.

Start the clusterware in exclusive mode using the following command as root user:

```
# crsctl start crs –excl -nocrs
```

Connect to the local ASM instance on the node and recreate the ASM diskgroup. Use the following SQL examples to drop and re-create the diskgroup with the same name:

```
SQL> drop diskgroup DG_OCR force including contents;

SQL> create diskgroup DG_OCR external redundancy disk 'diskname' attribute
                'COMPATIBLE.asm'='12.1.0';
```

Upon ASM disk re-creation, restore the most recent valid OCR backup. Use the example explained in the previous scenario.

If the voting disk exists in the same disk group, you also need to re-create the voting disk. Use the following example:

```
# crsctl replace votedisk +DG_OCR
```

After successfully completing the preceding steps, shut down the cluster on the local node and start the clusterware on all nodes subsequently.

Scenario 3: In the following example, we demonstrate the procedure to rebuild an OCR file in the event of all files becoming corrupted and when there is no valid backup available.

To be able to rebuild the OCR, you need to first unconfigure and then reconfigure the cluster. However, noncluster resource information, such as database, listener, services, instance, etc. needs to be manually added to the OCR. Therefore, it is important to collect the resource information using the crsctl, srvctl, oifcfg, etc., utilities.

To unconfigure the cluster, run the following example as root user across all nodes in sequence:

```
#$GI_HOME/crs/install/rootcrs.pl –deconfig –force –verbose
```

Upon successfully executing the preceding on all nodes, the following needs to be run on the first node of the cluster:

```
#$GI_HOME/crs/install/roortcrs.pl –deconfig –force –verbose –lastnode
```

After unconfiguring the cluster, you will now have to configure the cluster as root user with the following example:

```
#$GI_HOME/crs/config/config.sh
```

The `config.sh` invokes Grid Infrastructure configuration framework in Graphical User Interface (GUI) mode, and you need to provide the appropriate input through pages that displayed. Finally, you will have to run the root.sh to complete the configuration.

After recovering the OCR file from various failure scenarios, run through the following set of postrecovery steps to verify the OCR file integrity and cluster availability on all nodes:

```
# ocrcheck
# cluvfy comp ocr-n all –verbose
# crsctl check cluster -all
```

Summary

In a nutshell, this chapter summarized and offered the most useful tools, tips, and techniques to design and deploy optimal database backup and recovery procedures in a RAC environment using the RMAN tool. In addition, the internal mechanics behind instance and crash recovery operations in a RAC environment were explained in great detail. The chapter concluded by discussing and demonstrating various OCR recovery scenarios.

CHAPTER 9

Network Practices

by Riyaj Shamsudeen, Kai Yu

Reliable and scalable network architecture is a critical element in Real Application Cluster (RAC) infrastructure design. The availability of applications in a RAC database depends heavily on the reliability of the network infrastructure. Many design aspects of a RAC cluster's underlying network should be considered before implementing a cluster in a production environment, since making changes to the RAC cluster after production go-live is never easy. As nodes in a cluster are monitored by exchanging short messages between the Clusterware daemons, any network disconnect can lead to messaging failures, resulting in node restart or rebootless[1] Clusterware restarts.

Cache fusion traffic flows through the private network too. Latency in private network traffic will lead to elevated wait times for global cache events. Worse, dropped packets in the private network, due to problems in network configuration, can result in much longer global cache event waits.

Types of Network

A typical organization has different kinds of networks, and network traffic usually is separated into different network segments, possibly employing many physical infrastructure segments.

The *public network* is used primarily for general traffic such as applications connecting to a database, users executing queries from an application screen, etc. For a public network, default network attributes are sufficient as the default values are designed to match the traffic characteristics of public network traffic. Further, database listeners listen on an IP address on the public network. A *storage network* is primarily used for communication between servers and storage servers to support network-based storage. A *private network* is a special type of network used for communication among cluster nodes in a RAC cluster.

Many organizations also employ a *backup network* primarily carrying backup data of applications and databases. These networks also carry a huge amount of network traffic and might need further tuning to improve backup speed.

While many discussions in this chapter apply to all types of networks, the primary focus is on private networks. Logically, there are two types of network packets flowing through a private interconnect: big packets with a database block as payload, and small packets with short messages (~400 bytes) as payload. A private network must be tuned to operate better for bigger workloads and to have very low latency. In most cases, private network traffic is contained within a short network segment, that is, within a switch or two.

[1]Node reboots can be time-consuming and Clusterware tries to avoid them by restarting itself. Of course, a Clusterware restart will cause databases and listeners to restart.

Network Layers

Any network is implemented as a set of layers, each layer assigned to a specific task. To understand performance metrics of a network, you need to know about the flow of network packets through various network layers. Figure 9-1 shows an overview of packet transfer through five network layers in a RAC cluster. Clusterware processes (such as CSSD, CRSD, etc.) use TCP/IP to send streams of data to other nodes. Database server processes predominantly use User Datagram Protocol (UDP) to send data segments to other nodes.

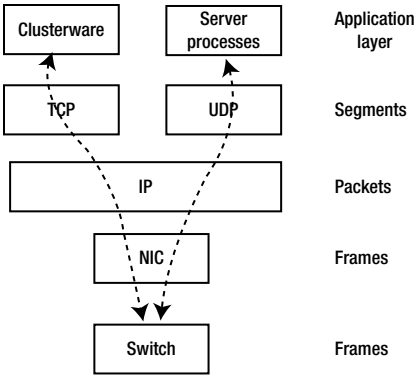


Figure 9-1. *Network layers*

Each layer is implemented as a kernel function, and data flows through the layers in opposite directions when sending and receiving packets. Both TCP and UDP layer kernel functions are implemented over IP layer functions. An IP layer function calls a device driver function specific to the particular hardware in question to send packets through physical interfaces.

Figure 9-2 shows how a stack of functions is called to send or receive a datagram. In Figure 9-2, the application layer populates the buffers to send and calls a network system function (such as udp_sendmsg, tcp_sendmsg, etc.) to send a data stream. The system calls, in turn, invoke the next layer kernel function in the call hierarchy, which calls the next layer function, and so on. Essentially, every layer performs a specific task and calls the next layer function in the hierarchy to do additional processing. For example, a UDP protocol handler fills the UDP header for source and destination port, and chunks the data stream into segments. UDP (or TCP) layer functions call IP layer functions to perform IP layer sanity checks. IP layer functions break the segments into packets, and add IP header details. Then, IP layer function invokes the device driver functions to send the packets. Device driver sends the frames through the network interface, wire, and switch to the destination.

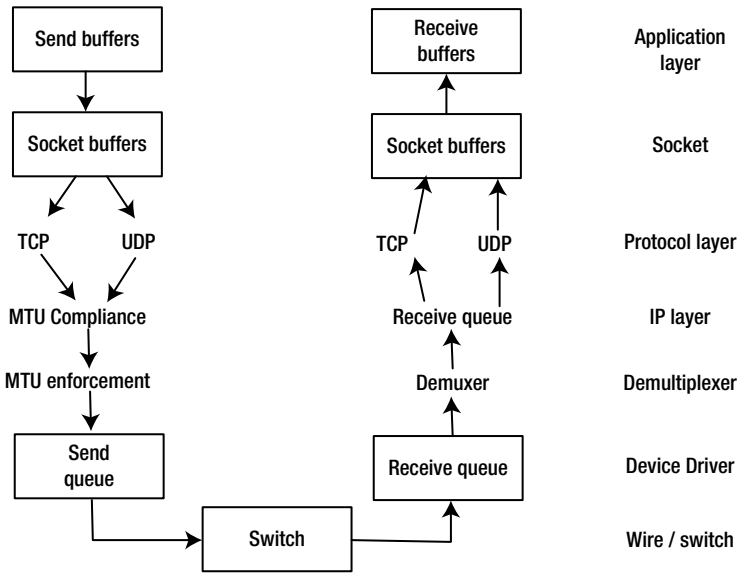


Figure 9-2. *Network layer kernel functions*

At the receiving node, network interface receives the frames, copies the packets into a buffer, and raises a softirq[2] to the CPU. After raising a softirq, the network interface will proceed without blocking and continue processing incoming frames. A kernel thread is scheduled in CPU to process the interrupt request and calls an IP layer function to process the packets. Then the kernel thread calls next layer functions such as tcp_recv, udp_recv, etc. After receiving all IP packets of a UDP or TCP segment, the packets are reassembled to create a TCP or UDP segment, and reassembled packets are available in socket buffers. If there is a process listening on the port specified in UDP or TCP segment, then that process is scheduled on the CPU to read from the socket buffers.[3]

IP layer functions negotiate MTU decisions too. MTU-specific processing is discussed in the Jumbo Frames section.

■ **Note** Nomenclature is important when you talk about networks. Network layer transmission is measured in terms of *frames*, IP layer transmission is measured in terms of *packets*, and in the TCP/UDP layer transmission is measured in terms of *segments*. The application sends a data stream or datagram depending upon whether TCP or UDP is used. Use of the correct word when referring to a network transmission will greatly improve clarity to network engineers. Also, network speed is measured in terms of bits, whereas DBAs typically specify everything in bytes.

In a RAC cluster, many kernel parameters for tuning the network must be set optimally. These tunables can increase transmission efficiency in various network layers.

[2]Softirq is an interrupt request to a CPU and it is a software-initiated interrupt request. Hardware irq is initiated by hardware. In layman's terms, softirq is equivalent to adding a request to a CPU queue. After the end of current timeslice, a softirq is serviced by the CPU.
[3]This discussion is provided to understand basics about network function calls and should not interpreted as an actual algorithm. For that matter, every kernel behaves differently.

Protocols

There are a few higher-level protocols such as TCP, UDP, and RDS used in a typical RAC cluster.[4] This section covers various protocols and their pros and cons.

TCP/IP is a stateful protocol. A connection between the sender and receiver must be established before sending a segment. Transmission of every segment requires an acknowledgement (TCP ACK) before a transmission is considered complete. For example, after sending a TCP/IP segment from one IP address and port number to another IP address and port number, kernel waits for an acknowledgement before declaring that transmission as complete.

UDP is a stateless protocol. No existing connection is required to send a datagram. A transmission is considered complete as soon as frames leave the network interface. No ACK required at all; it is up to the application to perform error processing. For example, if a UDP packet is lost in transmission, RAC processes re-request the packet. The UDP protocol layer is built upon the IP layer. However, both UDP and TCP/IP protocols have the overhead of double copy and double buffering, as the segments can be sent only after copying the datagrams from the user space to kernel space and received packets are processed in the kernel space and copied into user space.

The RDS (Reliable Datagram Socket) protocol requires specific hardware (InfiniBand fabric) and kernel drivers to implement. With the RDS protocol, all error handling is offloaded to the InfiniBand fabric, and the transmission is considered complete as soon as the frame reaches the fabric. The RDS protocol is used in the Exadata platform, providing lower latency and lower resource usage. Similar to UDP, there is no ACK mechanism in the RDS protocol. Further, RDS is designed as a zero-copy protocol, and the messages can be sent or received without a copy operation. The RDS protocol does not use IP layer functions and bypasses the IP layer completely.

The UDP protocol is employed for cache fusion on Unix and Linux platforms. On Exadata platforms, the RDS protocol is used for cache fusion. You can implement the RDS protocol on non-Exadata platforms too. At the time of writing, InfiniBand fabric hardware and RDS kernel drivers are available in some flavors of Unix and Linux. There are also vendor-specific protocols: for example, the LLT protocol is used for cache fusion with Veritas SFRAC.

Clusterware uses TCP/IP for heartbeat mechanism between nodes. While UDP stands for User Datagram Protocol, it is sometimes, in a lighter vein, referred as the Unreliable Datagram Protocol. However, it does not mean that UDP will suffer from data loss; thousands of customers using UDP in the Unix platform are proof that UDP doesn't affect the reliability of an application. In essence, the UDP protocol is as reliable as the network underneath.

While there are subtle differences between UDP and other protocols, UDP processing is simpler and easier to explain. In Figure 9-3, a UDP function stack is shown on a Linux platform. It is not necessary to understand the details of these function calls (after all, this chapter is not about network programming); just a high-level understanding of function execution flow is good enough. Application processes call udp_send system call; udp_sendmsg calls IP layer functions; IP layer function calls the device driver functions, recursively. On the receiving side, kernel threads call IP layer functions and then UDP layer functions, and then the application process is scheduled in the CPU to drain the socket buffers to application buffers.

[4]Other protocols may be in use in a third-party cluster. For example, a RAC database uses LLT protocol in a Veritas SFRAC cluster.

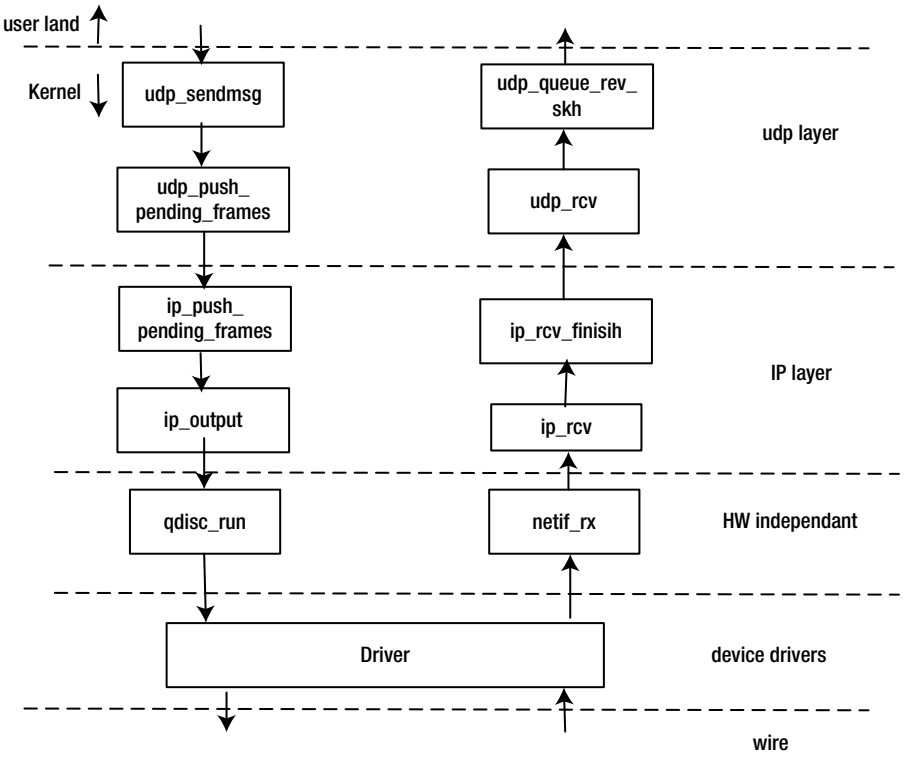


Figure 9-3. *UDP function execution stack*

Network system calls are executed in kernel mode. So, a large amount of network traffic can lead to higher CPU usage in kernel mode. In RAC, a large amount of cache fusion traffic can lead to higher CPU usage in kernel mode.

RDS protocol

RDS protocol is a low-latency, low-resource usage protocol. In UDP and TCP protocol processing, data is copied from user space to kernel space and sent downstream through the layers of network. In contrast, RDS protocol can send buffers directly from user space, avoiding a copy operation. On the receiving side, a copy operation is also avoided by buffering the data directly to user space buffers.

Figure 9-4 shows the flow of data with InfiniBand fabric in play. TCP and UDP packets can flow through InfiniBand infrastructure employing an additional *IP-over-IB* processing layer. The RDS protocol bypasses IP layers and sends the packets directly through Host Channel Adapter and then through the fabric. Due to reduction in processing layers, RDS is designed to be a lower-resource-usage protocol.

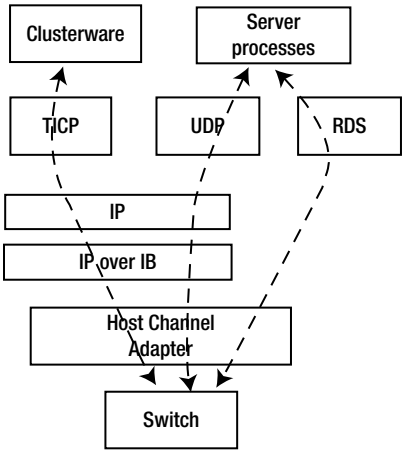


Figure 9-4. *RDS protocol layers*

The RDS library must be linked to both Database and Grid Infrastructure binaries. Clusterware uses the TCP/IP protocol in InfiniBand architecture too. The RDS protocol is supposed to be the lowest-resource-usage protocol, with a latency on the order of 0.1 ms.

Relinking Protocol Library

Oracle RAC provides protocol-specific library files to link with binary. For example, libskgxpg.so file implements the UDP protocol, and libskgxpr.so file implements the RDS protocol. So, to use the UDP protocol, Oracle Database binary must be linked with the libskgxpg.so library. But, instead of linking directly to the protocol-specific library file, the protocol library file is copied to the libskgxp11.so file, and then Oracle binary is linked to the libskgxp11.so file.

Database and ASM software are linked with a protocol library for cache fusion. For example, in Oracle Database version 11g, libskgxp11.so is the library file linked with the database binary. The following `ldd` command output in the Solaris platform shows that the libskgxp11.so file is linked to Oracle binary.

```
$ ldd oracle|grep skgxp
libskgxp11.so =>         /opt/app/product/11.2.0.2/lib/libskgxp11.so
```

In release 12c, an updated library file is used, and the following ldd command output shows that the libskgxp12.so library file is linked.

```
$ ldd oracle|grep skgxp
libskgxp12.so =>         /opt/app/product/12.1.0/lib/libskgxp12.so
```

The relink rule in the RDBMS makes file ins_rdbms.mk of Oracle database binary can be used to specify a protocol for cache fusion. The following ipc_g rule enables the use of the UDP protocol for cache fusion. Execution of this rule through a make command will remove file libskgxp11.so and copy UDP-specific library file libskgxpg.so to libskgxp11.so.

```
ipc_g:
        -$(RMF) $(LIBSKGXP)
        $(CP) $(LIBHOME)/libskgxpg.$(SKGXP_EXT) $(LIBSKGXP)

rm –f /opt/app/product/12.1.0/lib/libskgxp12.so
cp /opt/app/product/12.1.0/lib/libskgxpg.so /opt/app/product/12.1.0/lib/libskgxp12.so
```

To use a specific protocol, you may need to relink Oracle binary using the relink rule. For example, to relink the binary to UDP protocol, you can use the ipc_g rule.

```
$ make –f ins_rdbms.mk ipc_g oracle
```

Similarly, relink rule `ipc_r` and the `libskgxpr.so` library file are used for the RDS protocol. If you want to eliminate protocol completely, then `ipc_none` rule can be used, which links a dummy stub library `libskgxpd.so` file.

You can identify the current protocol linked to Oracle binaries by using the skgxpinfo utility. Relink rule ipc_relink uses this utility to identify the current protocol library linked to the binary.

```
$ skgxpinfo
Udp
```

By default, a single-instance binary is relinked with the libskgxpd.a file, which is a dummy driver.

```
$ strings libskgxpd.a |grep Dummy
Oracle Dummy Driver
```

ARP

While the Address Resolution Protocol (ARP) is not specific to a RAC database, a cursory understanding of ARP is essential to understand failover or routing problems. Consider that a process listening on 172.29.1.13 running in node 1 wants to send a packet to the IP address 172.29.2.25 in node 2. The network layer will broadcast a message like "Who has 172.29.2.25 IP address?". In this example, node 2 will respond with the MAC address of the interface configured with the IP address of 172.29.2.25. Mapping between the IP address and MAC address is remembered in the ARP cache in OS, switch, and router. With this mapping, packets are transferred between two MAC addresses or two ports in the switch for private network traffic.

In case of IP failover to a different interface, the ARP cache must be invalidated since the IP address has failed over to another interface with a different MAC address. Clusterware performs an ARP ping after a failover reloading the ARP cache with failed-over configuration. In some rare cases, the ARP cache may not be invalidated, and an invalid IP address to the MAC address may be remembered in a cache. This can lead to dropped packets or, worse still, node restarts due to dropped packets, even though the IP address has failed over to a surviving interface.

IPv4 vs. IPv6

IPv4 (IP version 4) is used for most electronic communication at the time of writing. As discussed earlier, higher-level protocols such as TCP, UDP, etc., use IP. In the all-too-familiar IP addressing scheme, for example 8.8.8.8,[5] a 32-bit IP addressing scheme is used; therefore, the theoretical maximum number of possible IP addresses is 2^{32} or about 4 billion IP addresses. That might seem a lot of IP addresses, but we are running out of public IP addresses for the Internet. IPv6 was introduced to provide a new IP addressing scheme.

IPv6 uses a 128-bit IP addressing scheme, so 2^{128} (or about 3×10^{38} ; some 340 billion billion billion billion) IP addresses can be allocated using the IPv6 format. Migrating from IPv4 to IPv6 is not that easy, as the majority of routers and servers do not support IPv6 at this time. However, eventually, the IPv6 addressing scheme will be predominant. An IPv6 IP address consists of eight groups of four-digit hexadecimal characters, separated by colons. Here is an example of an IPv6 address:

```
2200:0de7:e3r8:7801:1200:7a23:2578:2198
```

[5]IP address 8.8.8.8 is assigned to Google, and it is the IP address of Google's public DNS server. (In numerology, number 8888 means that there is light at the end of tunnel!)

Moreover, trailing zeroes can be omitted in the IPv6 protocol. For example, the following is also a valid IPv6 IP address and trailing zeroes have been omitted.

```
1000:0ee6:ab78:2320:0000:0000:0000:0000
1000:0ee6:ab78:2320
```

Until version 11gR2, IPv6 was not supported by RAC. Version 12c supports IPv6 in Public IP address, Virtual IP (VIP) addresses, and SCAN IP address. IPv6 is not supported in Private network, though.

VIPs

A VIP is an IP address assigned to a cluster node, monitored by Oracle Clusterware, which can be failed over to a surviving node in case of node failure. To understand the importance of VIP, you need to understand how a connection request is processed. Consider a connection request initiated by a client process connecting to a listener listening on IP address 10.21.8.100 and port 1521. Normally, a connection request will connect to the listener listening on that IP address and a port number, listener spawns a server process in the database server, and the client process will continue processing the data using the spawned process. However, if the database server hangs or dies, then the IP address will not be available (or will not respond) in the network. So, a connection request (and existing connections too) will wait for a TCP timeout before erroring out and trying the next IP address in the address list. The TCP timeout can range from 3 minutes to 9 minutes depending upon platform and versions. This connect timeout introduces unnecessary delays in connection processing.

In a RAC cluster, a VIP[6] is assigned to a specific node and monitored by the Clusterware, and each node can be configured with one or more VIPs. Suppose that `rac1.example.com` has VIP 10.21.8.100. If the node `rac1.example.com` fails, then the Clusterware running on other surviving cluster nodes will immediately relocate VIP 10.21.8.100 to a surviving node (say `rac2.example.com`). Since an IP address is available in the network, connection requests to the IP address 10.21.8.100 will immediately receive a CONN_RESET message (think of it as a No listener error message), and the connection will continue to try the next address in the address list. There is no wait for TCP timeout, as the IP address responds to the connection request immediately.

In a nutshell, VIP eliminates unnecessary waits for TCP timeout. VIPs are also monitored by Clusterware, providing additional high availability for incoming connections. Clusterware uses `ioctl` functions (on Unix and Linux) to check the availability of network interfaces assigned for VIPs. As a secondary method, the gateway server is also pinged to verify network interface availability. Any errors in the ioctl command will immediately relocate the IP address to another available interface or relocate the VIP resource to another node.

It is a common misconception that listeners start after a VIP failover to another node. Listeners are *not* started after a VIP failover to another node in RAC. The network layer level CONN_REST error message is the key to eliminate TCP timeouts.

Subnetting

Subnetting is the process of designating higher-order bits for network prefix and grouping multiple IP addresses (hosts) into subnets. The subnet mask and the network prefix together determine the subnet of an IP address. Counts of possible subnets and hosts vary depending on the number of bits allocated for network prefix.

Consider that first 24 bits of IP address range is assigned to network prefix with an IP prefix of 192.168.3.0. So, the first 24 bits (higher order) identify the subnetwork and the last 8 bits identify the host.

[6]Release 12c allows us to use DHCP-generated IP addresses for VIPs. Discussion in this section applies to DHCP-generated IP addresses also. The only difference between static IP allocation and DHCP-generated IP allocation is that DNS server is queried to retrieve IP address associated with the DHCP aliases, at the time of cluster startup.

In Table 9-1, the subnet mask is set to 255.255.255.0. The network prefix is derived from a BIT AND operation between the IP address and the subnet mask. In the Classless Inter-Domain Routing (CIDR) notation, this subnet is denoted 192.168.3.0/24, as the first 24 bits are used for network prefix. Next, the 8 lower-order bits identify the host. Even though 256 (2^8) IP addresses are possible in the subnet, only 254 IP addresses are allowed, as an IP address with all zeros or all ones in the host part is reserved for network prefix and broadcast address, respectively. So, a 24-bit network prefix scheme provides one subnet and 254 IP addresses in the 192.168.3.0 subnet.

Table 9-1. *Subnet address example*

Component	Binary form	Dot notation
IP address	11000000.10101000.00000010.10010000	192.168.3.144
Subnet mask	11111111.11111111.11111111.00000000	255.255.255.0
Network prefix	11000000.10101000.00000010.00000000	192.168.3.0
Broadcast address	11000000.10101000.00000010.11111111	192.168.3.255

The broadcast address is used to transmit messages to all hosts in a subnet, and it is derived by setting all ones in the host part. In the preceding subnet, the broadcast address is 192.168.3.255.

The IP address range 192.168.3.* can be further divided into multiple subnets. You can divide the IP range 192.168.3.* into two subnets by assigning first the 25 bits of the IP address for the network prefix, as printed in the following example. The subnet mask for these two subnets will be 255.255.255.128; in binary notation, the first 25 bits are set to 1. In CIDR notation, this network is denoted as 192.168.3.0/25.

```
Subnet Mask     : 255.255.255.128 = 11111111.11111111.11111111.10000000
Subnet #1       : 192.168.3.1 to 192.168.3.126
Subnet #2       : 192.168.3.128 to 192.168.3.254
```

Again, a BIT AND operation between IP address and subnet mask will result in the network prefix. Consider an IP address 192.168.3.144 in subnet #2. Then, the network prefix is calculated as 192.168.3.128 as follows:

```
Subnet Mask       : 11111111.11111111.11111111.10000000
IP address        : 11000000.10101000.00000010.10010000 = 192.168.3.144
Network Prefix    : 11000000.10101000.00000010.10000000 = 192.168.3.128
Broadcast         : 11000000.10101000.00000010.11111111 = 192.168.3.255
```

Consider another IP address, 192.168.3.38, in subnet #1 with a network prefix of 192.168.3.0. Notice that the 25th bit will be zero for this block of IP addresses.

```
Subnet Mask       : 11111111.11111111.11111111.10000000
IP address        : 11000000.10101000.00000010.00100110 = 192.168.3.38
Network Prefix    : 11000000.10101000.00000010.00000000 = 192.168.3.0
Broadcast         : 11000000.10101000.00000010.01111111 = 192.168.3.127
```

Similarly, an IP address change can be broken into four subnets by assigning 27 bits for the network prefix.

In an IPv6 scheme, as the number of possible IP addresses is huge, the lower-order /64 bits are used for the host part. In essence, it is a 64-bit addressing scheme.

Cluster Interconnect

Clusterware uses cluster interconnect for exchanging messages such as heartbeat communication, node state change, etc. Cache fusion traffic uses cluster interconnect to transfer messages and data blocks between the instances. Both database and ASM instance communicate via cluster interconnect.

■ **Note** In this chapter, the terms "cluster interconnect" and "private network" are used interchangeably, but "cluster interconnect" refers to the configuration in Oracle Clusterware and "private network" refers to the network configuration in a database server.

Clusterware captures the private network configuration details in OCR (Oracle Cluster Registry) and in an XML file. The command `oifcfg` will print network configuration details. As shown in the following output, subnet 172.18.1.x and 172.18.2.x are configured for cluster_interconnect. IP addresses for private network will be in the range of 172.18.1.1 – 172.18.2.255 in all nodes.

```
$ oifcfg getif
eth1  10.18.1.0  global  public
eth2  10.18.2.0  global  public
eth3  172.18.1.0  global  cluster_interconnect
eth4  172.18.2.0  global  cluster_interconnect
```

In this cluster node, IP address 172.18.1.13 is configured on the eth3 interface and IP address 172.18.2.15 is configured on the eth4 interface.

```
$ ifconfig –a |more
...
eth3: ..<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 4
        inet 172.18.1.13 netmask ffffff00 broadcast 172.18.1.255
...
eth4: ..<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 4
        inet 172.18.2.15 netmask ffffff00 broadcast 172.18.2.255
```

A primary requirement of private network interface configuration is that an IP address configured in the private network interface must be a non-routable interface. A non-routable IP address is visible only in a local network, and the packets destined to those non-routable addresses are guaranteed to be within a switch/router; hence, packets to those destinations will not be routed to the Internet. Communication between two computers within a default home network is a classic example that will help you to understand non-routable IP addresses.[7] The RFC 1918 standard specifies the following IP address ranges are non-routable IP addresses.

```
10.0.0.0 - 10.255.255.255
172.16.0.0 - 172.31.255.255
192.168.0.0 - 192.168.255.255
```

[7]The example of home networks will help to clear up any confusion about private networks. Network Address Translation (NAT) plays a critical role in accessing public IP addresses from private IP address spaces. For example, a home network router creates a private address space. Even though home routers will usually have just one public IP address, many computers are connected to that router, and many computers can access external websites through the router. The router will translate private IP address to a port and modify return address as the public address of the router. Using the router translation table, return packets are delivered to the correct port and hence the correct computer.

While you can possibly use any IP address, including a routable IP address, for a private network, it is a bad practice to use routable IP addresses for private networks. If you use a routable IP address for cluster_interconnect, a cluster may function normally, as ARP ping will populate the MAC address of a local node in the ARP cache, but there is a possibility that packets can be intermittently routed to the Internet instead of another node in the cluster. This incorrect routing can cause intermittent cluster heartbeat failure, even possibly node reboots due to heartbeat failures. Therefore, you should configure only non-routable IP addresses for private networks. Essentially, router or switch should not forward the packets out of the local network. In the case of a RAC cluster, since all cluster nodes are connected to a local network (notwithstanding Stretch RAC), you should keep all private network IP addresses non-routable.

Realizing the danger of routable IP addresses for cluster interconnect configuration, Oracle Database version 11.2 introduced a feature for cluster interconnect known as High Availability IP (HAIP). Essentially, a set of IP addresses, known as link local IP addresses, are configured for cluster interconnect. Link local IP addresses are in the range of 169.254.1.0 through 169.254.254.255. Switch or router does not forward frames to these link local IP addresses outside that network segment. Essentially, even if routable IP addresses are set up for private network on the interface, link local IP addresses protect the frames from being forwarded to the Internet.

■ **Note** If you are configuring multiple interfaces for cluster_interconnect, use different subnets for those private network interfaces. For example, if eth3 and eth4 are configured for cluster_interconnect, then configure 172.29.1.0 subnet in an interface, and 172.29.2.0 subnet in the second interface. A unique subnet per private network interface is a requirement from version 11.2 onward. If this configuration requirement is not followed, and if a cable is removed from the first interface listed in routing table, then ARP does not update the ARP cache properly, leading to instance reboot even though the second interface is up and available. To correct this problem scenario, you must use different subnets if multiple interfaces are configured for cluster_interconnect.

The following ifconfig command output shows that Clusterware configured a link local IP address in each interface configured for cluster_interconnect.

```
$ifconfig –a |more
...
eth3:1: ...<UP,BROADCAST,RUNNING,MULTICAST,IPv4 > mtu 1500 index 4
        inet 169.254.28.111 netmask ffffc000 broadcast 169.254.255.255
...
eth4:1: ...<UP,BROADCAST,RUNNING,MULTICAST,IPv4 > mtu 1500 index 4
        inet 169.254.78.111 netmask ffffc000 broadcast 169.254.255.255
```

As we discussed earlier, the `oifcfg` command shows the network configuration in the Clusterware stored in the OCR. From Oracle RAC version 11g onward, cluster_interconnect details are needed for Clusterware bootstrap process too; therefore, cluster_interconnect details are captured in a local XML file, aptly named profile.xml. Further, this profile.xml file is propagated while adding a new node to a cluster or while a node joins the cluster.

```
# Grid Infrastructure ORACLE_HOME set in the session.
$ cod $ORACLE_HOME/gpnp/profiles/peer/
$ grep 172 profile.xml
...
<gpnp:Network id="net3" IP="172.18.1.0" Adapter="eth3" Use="cluster_interconnect"/>
<gpnp:Network id="net4" IP="172.18.2.0" Adapter="eth4" Use="cluster_interconnect"/>
...
```

■ **Note** As the cluster interconnect details are stored in OCR and profile.xml file, changing network interfaces or IP addresses for private network from 11g needs to be carefully coordinated. You need to add new IP addresses and interfaces while the Clusterware is up and running using the oifcfg command. The oifcfg command updates both OCR and the profile.xml correctly. Then, shut down the Clusterware, make the changes in the network side, and restart the Clusterware. Drop the old interfaces after verifying that Clusterware is working fine and restart Clusterware again. Due to the complexity of the steps involved, you should probably verify the sequence of steps with Oracle Support.

To sum up, a private network should be configured with non-routable IP addresses, and multiple network interfaces in different subnets should possibly be configured so that you can avoid any single point of failure.

Jumbo Frames

Maximum Transmission Unit (MTU) defines the size of largest data unit that can be handled by a protocol layer, a network interface, or a network switch.

1. Default MTU for a network interface is 1,500 bytes. Size of Ethernet frames is 1,518 bytes on the wire.
2. Out of 1,518 bytes in a frame, 14 bytes are reserved for Ethernet header and 4 bytes reserved for Ethernet CRC, leaving 1,500 bytes available for higher-level protocol (TCP, UDP, etc.) payload.
3. For IPv4, a maximum payload of 1,460 bytes for TCP and a maximum payload of 1,476 for UDP are allowed.
4. For IPv6, a maximum payload of 1,440 bytes for TCP and a maximum payload of 1,456 for UDP are allowed. Again, IPv6 is not allowed for private network (including 12c). Jumbo Frame setup applies mostly to private network traffic, and so there is no reason to discuss IPv6 Jumbo Frame setup here.
5. *Path MTU* defines the size of the largest data unit that can flow through a path between a source and a target. In a RAC cluster, typically, a router/switch connects cluster nodes, and the maximum MTU that can be handled by the switch also plays a role in deciding path MTU.

For example, the following ifconfig command output shows an MTU of 1,500 bytes for eth3 logical interface. The size of a packet that flowing through this interface cannot exceed 1,500 bytes.

```
$ifconfig –a |more
...
eth3:1: ...<UP,BROADCAST,RUNNING,MULTICAST,IPv4 > mtu 1500 index 4
        inet 169.254.28.111 netmask ffffc000 broadcast 169.254.63.255
...
```

With a configured MTU of ~1,500 bytes in the path, transfer of a data block of size 8K from one node to another node requires six IP packets to be transmitted. An 8K buffer is fragmented into six IP packets and sent to the receiving side. On the receiving side, these six IP packets are received and reassembled to recreate the 8K buffer. The reassembled buffer is finally passed to the application for further processing.

Figure 9-5 shows how the scheme of fragmentation and reassembly works. In this figure, LMS process sends a block of 8KB to a remote process. A buffer of 8KB is fragmented to six IP packets, and these IP packets are sent to the receiving side through the switch. A kernel thread in the receiving side reassembles these six IP packets and reconstructs the 8KB buffer. The foreground process reads the block from socket buffers to its Program Global Area (PGA) and copies the buffer to database buffer cache.[8]

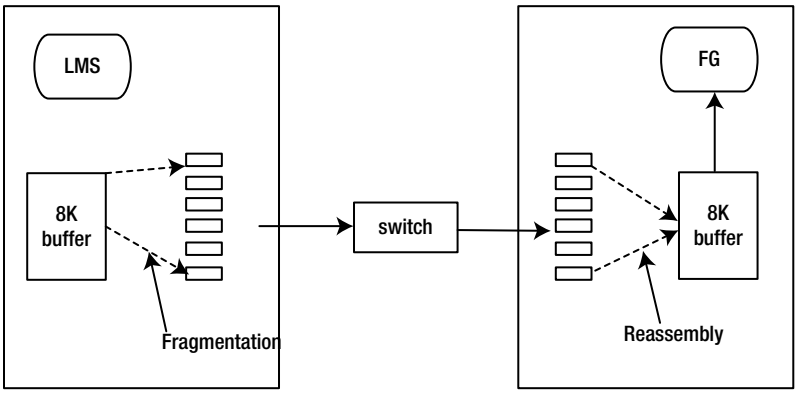


Figure 9-5. *8K transfer without Jumbo Frames*

Fragmentation and reassembly are performed in memory. Excessive fragmentation and reassembly issues can lead to higher CPU usage (accounted to kernel mode CPU usage) in the database nodes.

In a nutshell, Jumbo Frame is an MTU configuration exceeding 1,500 bytes. There is no IEEE standard for MTU of Jumbo Frame configuration, but the industry standard for Jumbo Frame MTU size is 9,000 bytes, with an Ethernet frame on the wire of 9,018 bytes. If the MTU size is set to 9,000, then fragmentation or reassembly is not required. Segments can be transferred from one node to another node without fragmentation or reassembly. As an 8K buffer can fit in a 9,000-byte network buffer, block can be transmitted from one node to another node without fragmentation. However, the path also must support the MTU of 9,000.

Output of the ifconfig command shows that MTU of the eth3 interface is set to 9,000. The maximum size of an IP packet that can flow through this network interface is 9,000 bytes. An MTU setting of 9,000 is known as Jumbo Frames.

```
$ifconfig -a |more
...
eth3:1: ...<UP,BROADCAST,RUNNING,MULTICAST,IPv4 > mtu 9000 index 4
        inet 169.254.28.111 netmask ffffc000 broadcast 169.254.63.255
...
```

MTU configuration must be consistent in all nodes of a cluster and the switch also should support Jumbo Frames. Many network administrators do not want to modify the switch configuration to support Jumbo Frames. Worse, another network administrator might change the setting back to the default of 1,500 in the switch, inducing an unintended Clusterware restart. In essence, setting and maintaining Jumbo Frames is a multidisciplinary function involving system administrators and network administrators.

Still, if your database block size is 16,384 and if path MTU is configured as 9,000, then a 16K block transfer would require a fragmentation and reassembly of two IP packets. This minor fragmentation and reassembly is acceptable, as increasing Jumbo Frames beyond 9,000 is not supported by many switches and kernel modules at this time.

[8]Not all buffers are copied to buffer cache. For example, parallel query copies buffers to its PGA.

Fragmentation and reassembly can happen for PX messaging traffic too. As the PX execution message size determines the size of buffer exchanged between the PX servers, the value of the PX execution message defaults to 16K in Database version 11.2 and 12c. So, RAC clusters with higher-level parallel statement execution would benefit from Jumbo Frames configuration.

Packet Dump Analysis: MTU=1500

Packet dump analysis of cache fusion is useful to understand the intricacies of protocols, MTU, and their importance for cache fusion traffic. The intent of this section is not to delve into network programming but to generate enough understanding so that you can be comfortable with aspects of network engineering as it applies to cache fusion.

In this packet dump analysis, I will use UDP with a block size of 8K transferred between two nodes. I captured these packet dumps in a two-node cluster using the Wireshark tool. The first frame is tagged as Frame 51 with a length of 1,514 bytes. Fourteen bytes are used for the Ethernet header section.

```
Frame 51 (1514 bytes on wire, 1514 bytes captured)

...
```

Source and Destination MAC addresses follow. These MAC addresses are queried from the ARP cache. Ethernet frame details are stored in the header section.

```
Ethernet II, Src: VirtualI_00:ff:04 (00:21:f6:00:ff:04),
             Dst: VirtualI_00:ff:02 (00:21:f6:00:ff:02)
     Destination: VirtualI_00:ff:02 (00:21:f6:00:ff:02)
         Address: VirtualI_00:ff:02 (00:21:f6:00:ff:02)
```

IP header shows the source and target IP address. Note that these IP addresses are link local IP addresses discussed earlier, as IP addresses are in the 169.254.x.x IP range. The length of the IP packet is 1,500 bytes, governed by MTU of 1,500. Note that the identification is set to 21310; all fragmented packets of *a* segment will have same identification number. Using this identification, kernel can reassemble all IP packets belonging to a segment.

```
Internet Protocol, Src: 169.254.90.255 (169.254.90.255), Dst: 169.254.87.19 (169.254.87.19)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 1500
    Identification: 0x533e (21310)
```

Continuing the IP header section, you can see that a few flags indicate fragmentation status. In this case, `more fragments` flag is set, indicating that there are more than one fragment in this stream. Fragment offset indicates the position of this IP packet while assembling the packets.

```
Flags: 0x02 (More Fragments)
    0... = Reserved bit: Not set
    .0.. = Don't fragment: Not set
    ..1. = More fragments: Set
Fragment offset: 0
Time to live: 64
```

In the preceding output, `time to live` is set to 64. If all IP packets needed to reassemble a segment are not received in about 60 seconds, then segment transmission is declared as lost and all packets with the same identification are thrown away.

The following section shows the actual contents of the data itself. Each 1,500-byte packet has a payload of about 1,480 bytes.

```
Data (1480 bytes)

0000  50 97 f5 c1 20 90 2c 9f 04 03 02 01 22 7b 14 00   P... .,.....{..
0010  00 00 00 00 4d 52 4f 4e 00 03 00 00 00 00 00 00   ....MRON........
0020  8c 5e fd 2b 20 02 00 00 94 20 ed 2a 1c 00 00 00   .^.+ .... .*....
```

Five more packets are received and printed by the Wireshark tool. Frame 52 starts at an offset of 1480. So, the first packet starts at an offset of 0 and the second packet has an offset of 1,480 bytes; this offset will be used by kernel to reassemble the buffer.

```
Frame 52 (1514 bytes on wire, 1514 bytes captured)
...
   Identification: 0x533e (21310)
...
    Fragment offset: 1480
    Time to live: 64
    Protocol: UDP (0x11)
```

The last packet in this series of packets is important, as it shows how reassembly is performed. `Frame 56` is the last packet with a size of 970 bytes and a payload of 936 bytes. `More fragments` bit is not set, indicating that there are no more fragments to receive.

```
Frame 56 (970 bytes on wire, 970 bytes captured)
...
    Total Length: 956
    Identification: 0x533e (21310)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
```

Output printed by Wireshark for the last packet shows various IP fragments as a summary. This summary shows that five packets with a payload of 1,480 bytes each and a sixth packet with a payload of 936 bytes were received. These six fragments assemble together to create one UDP buffer of size 8,336 bytes, which is essentially one 8K block plus header information for the cache fusion traffic.

```
[IP Fragments (8336 bytes): #51(1480), #52(1480), #53(1480), #54(1480), #55(1480), #56(936)]
        [Frame: 51, payload: 0-1479 (1480 bytes)]
        [Frame: 52, payload: 1480-2959 (1480 bytes)]
        [Frame: 53, payload: 2960-4439 (1480 bytes)]
        [Frame: 54, payload: 4440-5919 (1480 bytes)]
        [Frame: 55, payload: 5920-7399 (1480 bytes)]
        [Frame: 56, payload: 7400-8335 (936 bytes)]
```

This section shows that this segment uses UDP and specifies source and target ports. In this example, an LMS process uses source port and a foreground process uses destination port. In Linux, the lsof command can be used to identify the port used by a process.

```
User Datagram Protocol, Src Port: 20631 (20631), Dst Port: 62913 (62913)
..
Data (8328 bytes)

0000  04 03 02 01 22 7b 14 00 00 00 00 00 4d 52 4f 4e   ...."{......MRON
```

After receiving all required IP packets (six in this example), kernel thread reassembles the IP packets to a UDP segment and schedules a foreground process listening on the destination port 62913. User process will copy the buffers to application buffers (either PGA or buffer cache).

Packet Dump Analysis: MTU=9000

In this section, we will review the packet dump with a path MTU of 9,000 bytes. As an 8K segment can be transmitted using just one IP packet, there is no need for fragmentation or reassembly.

```
Frame 483 (8370 bytes on wire, 8370 bytes captured)
...
    Frame Number: 483
    Frame Length: 8370 bytes
    Capture Length: 8370 bytes
```

The IP header section shows the source and destination IP addresses of the IP packet. The size of one IP packet is 8,356 bytes. Also, note that `Don't Fragment` flag is set, indicating that this buffer should not be fragmented as path MTU exceeds the IP packet size.

```
Internet Protocol, Src: 169.254.90.255 (169.254.90.255), Dst: 169.254.87.19 (169.254.87.19)
    Version: 4
...
    Total Length: 8356
    Identification: 0x0000 (0)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
```

The UDP section identifies the source and destination ports. In this example, the length of the buffer is 8,336 bytes, with a data payload of 8,328 bytes.

```
User Datagram Protocol, Src Port: 20631 (20631), Dst Port: 62913 (62913)
    Source port: 20631 (20631)
    Destination port: 62913 (62913)
           Length: 8336
```

Essentially, one IP packet of 8,356 bytes is used to transport a payload of 8,328 bytes, avoiding fragmentation and reassembly.

Load Balancing and Failover

Single-component failure should not cause cluster-wide issues in RAC, but network interface failures can cause heartbeat failures leading to node hangs, freeze, or reboots. So, prior to 11gR2, high availability was achieved by implementing an OS-based feature such as IPMP (Sun), Link Aggregation (Sun), Bonding (Linux), etherChannel (AIX), etc. It is outside the scope of this chapter to discuss individual host-based features, but essentially, these features must provide failover and load balancing capabilities.

Linux Bonding is a prevalent feature in Linux clusters. Linux Bonding aggregates multiple network interface cards (NICs) into a bonded logical interface. An IP address will be configured on that bonded interface, and Clusterware will be configured to use that IP address for private network. Clusterware will configure a link local IP address on bonded interface. Depending on the configured mode of the bonding driver, either load balancing or failover is configured, or both are configured among the physical interfaces, as shown in Figure 9-6.

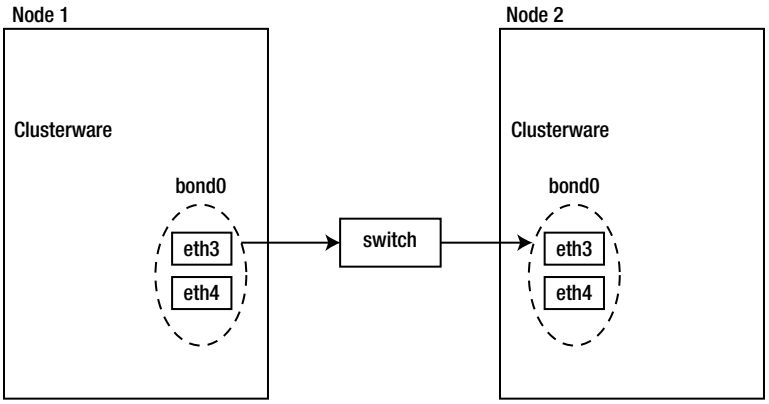


Figure 9-6. *Linux Bonding*

If the OS feature is implemented for load balancing and failover, RAC does not handle load balancing or failover for private network. RAC Database will use the IP address configured for private network, and it is up to the OS-based high availability feature to monitor and manage the network interfaces.

HAIP

HAIP is a high availability feature introduced by Oracle Clusterware in database version 11.2. RAC database and Clusterware are designed to implement high availability in private network with this feature. As HAIP is a feature applicable only to private networks, HAIP supports IPv4 only; IPv6 is not supported.[9]

Consider a configuration with two interfaces configured for private network. The HAIP feature will configure one link local IP address in each interface. RAC database instances and ASM instances will send and receive buffers through those two IP addresses, providing load balancing. Clusterware processes also monitor the network interfaces, and if a failure is detected in an interface, then the link local IP address from the failed interface will be failed over to

[9]Version 12c introduces a new feature, HAVIP, and you should not confuse HAVIP with the HAIP feature. The HAVIP feature is applicable to public network, and the HAIP feature is applicable to private network.

an available interface. Output of ifconfig shows that logical interface eth3:1 is configured with link local -IP address 169.254.28.111 and that eth4:1 is configured with link local IP address 169.254.78.12. Both Database and Clusterware will use both IP addresses for private network communication.

```
$ ifconfig –a |more
...
eth3: ..<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 9000 index 4
        inet 172.18.1.13 netmask ffffff00 broadcast 172.18.1.255
eth3:1: ..<UP,BROADCAST,RUNNING,MULTICAST,IPv4 > mtu 9000 index 4
        inet 169.254.28.111 netmask ffffc000 broadcast 169.254.63.255
...
Eth4: ..<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 9000 index 4
        inet 172.18.2.15 netmask ffffff00 broadcast 172.18.2.255
eth4:1: ..<UP,BROADCAST,RUNNING,MULTICAST,IPv4 > mtu 9000 index 4
        inet 169.254.78.12 netmask ffffc000 broadcast 169.254.127.255
...
```

You can identify the IP addresses and ports used by a process for cache fusion traffic using the `oradebug ipc` command. In the following example, I used the oradebug command to dump IPC context information to a trace file after connecting to the database.

```
RS@RAC1>oradebug setmypid
Oracle pid: 6619, Unix process pid: 28096, image: oracle@node1

RS@RAC1>oradebug ipc
Information written to trace file.

RS@RAC1>oradebug tracefile_name
/opt/app//product/12.1.0/admin/diag/rdbms/RAC/RAC1/trace/RAC1_ora_28096.trc
```

From the trace file, you can identify that foreground process listens in four IP addresses. Foreground process uses all four UDP ports to communicate between processes in other nodes. Essentially, since all IP addresses are in use, this scheme provides load balancing among the IP addresses.

```
SKGXP: SKGXPGPID Internet address 168.254.28.111 UDP port number 45913
SKGXP: SKGXPGPID Internet address 168.254.78.12 UDP port number 45914
SKGXP: SKGXPGPID Internet address 168.254.36.6 UDP port number 45915
SKGXP: SKGXPGPID Internet address 168.254.92.33 UDP port number 45916
```

If there is an interface assigned for cluster interconnect, then the HAIP feature will configure one link local IP address in that interface. If there are two interfaces assigned for private network, then two link local IP addresses will be configured per node. If more than two interfaces are assigned, then four link local IP addresses will be configured in those interfaces per node. For example, if there are four interfaces configured in the database server for private network, then Clusterware will configure one link local IP address each in the configured interfaces.

Also, Clusterware uses the network configuration from the first node you execute, root.sh, as the reference configuration. So, it is important to keep network configuration consistent in all nodes of a cluster.

Highly Available VIP (HAVIP)

Version 12c introduces the new feature HAVIP, a Clusterware-monitored VIP that can be used for non-Database applications. Typically, VIPs are created for VIP or SCAN listeners for Database connections. In some cases, the need arises to create a VIP monitored by Clusterware designed to support other applications.

For example, following command creates a new VIP resource in the Clusterware. Starting this VIP resource will plumb a new IP address in a node. The following command adds a VIP resource with an IP address of 10.5.12.110 in the Clusterware. Enabling the resource plumbs IP address 10.5.12.110 in `rac2.example.com` node. Note that this IP address is added to the second network, indicating that HAVIP can be added in any network.

```
# srvctl add havip -id customapp -address 10.5.12.110 -netnum 2
# srvctl enable havip -id customapp -node rac2.example.com
```

This HAVIP resource was primarily designed to support highly available NFS export file systems. The following command adds an NFS-exported file system using the HAVIP resource created earlier.

```
# srvctl add exportfs -name customfs1 -id customapp -path /opt/app/export
```

You could also create this file system automatically exported to clients, as shown in the following command. In this case, NFS is exported to hr1 and hr2 servers.[10]

```
# srvctl add exportfs -name customfs1 -id customapp -path /opt/app/export -clients hr1,hr2
```

The HAVIP feature is a new feature to extend Clusterware functionality, to implement and maintain highly available non-database VIP and export file system.

Kernel Parameters

Kernel parameters affecting network configuration are a critical area to review for Oracle RAC. Misconfiguration or undersized parameters can cause noticeable performance issues, even errors.

In the Linux platform, `rmem_max` and `rmem_default` control the size of receive socket buffers, and `wmem_max` and `wmem_default` control the size of send socket buffers. If the interconnect usage is higher and if you have plenty of memory available in the server, then set rmem_default and wmem_default to 4M, and set rmem_max and wmem_max parameters to 8M. However, these four parameters control the size of socket buffers for all protocols. In the Solaris platform, udp_xmit_hiwat and udp_recv_hiwat parameters control the size of socket buffers for UDP, and the documentation recommends setting these two parameters to at least 64K. Our recommendation is to increase these parameters to about 1M.

Memory allocated for fragmentation and reassembly should be configured optimally too. If the buffer size is too small, then the fragmentation and reassembly can lead to failures that lead in turn to dropped packets. In Linux, parameters `net.ipv4.ipfrag_low_thresh` is the lower bound and `net.ipv4.ipfrag_high_thresh` is the upper bound for memory allocated for fragmentation and reassembly.

The parameter `net.ipv4.ipfrag_time` determines the duration to keep IP packets in the memory waiting for reassembly, and the value of this parameter defaults to 60 seconds. If the IP packets are not assembled within 60 seconds, for example, a required IP packet is not received, then all IP packets of that UDP segment are dropped. However, this timeout is probably not effective in Oracle RAC, since database-level timeout is configured to be smaller than 60 seconds. We will explore database-specific timeout while discussing go lost packets wait event.

[10]Note that this is not a complete set of steps to create NFS exported file system.

Network Measurement Tools

Debugging RAC network performance issues involves a review of both database-level statistics and network-level statistics, if database statistics indicate possible network performance issues. It is not possible to discuss every flag and option in network tools, and so this section discusses important flags for Solaris and Linux platforms. You can identify the flags in your platform of choice by reviewing man pages or googling for options in the tool.

Ping is useful to validate connectivity to a remote node, `traceroute` is useful to understand the route taken by a network packet between the nodes, and `netstat` is useful to understand various network-level statistics.

Ping Utility

The ping command sends a packet to a remote IP address, waits for a response from the target IP address, computes the time difference after receiving the response for a packet, and prints the output. By default, the ping utility uses the ICMP protocol, a lightweight protocol designed for utilities such as ping.

If there are multiple interfaces in the server, you can specify a specific interface using the –i flag in Solaris. In the following example, eth3 interface will be used to send a packet to a remote host.

```
$ /usr/sbin/ping –s -i eth3 169.254.10.68
PING 169.254.10.68: 56 data bytes
64 bytes from 169.254.10.68: icmp_seq=0. time=0.814 ms
64 bytes from 169.254.10.68: icmp_seq=1. time=0.611 ms
...
```

As RAC databases use UDP predominantly, it is beneficial to specify UDP for a ping test instead of the default ICMP protocol. In the following example, -U flag is specified to test the reachability using UDP through the eth3 interface. UDP requires a port number, and the ping utility sends packets to ports starting with 33434. Also, -i flag specifies eth3 interface for this test.

```
$ /usr/sbin/ping –s -U -i eth3 169.254.10.68
PING 169.254.28.111: 56 data bytes
92 bytes from 169.254.10.68: udp_port=33434. Time=0.463 ms
92 bytes from 169.254.10.68: udp_port=33435. time=0.589 ms
92 bytes from 169.254.10.68: udp_port=33436. time=0.607 ms
...
```

By default, ping sends a small packet of 56 bytes. Typically, a RAC packet size is much bigger than 56 bytes. Especially, Jumbo Frame setup requires a bigger packet to be sent to the remote node. In the following example, a packet of 9,000 bytes is sent by the ping utility, and five packets were transmitted. An acknowledgement of 92 bytes was received in 0.6 ms. The round trip of these packets took approximately 0.6 ms; that is, latency for a transmission of 9,000 bytes and receipt of 56 bytes took approximately 0.6 ms.

```
$ /usr/sbin/ping -s -U -i eth3 169.254.10.68 9000 5
PING 169.254.10.68: 9000 data bytes
92 bytes from 169.254.10.68: udp_port=33434. time=0.641 ms
92 bytes from 169.254.10.68: udp_port=33435. time=0.800 ms
92 bytes from 169.254.10.68: udp_port=33436. time=0.832 ms
...
```

The preceding examples are from a Solaris OS. Syntax may be slightly different in other platforms. In the Linux platform, the following ping command can send a Jumbo Frame using the TCP/IP protocol.

```
$ ping –s 8500 –M do 169.254.10.68 –c 5
```

The Linux platform supports many tools such as lperf, echoping, tcpdump, etc. echoping tool is useful to verify Jumbo Frames configuration using UDP protocol.

```
$ echoping -u -s 8500 -n 5 169.254.10.68:12000
```

The lperf tool is a better tool to measure network throughput using the UDP protocol. However, lperf uses a client and server mode execution to send and receive packets.

On the server side:

```
$ lperf -u -s 169.254.10.68 -p 12000
```

On the client side:

```
$ lperf -u -c  169.254.10.68 -p 12000
```

Traceroute

The route taken by a packet can be identified using the traceroute utility. Cluster interconnect packets should not traverse multiple hops, and the route should have just one hop (Extended RAC excluded). The traceroute utility can be used to identify routing issues if any.

The following is an example to trace the route from a local IP address to a remote IP address, both link local IP addresses. As they are link local IP addresses, there should be just one hop. In the following example, the source IP address is specified with –s flag `169.254.28.111` and the target IP address is 169.254.10.68. The size of the packet is specified as 9,000 bytes. A 9,000-byte packet is sent and a response received after reaching the target. In this example, the packet traversed just one hop, as this is a link local IP address.

```
$ /usr/sbin/traceroute   -s 169.254.28.111  169.254.10.68 9000
traceroute to 169.254.10.68 (169.254.10.68) from 169.254.28.111, 30 hops max, 9000 byte packets
169.254.10.68 (169.254.10.68)  1.120 ms  0.595 ms  0.793 ms
```

A private network should not have more than one hop; if there are more hops, you should work with network administrators to correct the issue. Traceroute can also be used to identify if the packet route between an application server and database server is optimal or not.

Netstat

Netstat is an important utility to troubleshoot network performance issues. Netstat provides statistics of various layers of network such as TCP/UDP protocol, IP protocol, Ethernet layer, etc. You can identify the layer causing the problem using the netstat utility too. Since output and options between the operating systems are different, I have divided this section into two, discussing Solaris and Linux platforms.

Netstat: Solaris

The following command queries the network statistics for UDP and requests five samples of 1 second each. In the following output, the first line prints the cumulative statistics from the restart of a server; about ~100 billion UDP datagrams (or segments) received and ~98 billion UDP datagrams are sent by the local node. However, the rate is much more important, and approximately 40K UDP segments are sent and received per second. Note that the size of UDP segment is a variable; it is not possible to convert the number of segments to bytes in the UDP layer. Also note that there are no errors reported for UDP layer processing.

```
$ netstat –s –P udp 1 5  |more
...

UDP     udpInDatagrams      =109231410016       udpInErrors         =     0
        udpOutDatagrams     =98827301421        udpOutErrors        =     0

UDP     udpInDatagrams      = 40820     udpInErrors         =     0
        udpOutDatagrams     = 39340     udpOutErrors        =     0
```

The following output shows the statistics in IP layer processing specifying `–P ip` flag. Version 11gR2 does not support IPv6, but version 12c supports IPv6 for public network. If you have configured to use IPv4, then use the IPv4 section of netstat; if you are reviewing metrics for IPv6-based public network, then use the IPv6 section of network statistics. The following statistics show that about 63K IP packets were received and 64K IP packets were sent for IPv4. There is no activity in IPv6, as all networks in this database server use the IPv4 protocol.

```
$ netstat -s -P ip 1  5
...
IPv4    ipForwarding        =     2     ipDefaultTTL        =   255
...
        ipInAddrErrors      =     0     ipInCksumErrs       =     0
..
        ipInDelivers        = 63639     ipOutRequests       = 64670

$ netstat -s -P ip6 1  5
IPv6    ipv6Forwarding      =     2     ipv6DefaultHopLimit =   255
        ipv6InReceives      =     0     ipv6InHdrErrors     =     0
```

Continuing the output for IP protocol, the following output shows that about 4,068 packets needed to be reassembled, and all packets were reassembled successfully (Receive metrics). If there are reassembly errors, you would see that the ipReasmFails counter is incremented frequently. A few failures may be acceptable, as TCP/IP processing also uses an IP layer underneath.

```
        ipOutDiscards       =     0     ipOutNoRoutes       =     0
        ipReasmTimeout      =     0     ipReasmReqds        =  4068
        ipReasmOKs          =  4068     ipReasmFails        =     0
...
```

The following output shows IP packet fragmentation (Send metrics). About ~4,000 segments were fragmented, creating ~20,000 IP packets. You can guess that DB block size is 8K and MTU is set to 1,500, as there is a 1:6 ratio between IP fragment tasks and the number of IP fragments.

```
        ipFragOKs           =  3933     ipFragFails         =     0
        ipFragCreates       = 19885     ipRoutingDiscards   =     0
```

Interface layer statistics can be queried using –I flag in netstat (Solaris). In the following output, the first five columns show the frame level statistics for the eth3 interface. The next five columns show cumulative statistics for all interfaces in the node. Approximately 13,000 packets were received by the eth3 interface, and 13,000 packets were sent through the eth3 interface.

```
$ netstat -i -I eth3  1  5 |more

    input   eth3     output       input  (Total)    output
packets errs  packets errs  colls  packets errs  packets errs  colls
2560871245 0     4173133682 0     0      20203010488 0     20470057929 0     0
13299   0     13146   0     0      79376   0     96396   0     0
11547   0     12497   0     0      72490   0     90787   0     0
12416   0     13570   0     0      73921   0     92297   0     0
12011   0     12154   0     0      71470   0     85764   0     0
```

Netstat: Linux

The output of netstat utility in Linux is different from that on Solaris. Also, netstat command options are slightly different from the Solaris platform.

The following output shows that about ~19,000 UDP datagrams were received and ~20,000 UDP datagrams were sent. Command requests a 1-second sample.

```
$ netstat –s 1
...
Udp:
    19835 packets received
    8 packets to unknown port received.
    0 packet receive errors
    20885 packets sent
...
```

Similarly, IP layer output shows details about processing in the IP layer. For example, output shows that 4068 reassembly occurred with no failures.

```
Ip:
ipOutDiscards       =     0     ipOutNoRoutes       =     0
ipReasmTimeout      =     0     ipReasmReqds        =  4068
ipReasmOKs          =  4068     ipReasmFails        =     0
```

Interface-level statistics can be retrieved using –i flag. As you can see, interfaces work fine with no Errors or Drops in the interfaces.

```
$ netstat -i  1|more

Kernel Interface table
Iface       MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500   0     2991      0      0      0      770      0      0      0 BMRU
eth1       1500   0    27519      0      0      0    54553      0      0      0 BMRU
```

Example Problem

The following shows the output of the netstat command when there were reassembly errors in the server. In the following output of netstat command, ~1.4 billion packets were received and 1.3 billion packets were sent. Also, ~100 million packets needed to be reassembled, but 1.6 million packets failed for reassembly. This output is from OSWatcher file, and given here is just the first sample. As there were numerous reassembly failures, this server would require further review to understand the root cause.

```
Ip:
    1478633885 total packets received     → Total packets received so far.
    28039 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    1385377694 incoming packets delivered → Total packets sent so far
    1045105164 requests sent out
    6 outgoing packets dropped
    57 dropped because of missing route
    3455 fragments dropped after timeout
    106583778 reassemblies required → Reassembly required
    32193658 packets reassembled ok
    1666996 packet reassembles failed → Failed reassembly
    33504302 fragments received ok

    111652305 fragments createdThere are many reasons for reassembly failures. More analysis is required using
```

tools such as tcpdump or Wireshark to understand the root cause. In most cases, troubleshooting reassembly failure is a collaborative effort among DBAs, network administrators, and system administrators.

GC Lost Block Issue

If there is no response to a request within 0.5 seconds, then the block is declared lost and the lost block–related statistics are incremented. Reasons for lost block error condition include CPU scheduling issues, incorrect network buffer configuration, resource shortage in the server, network configuration issues, etc. It is a very common mistake to declare a gc lost block issue as a network issue: you must review statistics from both database and OS perspectives to understand the root cause.

Oracle RAC uses an algorithm shown in Figure 9-7 to detect lost blocks. First, a foreground process sends a request to the LMS process running in remote node. Until a response is received from the LMS process, foreground process cannot continue. So, foreground process sets up a timer with an expiry time of 0.5 seconds (in the Windows platform, this timer expiry is set to 6 seconds) and goes to sleep. If a block/grant to read a block is received by the foreground process, then the foreground process will continue processing. If no response is received within a 0.5-second time interval, then the alarm will wake up the foreground process, which will declare the block as lost, and account wait time to `gc cr block lost` or `gc current block lost`. The foreground process will resubmit the request to access for the same block to LMS process. The foreground process can get stuck waiting for a block in this loop if there is a problem sending or receiving a block.

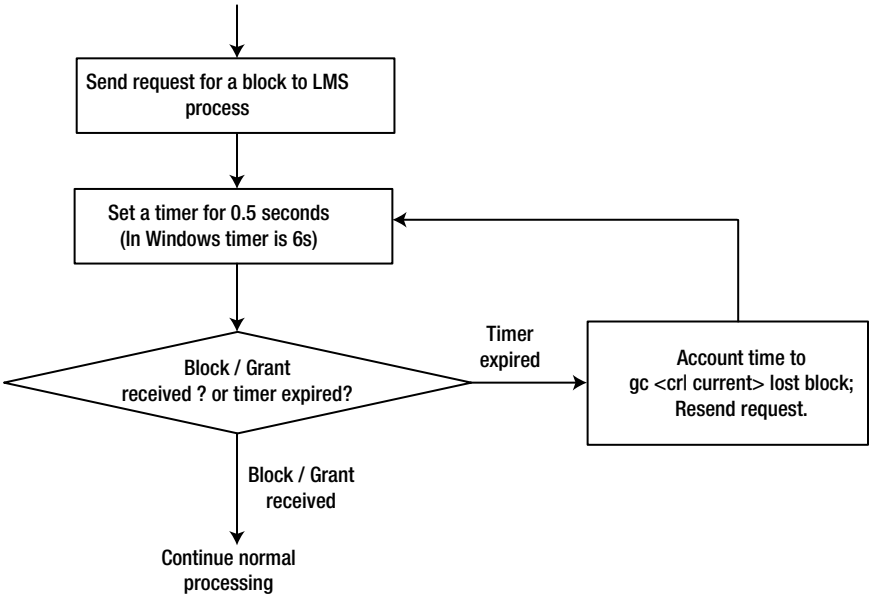


Figure 9-7. *gc lost block logic*

The SQL Trace file of a process stuck waiting for a block is as follows. The foreground process is waiting for a block with file#=33, block#=1701. You can see the approximate wait time of 0.5 seconds in the `ela=` field for these waits. After 0.5 seconds of sleep, the process re-requests for the block, but the block is not received in that interval, going into a continuous sleep-resend cycle. (In this problem scenario, due to a switch issue, few blocks were blocked by the switch never arriving to the requestor.)

```
*** 2012-04-05 10:51:47.280
WAIT #47390186351424: nam='gc cr block lost' ela= 520692 p1=33 p2=1701 p3=1 obj#=409860
*** 2012-04-05 10:51:48.318
WAIT #47390186351424: nam='gc cr block lost' ela= 516782 p1=33 p2=1701 p3=1 obj#=409860
WAIT #47390186351424: nam='gc cr block lost' ela= 523749 p1=33 p2=1701 p3=1 obj#=409860
...
```

Parameter `_side_channel_batch_timeout_ms` controls the duration of sleep in a database instance and defaults to 0.5 seconds (prior to 10gR2, this parameter default value was 6 seconds) in the Unix platform. On Windows, another parameter, `_side_channel_batch_timeout`, is used with a default value of 6 seconds. There is no reason to modify any of these parameters.

It is always important to quantify the effect of a wait event; in a perfectly healthy database, you can expect a few waits for lost block–related wait events, and you should ignore those events if the DB time spent waiting for lost blocks is negligible. Also, it is possible for another event such as gc current block receive time or gc cr block receive time to be elevated due to the loss of blocks.

Figure 9-8 shows an AWR report for the duration of a few minutes; 62% of DB time was spent in waiting for `gc cr block lost` event. As the percentage of DB time spent on this wait event is very high, this database must be reviewed further to identify the root cause.

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
gc cr block lost	557	293	526	62.54	Cluster
DB CPU		110		23.52	
db file sequential read	3,806	26	7	5.44	User I/O
log file sync	1,154	12	10	2.52	Commit
DFS lock handle	5,400	7	1	1.44	Other

Figure 9-8. *Top five timed foreground events*

Blocks can be lost due to numerous reasons such as network drops, latencies in CPU scheduling, incorrect network configuration, and insufficient network bandwidth. When a process is scheduled to run on a CPU, that process will drain socket buffers to application buffers. If there are delays in CPU scheduling, then the process might not be able to read socket buffers quick enough. If the incoming rate of packets to the socket buffers for that process is higher, then the kernel threads might encounter "buffer full" condition, and kernel threads will drop incoming packets silently. So, all necessary IP packets may not arrive, leading to UDP segment reassembly failures.

If there are lost block issues and if the DB time spent on lost block issue is not negligible, then you should identify if there was any CPU starvation in the server during the problem duration. I have seen severe latch contention causing higher CPU usage leading to lost block issues. Also, kernel threads have higher CPU priority than the User Foreground processes, so, even if the foreground processes are suffering from CPU starvation, kernel threads can receive the network frames from the physical link layer and fail to copy the frames to socket buffers due to "buffer full" conditions.

Network kernel parameters must be configured properly if the workload induces a higher rate of cluster interconnect traffic. For example, rmem_max and wmem_max kernel parameters (Linux) might need to be increased to reduce the probability of buffer full conditions.

Memory starvation is another issue that can trigger a slew of lost bock waits. Memory starvation can lead to swapping or paging in the server affecting the performance of foreground processes. So, foreground processes will be inefficient in draining socket buffers and might not be able to drain network buffers quick enough, leading to "buffer full" conditions. On the Linux platform, HugePages is an important feature to implement; without HugePages configured, kernel threads can consume all available CPUs during memory starvation, thus causing the foreground process to suffer from CPU starvation and lost blocks issues.

In summary, identify if the DB time spent for lost block waits is significant. Check if the resource consumption in the database server(s) is good, verify that kernel parameters related to network are properly configured, and verify if the routes between the nodes are optimal. If resolving these problems does not resolve the root cause, then review link/switch layer statistics to verify if there are any errors.

Configuring Network for Oracle RAC and Clusterware

As we discussed at the beginning of this chapter, Oracle RAC and Oracle Grid Infrastructure require three types of network configurations: public network, private network, and storage network. The public network is primarily for the database clients to connect to the Oracle database. Also, the public network may connect the database servers with the company corporate network, through which applications and other database clients can connect to the database. The private network is the interconnect network just among the nodes on the cluster. It carries the cluster interconnect heartbeats for the Oracle Clusterware and the communication between the RAC nodes for Oracle RAC cache fusion.

The storage network connects the RAC nodes with the external shared storage. This chapter focuses on public network as well as private network. Chapter 2 has more details about the configuration of the storage network. The configuration of the public and private networks consists of the following tasks:

1. Configuring network hardware
2. Configuring network in OS
3. Establishing the IP address and name resolution
4. Specifying network configuration in Grid Infrastructure installation

The next few sections discuss the configuration of these two types of network.

Network Hardware Configuration

The network hardware for Oracle RAC and Oracle Clusterware includes the NICs or network adapters inside the hosts and the network switches between the hosts (nodes) or between the host and storage. For example, each host can have multiple NICs, and some NICs can have multiple ports. Each port is presented as a network interface in OS. These network interfaces are numbered as eth0, eth1, and so on in Linux. As a minimal configuration, two network interfaces are required; one for public network and one for the private network (one NIC portis needed for the storage if you use network based storage). You should use the network switch for the network connection and should not use the crossover cable for the private interconnect to connect between the RAC nodes. For high availability, it is highly recommended that you use the following redundant configuration:

- One or two network interfaces for the public network with one or two redundant public switches to connect to the public network
- Two network interfaces for the private network with two dedicated physical switches that are connected only to the nodes on the cluster. These two network interfaces should not be on the same NIC if you use a dual-port NIC.
- Two or more network interfaces or HBAs for the storage network with two or more dedicated switches, depending on the storage network protocol.

With Oracle HAIP, you can configure up to four network interfaces for the private network. Exactly how many network interfaces you can use for each network is also limited by the total number of the NIC ports you have on your server. However, you should use the same network interface for the same network in all the nodes in the same cluster. For example, you can use eth0 and eth1 for the public network and eth3 and eth4 for the private network on all the nodes.

The network adapter for the public network must support TCP/IP protocol, and the network adapter for the private network must support the UDP. The network switch for the private network should support TCP/IP, and it should be at least 1-gigabit Ethernet or higher to meet the high-speed interconnect requirement. Recently, higher-speed networks such as 10-gigabit Ethernet or InfiniBand switches have been widely adapted for private network. To use a higher-speed network for private network, you need to make sure that all the components on the network path such as NICs, network cables, and switches, operate at higher speed.

It is important that these networks should be independent and not physically connected. It is also recommended that two dedicated switches be used for the private interconnect. Figure 9-9 shows an example of such a fully redundant private network between two nodes of the clusters.

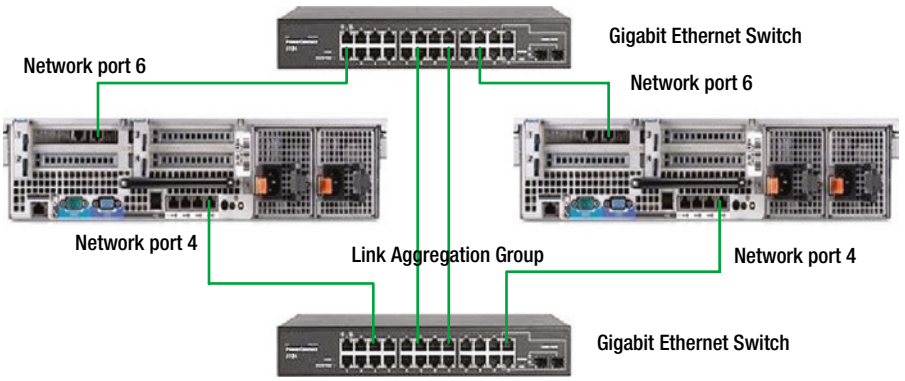


Figure 9-9. *A fully redundant private network configuration*

In this example, both nodes dedicate two network ports, port 4 and port 6, to the private network. Each of these two network ports connects two different-gigabit Ethernet switches dedicated to the private network. These two switches are also interconnected with two redundant network cables, as shown in Figure 9-9. Although this example shows the private network configuration of a two-node cluster, a cluster with three or more nodes is connected in a similar way. This configuration ensures there is no single point of failure on the private network.

Configuring Network in OS

Once you decide the network interfaces for the public network and private network, you can start configuring these two networks based on the network interfaces. In this chapter, we mainly focus on how to configure network in Linux.

The public network includes the network interface and the IP address of the host and virtual hostname/IP for each node and the SCAN name/IP for the cluster. Configuring the public network requires you to do the following:

- Give the hostname and its IP address of each RAC node. For example, you have two hosts and their IP addresses: k2r720n1(172.16.9.71) and k2r720n2 (172.16.9.72). Be aware that the hostname should be composed of alphanumeric characters, and the underscore ("_") is not allowed in the hostname.
- Configure the network interface of the public network on each RAC node. For example, on Linux, edit /etc/sysconfig/network-scripts/ifcfg-eth0:

```
DEVICE=eth0
BOOTPROTO=static
HWADDR=00:22:19:D1:DA:5C
ONBOOT=yes
IPADDR=172.16.9.71
NETMASK=255.255.0.0
TYPE=Ethernet
GATEWAY=172.16.0.1
```

In the preceding output, 172.16.9.71 is the IP address of the public network for the node.

- Assign the virtual hostname and VIP for each RAC node. It is recommended that you use the format <public hostname>-vip for the virtual hostname, for example, racnode1-vip. An example we have set for the following virtual names and VIPs for test cluster: k2r720n1-vip with IP address 172.16.9.171 and k2r720n2-vip with IP address 172.16.9.172
- Give the SCAN (Single Client Access Name) and three SCAN VIPs for the Cluster.For example, we have set the SCAN name kr720n-scan with three scan IP: 172.16.9.74, 172.16.9.75, 172.16.9.76.

- To achieve the high availability and load balancing, multiple redundant network interfaces for the private network should be bonded or teamed together with OS bonding or teaming methods prior to 11gR2 (11.2.0.2). With the introduction of Oracle HAIP in 11.2.0.2, bonding or teaming in OS is no longer needed, as HAIP allows you to use these redundant network interfaces directly for the private interconnect, and HAIP handles the load balance across these network interfaces and the failover to other available network interfaces in case any of these network interfaces is not responding. The following shows the example of the two network interfaces, eth1 and eth2:

```
DEVICE=eth1
HWADDR=BC:30:5B:ED:68:C1
ONBOOT=yes
BOOTPROTO=static
TYPE=Ethernet
IPADDR=192.168.9.71
NETMASK=255.255.255.0

DEVICE=eth2
HWADDR=BC:30:5B:ED:68:C2
ONBOOT=yes
BOOTPROTO=static
TYPE=Ethernet
IPADDR=192.168.9.72
NETMASK=255.255.255.0
```

After you complete the configuration and save the files, you can restart the network service:

```
#Service network restart
```

Then you can use these two network interfaces, eth12 and eth2, directly for the private network configuration during the Oracle Grid Infrastructure installation.

Establishing IP Address and Name Resolution

As you connect to a host using the hostname, Oracle database clients connect to the database using a SCAN name. These hostnames and SCAN names need to be resolved into their associated IP addresses to allow the clients to connect. In Oracle RAC and Clusterware, the SCAN name can be resolved into up to three SCAN IPs. For example, the SCAN name kr720n-scan are resolved into three IPs, 172.16.9.74, 172.16.9.75, and 172.16.9.76, in the following example. The virtual hostname is resolved into the VIP; for example, k2r720n1-vip being resolved to VIP 172.16.9.171. The public hostname is resolved into the host IP; for example, k2r720n1 being resolved into 172.16.9.71.

The technology to resolve a name to the IP address is called name resolution. In the Oracle RAC environment, there are two ways to implement this name resolution: 1) name resolution using DNS, and 2) name resolution using GNS (Grid Naming Services). Oracle recommends that you establish a static hostname resolution using DNS for all non-VIP node public hostnames. You can use either DNS or GNS for the VIP name resolution. In the following sections, we will use two clusters as an example to show the configuration of each of these two methods and how it works: 1) kr720n-cluster for the DNS method, and 2) owirac-cluster for the GNS method.

Name Resolution Using DNS

In the DNS method, you need to register the SCAN name of the cluster and the virtual hostnames and public hostname of each cluster node along with the corresponding IP address. This registration is performed on the DNS server as well as on the DNS clients which are all the cluster nodes.

On the DNS server, we need to add the name resolutions for the SCAN name, virtual hostname(VIP), and public hostname on DNS. The following example shows the added entries in the DNS for the "kr720n-cluster" cluster:

```
kr720n-scan.dblab.com    IN   A  172.16.9.74
                         IN   A  172.16.9.75
                         IN   A  172.16.9.76
k2r720n1.dblab.com       IN   A  172.16.9.71
k2r720n1-vip.dblab.com   IN   A  172.16.9.171
k2r720n2.dblab.com       IN   A  172.16.9.72
k2r720n2-vip.dblab.com   IN   A  172.16.9.172
```

On each DNS client that is a node in the cluster, we need to configure the /etc/resolve.conf file by adding the entries that resolve the DNS server.

```
#cat /etc/resolv.conf
search dns.dblab.com
nameserver 172.16.150.55
```

Here, dns.dblab.com is the DNS server hostname, and 172.16.150.55 is the DNS server IP address on the network.

On each RAC node, you also need to modify /etc/nsswitch.conf file to change the entry

```
        hosts:   files  nis  dns
To
        Hosts:   dns files  nis
```

After saving the change on /etc/nsswitch.conf, you need to restart the nscd daemon on each node with the following command:nscd daemon on each node with the following command:

```
#/sbin service restart
```

This method places the DNS entry first and the NIS entry at the end of the search.

Oracle also recommends that you add the public hostnames and IP addresses of all the nodes on /etc/hosts file on the each node. For example, add these two entries in /etc/hosts on each node.

```
#public
172.16.9.71  k2r720n1.dblab.com k2r720n1
172.16.9.72  k2r720n2.dblab.com k2r720n2
```

It is also recommended that you do not add the SCAN name and IPs in the /etc/hosts file. Let DNS resolve SCAN VIPs.

After this configuration is completed, you can ping the SCAN name and each host's virtual hostname. The DNS resolves the names to IPs; however, these IPs are not started until you finish the Grid Infrastructure installation using these names and VIPs and start the Grid Infrastructure. All the VIP addresses, including SCAN VIP addresses, should not be used for other purposes and should not be ping-able before the Grid Infrastructure installation completes.

As remarked previously, in this DNS method you need to add three entries for the SCAN; one entry each for the virtual hostname for every node in the cluster; and one entry each for the public hostname for every node on the cluster. This manual configuration is not too cumbersome for a small environment; however, in a production environment with

many RAC clusters, each cluster with many nodes, numerous DNS entries must be configured.
For example, a six-node cluster requires 2 x 6 + 3 = 15 entries, and ten such clusters require 150 entries. As the number of cluster databases and the total number of nodes increase, the DNS methods became less manageable. For this reason, Oracle 11gR2 Clusterware introduced the second method called GNS.

Name Resolution Using GNS

GNS provides name resolution for VIPs based on DHCP (Dynamic Host Configuration Protocol). In this method, GNS provides dynamic IP addresses for the virtual hostnames of all the nodes and SCAN in the cluster, through the DHCP service. In GNS, the only GNS VIP is given a static IP, and this static GNS VIP is registered in DNS. The GNS method use the subdomain delegation, DNS forwards any network name request on a particular subdomain to the GNS server, and the GNS will provide the name resolution to the dynamic IP. GNS uses multicast Domain Name Server (mDNS) in the Oracle Clusterware to provide a DHCP IP address mapping to the VIP name. The benefit of GNS is the simplification of SCAN VIPs and node VIPs by moving the management of these VIPs from DNS to the GNS local to the cluster. With GNS, adding or deleting a node is even simpler than with the DNS method, as there is no change in DNS when you add or delete a node.

Here, we use the knewrac cluster as an example to illustrate how GNS name resolution works (Figure 9-10) and the steps to configure it.

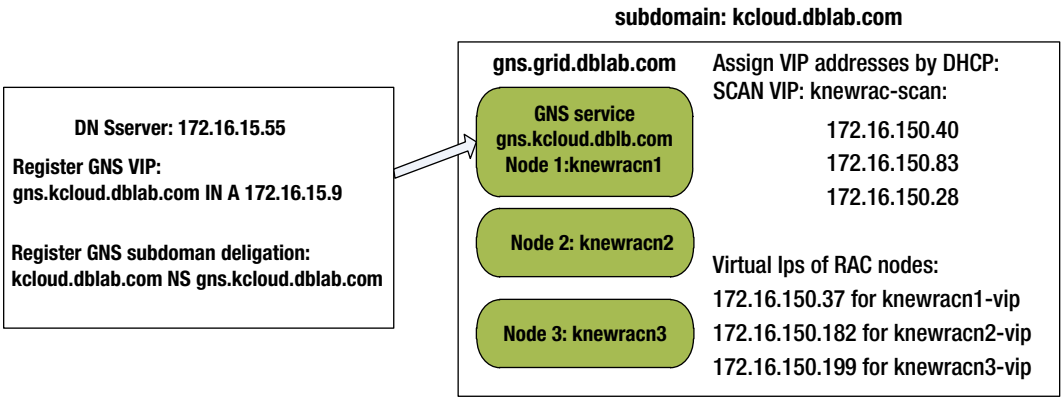


Figure 9-10. *Name resolution using GNS*

This cluster uses the subdomain `kcloud.dblab.com`. In DNS, two entries are requested:

- `the static IP address of GNS VIP: 172.16.15.9`
- `the subdomain deligagtion:`

  `subdomain kcloud.dblab.com to GNS server gns.kcloud.dblab.com`

With the proceeding setup, any host request in the domain "`*.kblod.dblab.com`", for example, `knewrac-scan.kclod.dblab.com`, will be handed off to the GNS server: `gns.kcloud.dblab.com` with VIP address 172.16.15.9. This GNS server maps the VIP name `knewrac-scan.kclod.dblab.com` to one of the three SCAN VIPs: 172.16.150.40, 172.16.150.83, or 172.16.150.28.

The steps for implementing GNS name resolution using this example are as follows:

Step 1: On the DNS server, add a zone dblab.com to /etc/named.conf file. This zone handles DNS queries for the domain's subnet, "dblab.com":

```
zone "dblab.com" {
type master;
file "dblab.com.zone";
};
```

Then, add these entries to the zone file /var/named/dblab.com.zone to specify the VIP of GNS server gns.kcloud.dblab.com: 172.16.150.9 and the subdomain kcloud.dblab.com delegation to forward to GNS server gns.kcloud.dblab.com.

```
# cat /var/named/dblab.com.zone
        ◦ ◦ ◦ ◦ ◦ ◦
$ORIGIN kcloud.dblab.com.
@       IN      NS      gns.kcloud.dblab.com.
gns.kcloud.dblab.com.    IN      A       172.16.150.9
```

Step 2: On all DNS clients (all the nodes in the cluster), add these entries to /etc/resolve.conf file:

```
# cat /etc/resolv.conf
options attempts: 2
Options timeout: 1
nameserver 172.16.15.9
nameserver 172.16.168.8
search kcloud.dblab.com dblab.com
```

Here, 172.16.168.8 is the IP of the DNS server and 172.16.15.9 is the GNS server VIP of the cluster.

Step 3: Enable the "Configure GNS" option in Grid Infrastructure installation and specify the GNS configuration, including GNS subdomain and GNS VIP address. The Grid Infrastructure installation will create and configure GNS service. This will be discussed in the next session.

Network Specification in Grid Infrastructure Installation

With all the configurations and preparation work done, you can complete the network configuration through Oracle Grid Infrastructure installation. During the installation, the network configuration information, such as the public hostname and virtual hostname of each node in the cluster, the SCAN name, and the network interfaces for the public and private networks, have to be specified.

Select Whether or Not to Use GNS

If you decide to use GNS for the name resolution, you need to check the box next to "Configure GNS" on the Grid Plug and Play information page. In Oracle 12c, a single GNS can be shared by multiple clusters. When you create a new cluster, you can either share an existing GNS with other cluster by selecting "Use Shared GNS" or create a new GNS by selecting the "Create a New GNS" option. On the same page, you also need to fill out the GNS subdomain and the GNS VIP address as shown in Figure 9-11, which uses the knewrac cluster as an example.

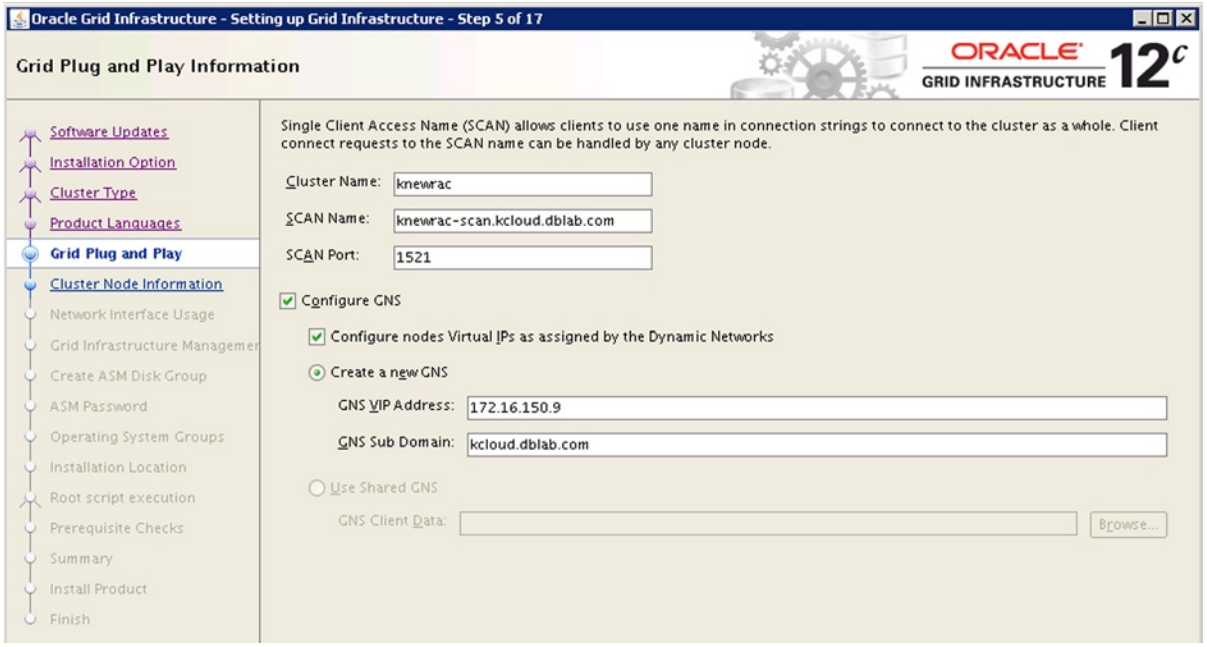


Figure 9-11. *Specifying the GNS configuration*

If you use the typical standard cluster installation, in which you by default use the DNS, you leave the "Configure GNS" box unchecked. However, if you plan to implement the Oracle Flex Cluster option in Oracle 12cR1, you are required to configure GNS with a fixed VIP on one of the hub nodes.

Refer to Chapter 4 for a detailed discussion of the network configuration for Flex Clusters and Flex ASM.

Specify Hostnames

At that point, the public hostname and virtual hostname of all the nodes in the cluster will be specified, as shown in the following knewrac cluster installation (Figure 9-12).

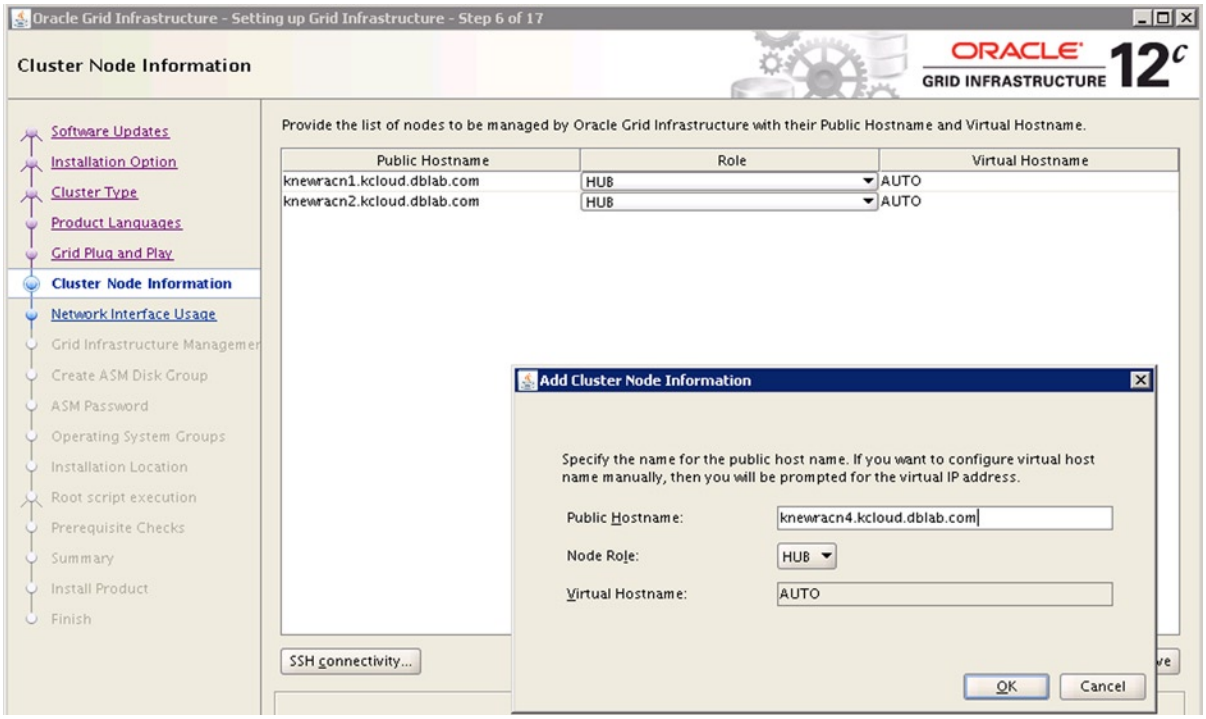


Figure 9-12. *Specifying public hostnames and virtual hostnames*

During the Flex Cluster installation, you need to specify the role of each node: hub node or leaf node.

Specify Network Interfaces

The network interfaces for public and private networks also need to be specified. Since Oracle introduced HAIP in 11.2.0.2, we can directly use multiple network interfaces (NICs) for the private interconnect configuration. Figure 9-13 shows an example of selecting two network interfaces, eth1 and eth2, for the private interconnect network.

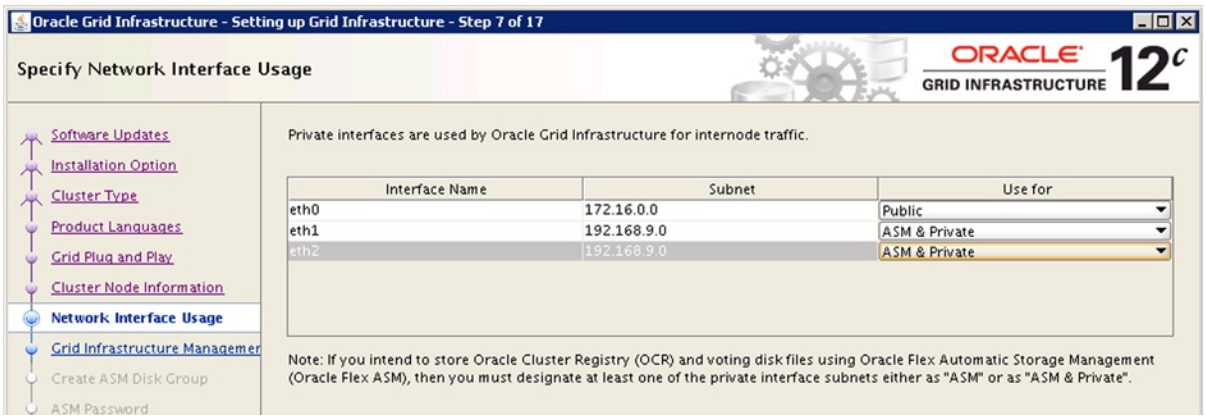


Figure 9-13. *Specifying network interfaces*

If you plan to use Oracle 12c Flex ASM, you need to specify the network interface for ASM network, which allows ASM clients to connect with the remote ASM instances. In Oracle 12cR1, you can have ASM network and Private network share the same network interfaces. Figure 9-13 shows that the ASM network and the private network share the same network interfaces, eth1 and eth2.

Verify Network Configurations

After a successful installation of the Grid Infrastructure, you can verify the network configuration.

On the knewrac cluster, the SCAN name with the VIPs dynamically generated by DHCP are as follows:

```
$ srvctl config scan
SCAN name: knewrac-scan.kcloud.dblab.com, Network: 1
Subnet IPv4: 172.16.0.0/255.255.0.0/eth0
Subnet IPv6:
SCAN 0 IPv4 VIP: -/scan1-vip/172.16.150.40
SCAN name: knewrac-scan.kcloud.dblab.com, Network: 1
Subnet IPv4: 172.16.0.0/255.255.0.0/eth0
Subnet IPv6:
SCAN 1 IPv4 VIP: -/scan2-vip/172.16.150.83
SCAN name: knewrac-scan.kcloud.dblab.com, Network: 1
Subnet IPv4: 172.16.0.0/255.255.0.0/eth0
Subnet IPv6:
SCAN 2 IPv4 VIP: -/scan3-vip/172.16.150.28
```

Notice that Oracle 12cR1 adds support for the IPv6 protocol for network configuration.

The VIPs for the public hosts on owirac-cluster cluster are as follows:

```
$ srvctl config vip -n knewracn1
VIP exists: network number 1, hosting node knewracn1
VIP IPv4 Address: -/knewracn1-vip/172.16.150.37
VIP IPv6 Address:

$ srvctl config vip -n knewracn2
VIP exists: network number 1, hosting node knewracn2
VIP IPv4 Address: -/knewracn2-vip/172.16.150.182
VIP IPv6 Address:
```

Network Configuration in Clusterware

Network configuration is stored in OCR, and Clusterware manages the network using the configuration stored in the Clusterware. Configuration details about both public and private networks are stored in the form of resources in Clusterware. Clusterware monitors network interfaces for high availability reasons.

Public Network

VIP addresses and listeners are configured in a public network. Three types of Clusterware resources with dependency between them are configured to track the network and its status. Figure 9-14 shows three types of resources implementing public network.

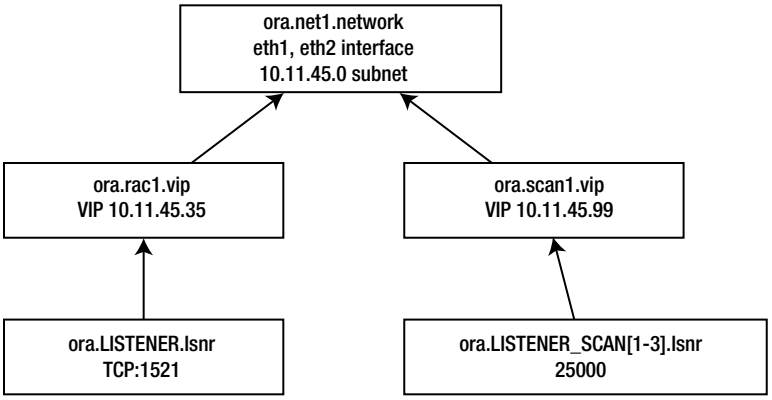


Figure 9-14. *Public network resources*

A Clusterware resource of type ora.network.type is created for each subnet. First, network resource is created with a cardinality of 1 and named `ora.net1.network` resource. The output of the following crsctl command shows a resource created for the public network. This network is owned by the root user, with oracrs userid having Read and Execute permissions.

```
$ crsctl stat res ora.net1.network -p |more
NAME=ora.net1.network
TYPE=ora.network.type
ACL=owner:root:rwx,pgrp:root:r-x,other::r--,group:oinstall:r-x,user:oracrs:r-x
...
```

The resource attribute agent_filename indicates that this resource is monitored by orarootagent process. Also, if there are errors, then the resource is restarted five times before the resource is marked offline.

```
AGENT_FILENAME=%CRS_HOME%/bin/orarootagent%CRS_EXE_SUFFIX%
...
RESTART_ATTEMPTS=5
SCRIPT_TIMEOUT=60
```

Interfaces eth1 and eth2 are configured for the public network. Further, this resource captures subnet details of the public network; thus, all IP public addresses must be in this configured subnet.

```
...
USR_ORA_IF=eth1 eth2
USR_ORA_NETMASK=255.255.254.0
USR_ORA_SUBNET=10.11.45.0
VERSION=12.1.0.0.0
```

Network configuration as discussed so far can also be easily retrieved using the srvctl command. Output of the srvctl command shows that there is a `net1` network configured in the 10.11.45.0 subnet on the eth1 and eth2 interfaces.

```
$ srvctl config network
Network exists: 1/10.11.45.0/255.255.254.0/eth1:eth2, type static
```

The VIP address is configured as a dependent resource of the ora.net1.network resource in Clusterware. The VIP resource is named ora.<node name>.vip; in this example configuration, the name of the resource is ora.rac1.vip. If the network is restarted, then the VIP resource is also restarted by the dependency-tracking mechanism. Note that the type of VIP resource is ora.cluster_vip_net1.type, and this type will be used to build dependency in listener resources.

```
$ crsctl stat res ora.rac1.vip -p
NAME=ora.rac1.vip
TYPE=ora.cluster_vip_net1.type
...
DESCRIPTION=Oracle VIP resource
ENABLED=1
...
START_DEPENDENCIES=hard(ora.net1.network) pullup(ora.net1.network)
STOP_DEPENDENCIES=hard(ora.net1.network)
```

The VIP address rac1-vip is associated with this resource. The IP address of this VIP is queried from DNS or /etc/hosts file, and that IP address must be in the same subnet as the subnet of dependent network resource (in this example, 10.11.45.0).

```
...
USR_ORA_ENV=
USR_ORA_VIP=rac1-vip
VERSION=12.1.0.0.0
```

You can also query VIP configuration using the srvctl command.

```
$ srvctl config vip  -n rac1
VIP exists: /rac1-vip/10.11.45.35/10.11.45.0/255.255.254.0/eth1:eth2, hosting node rac1
```

Listener is configured as a dependent resource of the VIP resource. The owner of Grid Infrastructure will be the owner of this resource.

```
$ crsctl stat res ora.LISTENER.lsnr -p

NAME=ora.LISTENER.lsnr
TYPE=ora.listener.type
ACL=owner:oracrs:rwx,pgrp:oinstall:rwx,other::r--
...
AGENT_FILENAME=%CRS_HOME%/bin/oraagent%CRS_EXE_SUFFIX%
...
```

Listener resource will be checked every 60 seconds, and the check command should complete in 30 seconds. Also, the port number of the listener is configured in the ENDPOINTS field of this resource. Listener resource is a dependent resource of type ora.cluster_vip_net1.type. As discussed earlier in this section, VIP resource is of type ora.cluster_vip_net1.type. Startup or shutdown of any resource with ora.cluster_vip_net1.type will trigger the listener resource to be restarted.

```
CHECK_INTERVAL=60
CHECK_TIMEOUT=30
...
ENDPOINTS=
```

```
...
ORACLE_HOME=%CRS_HOME%
PORT=1521
...
START_DEPENDENCIES=hard(type:ora.cluster_vip_net1.type) pullup(type:ora.cluster_vip_net1.type)
START_TIMEOUT=180
STATE_CHANGE_TEMPLATE=
STOP_DEPENDENCIES=hard(intermediate:type:ora.cluster_vip_net1.type)
...
USR_ORA_ENV=ORACLE_BASE=/opt/app/rac/crsbase
```

You can also query listener configuration using srvctl command. The listener is listening on port 1521 using TCP protocol.

```
$ srvctl config listener -l LISTENER
Name: LISTENER
Network: 1, Owner: oracrs
Home: <CRS home>
```

End points: TCP:1521In version 11gR2 and prior, VIP and SCAN IP addresses must be static IP addresses, meaning that VIP/SCAN/Public DNS alias to IP address mapping is statically stored in a DNS server. Version 12c supports DHCP-generated IP addresses for public, VIP and SCAN IP addresses. In version 12c, during cluster startup, DNS is queried by a VIP agent to identify IP addresses associated with DNS aliases, and retrieved IP addresses are used to configure VIP and SCAN resources. We recommend using static IP addresses for public, VIP, and SCAN IP address unless there is a valid reason to use DHCP-generated IP addresses. In environments with numerous RAC clusters, the use of GNS may be a preferred option.

In summary, the resource ora.net1.network captures the subnet and netmask details; resource ora.rac1.vip captures the VIP address and is dependent upon ora.net1.network resource; and ora.LISTENER.lsnr captures the protocol and port number details and is dependent upon ora.rac1.vip type. Any change in a resource will trigger dependent resources to be restarted or stopped.

SCAN VIP and SCAN Listener

The SCAN VIP and SCAN listener configuration is also stored as resources in Clusterware. SCAN listener simply acts as a redirection mechanism for a new connection, and VIP listeners do the heavy lifting of connection creation. (Further details about SCAN listener are given in Chapter 3, as part of the workload management discussion). The following crsctl command shows that the ora.scan1.vip resource is dependent on the ora.net1.network resource.

```
$ crsctl stat res ora.scan1.vip -p |more
NAME=ora.scan1.vip
TYPE=ora.scan_vip.type
ACL=owner:root:rwx,pgrp:root:r-x,other::r--,group:oinstall:r-x,user:oracrs:r-x
CARDINALITY=1
...
SCAN_NAME=rac1-scan
SCRIPT_TIMEOUT=60
START_DEPENDENCIES=hard(ora.net1.network) dispersion:active(type:ora.scan_vip.type)
pullup(global:ora.net1.network)
...
```

```
STOP_DEPENDENCIES=hard(ora.net1.network)
...
USR_ORA_VIP=10.11.45.99
VERSION=12.1.0.0.0
```

As the subnet value is coded in the ora.net1.network resource, SCAN VIP also should be in the same subnet as other VIP resources. All VIPs and SCAN VIPs of a network should be configured in the same subnet. You can query the SCAN details using the following srvctl command. By specifying an ordinal number with –i flag, command output is filtered to just the SCAN1 resource.

```
$ srvctl config scan -i 1
SCAN name: rac1-scan, Network: 1/10.11.45.0/255.255.254.0/eth1:eth2
SCAN VIP name: scan1, IP: /rac-scan.local.net/10.11.45.99
```

Similarly, SCAN listener is configured as a resource specifying port number and protocol. SCAN listener is a dependent resource of ora.scan1.vip.

```
$ crsctl stat res ora.LISTENER_SCAN1.lsnr -p  |more
NAME=ora.LISTENER_SCAN1.lsnr
TYPE=ora.scan_listener.type
...
PORT=25000
...
START_DEPENDENCIES=hard(ora.scan1.vip) dispersion:active(type:ora.scan_listener.type) pullup(ora.
scan1.vip)
...
STOP_DEPENDENCIES=hard(intermediate:ora.scan1.vip)
```

Up to version 11gR2, SCAN IP addresses were required to be in the first network only. Version 12c allows you to create SCAN listeners and SCAN IP addresses in the second network also.

In summary, the resource ora.net1.network captures the subnet and netmask details; resource ora.scan1.vip captures the VIP address and is dependent on ora.net1.network resource; and ora.LISTENER_SCAN1.lsnr captures the protocol and port number details, and is dependent on ora.scan1.vip type. Any state change in a resource will trigger dependent resources to be restarted or stopped.

Private Network

Private network configuration is also stored in OCR, and OCR can be stored in ASM from Database version 11g onward. However, during Clusterware startup, the CSSD daemon is started earlier than the ASM instance. So, a few critical details are required for HAS startup and are captured in an XML file and Oracle Local Registry (OLR). The following lines show cluster_interconnect details in the profile.xml file.

```
 $ cd $ORACLE_HOME/gpnp/profiles/peer/
$ grep 172 profile.xml
...
<gpnp:Network id="net3" IP="172.18.1.0" Adapter="eth3" Use="cluster_interconnect"/>
<gpnp:Network id="net4" IP="172.18.2.0" Adapter="eth4" Use="cluster_interconnect"/>
...
```

A Clusterware resource is employed to monitor the private network with the HAIP feature. The resource `ora.cluster_interconnect.haip` implements the HAIP feature for cluster interconnect, and an oraroot agent monitors these private network interfaces.

```
$ crsctl stat res ora.cluster_interconnect.haip -init  -p |more
NAME=ora.cluster_interconnect.haip
...
START_DEPENDENCIES=hard(ora.gpnpd,ora.cssd)pullup(ora.cssd)
START_TIMEOUT=60
STATE_CHANGE_TEMPLATE=
STOP_DEPENDENCIES=hard(ora.cssd)
...
USR_ORA_IF_GROUP=cluster_interconnect
```

Resource ora.cluster_interconnect.haip is also dependent upon CSSD and GPNP resource. So, this resource is started after CSSD startup. Also, notice that USE_ORA_IF_GROUP is specified as cluster_interconnect, referring to the network interface details stored in the GPNP profile.

The following output is taken from orarootagent_root log files. As you see in the following, the HAIP resource is configuring link local IP addresses on the interfaces designated for cluster_interconnect. The agent monitors these network interfaces too. If any interface fails, then the link local IP addresses are reconfigured to an available interface, thus providing high availability.

```
[ USRTHRD][16] {0:0:2} HAIP: initializing to 2 interfaces
[ USRTHRD][16] {0:0:2} HAIP: configured to use 2 interfaces
[ USRTHRD][16] {0:0:2} HAIP: Updating member info HAIP1;172.18.1.0#0
[ USRTHRD][16] {0:0:2} InitializeHaIps[ 0] infList 'inf eth3, ip 172.18.1.0, sub 172.18.1.0'
[ USRTHRD][16] {0:0:2} HAIP: starting inf 'eth3', suggestedIp '169.254.28.111', assignedIp ''
[ USRTHRD][16] {0:0:2} Thread:[NetHAWork]start {
[ USRTHRD][16] {0:0:2} Thread:[NetHAWork]start }
[ USRTHRD][17] {0:0:2} [NetHAWork] thread started
[ USRTHRD][16] {0:0:2} HAIP: starting inf 'eth4', suggestedIp '169.254.78.111', assignedIp ''
```

Prior to 11g, private network details were stored in OCR only. Changing the private network details was easier. From 11g onward, since private network configuration is stored in OCR, OLR, and profile.xml file, you need to pay special attention to the sequence of steps to modify the private network configuration. Incorrect sequencing of steps will lead to hung Clusterware startup. The high-level sequence of steps is as follows: 1) add private network interface using `oifcfg add if` command, 2) shut down the Clusterware, 3) change the interface details in the database node, 4) restart Clusterware, and then 5) delete old private network details. As the oifcfg command updates the profile.xml file and OLR, change using oifcfg command is the correct way to modify network configuration. If you have modified the private network in the OS without modifying the Clusterware, then a special procedure is needed to update the profile.xml file. Profile.xml files are signed XML files; you should not update profile.xml file directly either.

Database instance and ASM instance query Clusterware to identify the private network details and use that configuration for private network traffic. For example, the gv$cluster_interconnects view shows the interface and IP address used for private interconnect traffic. Notice that the SOURCE column is null, indicating that this is queried from Clusterware processes.

```
RS@SQL> select * from gv$cluster_interconnects order by inst_id;

INST_ID NAME          IP_ADDRESS          IS_ SOURCE
---------- -------------  ----------------          --- -------------------------------
             1 ce0:1           169.254.72.158     NO
             1 ce6:1           169.254.195.203    NO
             2 ce6:2           169.254.60.160     NO
             2 ce6:1           169.254.181.11     NO
```

You can also use specific IP addresses using the `cluster_interconnnects` initialization parameter. Usually, changes to the cluster_interconnects parameter are not needed, as the database will retrieve link local IP addresses from the Clusterware processes, so this parameter is best left untouched from 11gR2 onward. Notice that the SOURCE column indicates the source of the IP address values; in this example, values are retrieved from the cluster_interconnects database parameter.

```
cluster_interconnects                string      172.29.1.11:172.29.2.12

RS @ RAC1> select * from gv$cluster_interconnects
INST_ID NAME            IP_ADDRESS       IS_ SOURCE
---------- -------------- ---------------- --- -------------------------------
         1 aggr3          172.29.1.11      NO  cluster_interconnects parameter
         1 aggr4          172.29.2.12      NO  cluster_interconnects parameter
         3 aggr3          172.29.1.31      NO  cluster_interconnects parameter
         3 aggr4          172.29.2.32      NO  cluster_interconnects parameter
         2 aggr3          172.29.1.21      NO  cluster_interconnects parameter
         2 aggr4          172.29.2.22      NO  cluster_interconnects parameter
```

Network Failover

Network failover can happen in both public and private network interfaces. As Clusterware processes are monitoring network interfaces, failure is immediately detected, and appropriate action is taken to improve high availability.

If there is OS-based network high availability solution already configured, such as Linux Bonding, then the high availability is handled completely by the OS solution. Clusterware does not implement high availability solution in this scenario and simply uses the implemented OS solution.

If there is no OS-based solution, then HAIP can be used for cluster interconnect to implement high availability. If the private network interface fails, and if there are multiple interfaces configured for cluster interconnect, then the link local IP address from the failed interface is failed over to the available interface. Database connections might briefly run into errors during the IP reconfiguration, but those errors should quickly disappear, and in most cases, foreground processes will retry a request and succeed. After a link local IP failover to a surviving interface, Clusterware will perform re-ARP inducing a new map between an IP address and MAC address.

If all interfaces configured for cluster interconnect fail, then the link local address cannot be configured in any interface. This would also lead to heartbeat failure; so, rebootless restart or node eviction might be the result.

If a network adapter for public network fails, and if there are no other suitable adapters configured, then the VIP listener will be shut down, and VIP address will fail over to a surviving node, but no listener will be restarted on failed-over VIP. Any new connection to that failed VIP will get CONN RESET immediately, and the connection will try the next connection entry in the list.

If a network adapter for SCAN IP fails, then the SCAN IP and SCAN listener will be failed over to a surviving node.

Second Network

In 11gR2, you can configure second network for public VIPs only. However, the second network cannot be configured for cluster interconnect or SCAN network. In version 12c, the second network is supported for public, VIP, and SCAN IP addresses. Second network is useful if there are multiple subnets configured for public network traffic to the database.

The first step in adding second network resources is to add a network resource. The following command uses –k 2, indicating that this is a second network, and lists the subnet as 10.5.12.0 on aggr1 and aggr2 devices.

```
$ srvctl add network –k 2 –S 10.5.12.0/255.255.254.0/aggr1:aggr2
```

Next, the VIP on that network is added using the following srvctl commands. For all commands, -k 2 indicates that this VIP is for second network. A listener listening on second network port 1521 is also added using the add listener command.

```
$ srvctl add vip –n node1 –k 2 -A 10.5.12.99/255.255.254.0/aggr1:aggr2
$ srvctl add vip –n node2 –k 2 -A 10.5.12.100/255.255.254.0/aggr1:aggr2
$ srvctl add listener –l listener2 –p 1521 –k 2
```

The next command modifies a listener_networks parameter in the database. Database PMON or LREG process registers with the listener using this configuration. Note that for the first network, SCAN listener is specified as remote_listener, but for the second network a VIP listener in remote node is specified.

```
alter system set LISTENER_NETWORKS='((NAME=network1)(LOCAL_LISTENER=listener_net1)(REMOTE_LISTENER=r
acscan:1521))','((NAME=network2)(LOCAL_LISTENER=listener_net2) (REMOTE_LISTENER=remote_net2))';
```

In 11gR2, while the listener_networks can be used for automatic service registration with both local VIP listener and SCAN listener, it is not possible to use SCAN listener to redirect the connection for the second network. So, the second network cannot be used in conjunction with SCAN listener as of 11gR2. Version 12c allows SCAN listeners in the second network with a seamless network management of packets staying in the same subnet. Refer to Chapter 3 for exact commands to create a SCAN listener in the second network.

Further, listeners specified in the listener_networks parameter should not be specified in local_listener or remote_listener parameter, since that will lead to cross-registration between two networks.

Summary

The network is a critical piece of RAC infrastructure. Understanding network architecture, Clusterware implementation of the network, and the performance statistics of the network is of paramount importance when designing a RAC infrastructure or debugging RAC performance issues.

Pay close attention to the kernel parameters, subnet masks, network interface configuration, etc., while configuring clusters for RAC installation. It is much easier to resolve OS errors before configuring RAC. After the RAC configuration, OS level change requires further considerations.

If you have errors in the database due to lost packets or performance issues indicating network issues, then you should review network performance metrics using OS tools to identify any network configuration or performance issues. We discussed various tools such as ping, netstat, traceroute, etc., which are essential to identify the root cause of a performance issue.

CHAPTER 10

RAC Database Optimization

by Riyaj Shamsudeen

The strategy and methods employed to optimize an application in a Real Application Cluster (RAC) database are similar to those for tuning a single instance. However, in RAC, performance problems in one instance can affect the performance of other instances. In addition to single-instance wait events, RAC introduces numerous RAC-specific wait events. Time model–based tuning is employed to tune RAC databases too.

In this chapter, I discuss RAC-related wait events, statistics, Automatic Workload Repository (AWR) report sections applicable to a RAC database, etc. Almost everything in this chapter is tested with and applicable to Oracle Database release 11.2 and version 12c. There may be minor differences in earlier versions.

Introduction to Cache Fusion

To understand RAC wait events, you should first understand the internal workings of cache fusion. In RAC, buffer cache memory areas from all instances are effectively shared to create an effect of fused buffer cache. Some background processes play critical roles in managing the processing of cache fusion.

A database block holds row pieces of a database table and a change atomically modifies a database block. Database blocks are modified in the buffer cache of an instance System Global Area (SGA) (excluding direct mode writes). Since a database block can be modified in any instance, changes to the buffers in the SGA must be globally coordinated. This coordination mechanism is implemented using resources and locks; these are allocated as memory structures in the Global Resource Directory (GRD).

A resource is allocated in GRD to protect a database block. Before modifying a block, a lock on that resource must be acquired in an exclusive mode. Further, a disk read of a block into the buffer cache might require a lock on the resource too. This scheme of reads needing to acquire buffer locks (BL) is required to prevent one instance modifying the block while another instance is holding the block in its buffer cache in read-only mode.

A resource name is coined in such a way that it uniquely identifies a block. An instance is also chosen to serve as a master instance of a resource to track the resource state.

Note Chapter 11 discusses resources and locks in more detail. I will briefly introduce resources and locks in this chapter, but a much more elaborate description can be found in Chapter 11.

You should realize that cache fusion locks do not replace single-instance locks, and both single-instance and cache fusion locks coexist in RAC database. Row-level locks protect rows from concurrent changes, and similarly, cache-fusion BL locks protect buffers from inconsistent concurrent changes. Essentially, row-level locks and buffer-level locks coexist.

The global cache coordination mechanism, also known as the RAC locking scheme, is centered on the following premises:

1. A block can be resident in a modifiable mode in one instance only. Before a block is modified, a global cache lock is acquired on a resource in exclusive mode, effectively protecting the block from concurrent changes.
2. A block can be resident in Protected Read (PR) mode in multiple instances. Buffers protected with PR mode locks are considered to be in read-only mode.
3. A session must acquire the lock on a block in exclusive mode before modifying the block. Therefore, other instances must downgrade the locks to null mode before the lock is acquired in exclusive mode.
4. Buffers reconstructed for Consistent Read (CR) mode do not require any BL locks to be held.

Cache Fusion Processing

In a single instance, a database block can reside only in the local buffer cache. Therefore, if the block is not in the local buffer cache, it is read from the disk. In RAC, the block can be resident in any buffer cache. So, a special type of processing takes place that deviates sharply from single-instance processing.

The following section provides a rough outline of the cache fusion-processing scheme. This discussion should be considered as an outline of cache fusion processing, not necessarily a documentation of the complete algorithm.

Figure 10-1 shows the mechanics of a cache fusion block transfer.

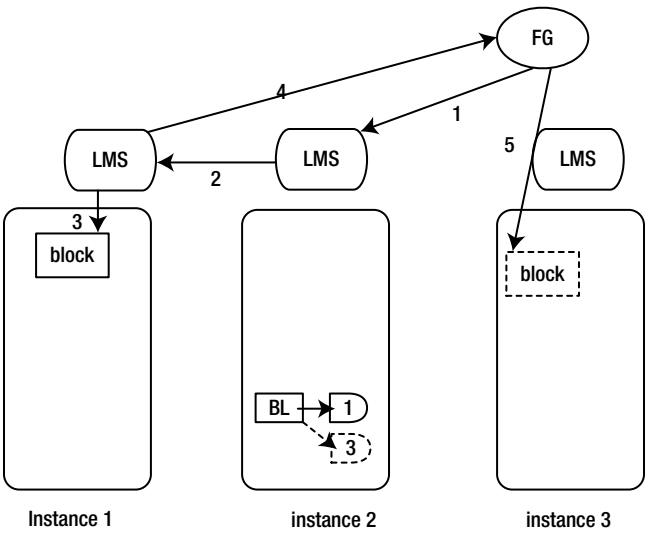


Figure 10-1. *Cache fusion processing 3-way*

Each of the numbered steps is explained below. Items with dotted lines in the picture indicate the state after the block transfer.

1. A foreground (FG) process (also known as a server process) tries to access a database block and the data block is not in the local buffer cache. A hashing algorithm[1] is applied on the resource_name protecting the block to calculate the resource master instance. In Figure 10-1, the resource master instance for this resource is 2.

    a. The FG process retrieves the network address of a remote LMS background process running on the resource master instance.

    b. The FG process constructs a message for the block access and sends that message to the remote LMS process.

    c. The FG process must wait until a response is received from the remote LMS process. FG process cannot predict the LMS response since the block could be in a remote buffer cache or the block may not be resident in any buffer cache. So, the FG process waits on a *placeholder* wait event.

2. LMS process in the resource master instance receives the message, identifies the block, and accesses GRD to identify the current state of the resource.

    a. For CURRENT mode request, if the block is not in any buffer cache, the LMS process allocates a new resource in GRD; then allocates a lock on the resource;[2] and then sends a message back to the FG process with a *grant* to read the block from the disk.

    b. For CR mode requests, GRD representation of resource is not required. The block may be simply reconstructed without requiring additional lock conversion.

3. In Figure 10-1, the block is resident in the buffer cache of instance 1. So, an LMS process running on the *resource master* instance sends a request to the LMS process running on the *owning* instance (instance 1).

4. LMS process running in owning instance constructs a block and sends the block to the FG process running in instance 3.[3]

5. The FG process receives the block, copies the block to the instance 3 buffer cache, and continues processing further.

After the transfer of a block, LMS process running on the resource master instance also modifies the resource and adds a lock indicating that the block is now resident in both instances 1 and 3. Obviously, the LMS background process plays a critical role in the cache fusion process and it is the workhorse process behind cache fusion processing.

---

[1]This algorithm is a variant of hashing algorithm. In a nutshell, in a three-instance database, the first contiguous 256 blocks of a data file will be mastered by instance 1, the next 256 blocks will be mastered by instance 2, the third 256-block range is to be mastered by instance 3, and the fourth 256-block range is to be mastered by instance 1, in a cyclic way. This blocking factor is controlled by _lm_contiguous_res_count parameter and defaults to 256 in 11g (in 10gR2, defaults to 128). DRM is a feature that alters the round-robin mastership and is discussed later in this chapter.

[2]Note that this algorithm is a rough outline; the actual algorithm is more complicated. The action depends upon the request type CR or current mode, whether the buffer is in buffer cache of resource master, whether the buffer is in another instance buffer cache, whether downgrade is possible or not, etc.

[3]The Wireshark utility is a great tool to trace this activity. You can see that FG (server process) receives the blocks directly from LMS process from 10gR2 onward.

Even though an individual session requests locks, cache fusion locks are owned by instances, not individual sessions. Let's say that another process in instance 3 accesses the same block: there is *no* additional cache fusion processing needed, as the lock is already held in a compatible mode. As long as the lock held is in compatible mode, any process can access the block locally without additional cache fusion processing. Of course, if the block is not in a compatible mode, then the block lock mode must be upgraded.

■ **Note** In this chapter, I am using the term "global cache lock *requests*." The more accurate term is "global cache lock *conversions*." Initially, locks are acquired in NULL mode and then converted to another higher-level mode such as exclusive or PR mode. Use of the term "global cache lock request" is much more readable than "global cache lock conversion." This difference in terminology does not matter, but if you are reading internal traces this difference becomes obvious.

GRD

A combination of file_id and block_id uniquely identifies a block; BL resource names are coined using the combination of file_id and block_id of the block. A lock on that BL resource is acquired before altering the block or reading from the disk.

Let me explain the BL locking mechanism with a small table. The following code creates a table t_one and populates the table with 500 rows, adds an index, and then collects statistics on the table. With this setup, I will query a row to show how BL resources are employed to maintain cache fusion consistency.

```
DROP TABLE t_one;
CREATE TABLE t_one (n1 NUMBER , v1 VARCHAR2(100));
INSERT INTO t_one
SELECT n1, lpad (n1, 100,'DEADBEEF')
FROM
  ( SELECT level n1 FROM dual CONNECT BY level <=500
  );
COMMIT;
CREATE INDEX t_one_n1 ON t_one (n1);
EXEC dbms_stats.gather_table_stats ( USER, 't_one', CASCADE =>true);
```

BL Resources and Locks

The query printed below has an index range scan access path, so index block is read using the predicate n1=100, and then the table block is read using the rowid retrieved from the index entry. For this discussion, I will exclusively focus on BL resources and locks protecting the table data block, although a similar locking scheme is applied to index block also. As BL resource_name is derived using the combination of file_id and block_id, I will query the file_id and block_id of a block using dbms_rowid package.

```
-- script tc_one_row.sql --
SELECT n1,
  dbms_rowid.rowid_to_absolute_fno (rowid, user,'T_ONE') fno,
  dbms_rowid.rowid_block_number(rowid) block,
  dbms_rowid.rowid_object(rowid) obj,
  LENGTH(v1) v1
```

```
FROM t_one
WHERE n1=100;

N1         FNO          BLOCK       OBJ         V1
---------- ---------- ---------- ---------- ----------
       100          4        180      75742        250
```

■ **Note** The previous SQL statement has `length(v1)` phrase in the `SELECT` list. That column is needed to force reading the data block of the table. Query execution will read the block from the disk into the buffer cache.

Resource_name starts with a format [0xblock_id][0xfile_id],BL, where both block_id and file_id are converted to hexadecimal format.[4] The query printed below constructs a resource_name protecting the block using dbms_rowid package. This resource_name will be used to query GRD to view resources and locks.

```
SELECT DISTINCT '[0x'||trim(TO_CHAR(dbms_rowid.rowid_block_number(rowid), 'xxxxxxxx'))
    ||'][0x'|| trim(TO_CHAR(dbms_rowid.rowid_to_absolute_fno (rowid,user,'T_ONE'),'xxxx'))
    ||'],[BL]' res
FROM t_one
WHERE n1=100;
RES
---------------------------
[0xb4][0x4],[BL]
```

With the derived resource_name, I will query gv$ges_resource to print the GRD *resource* protecting the block. The output of the query below shows that this resource is visible in inst_id=2 (I am connected to instance 2). Master node of the resource is 1 (Master_node starts with 0, so master_node=1 is inst_id=2). This resource protects the block with (file_id=4 and block_id=180). Column on_convert_q shows the count of processes waiting in the converting queue to acquire locks, indicating the count of lock waiters. Column on_grant_q shows the count of processes holding locks, essentially, a count of lock holders.

```
RS@ORCL2:2> SELECT resource_name, ON_CONVERT_Q, ON_GRANT_Q, MASTER_NODE
FROM gv$ges_resource
WHERE resource_name LIKE '[0xb4][0x4],[BL]%'
/
RESOURCE_NAME                  ON_CONVERT_Q ON_GRANT_Q MASTER_NODE
------------------------------ ------------ ---------- -----------
[0xb4][0x4],[BL][ext 0x0,0x0]             0          1           1
```

gv$ges_enqueue externalizes the *locks* held in GRD. The following query accesses gv$ges_enqueue to print lock details. We can see that owner_node column is set to 1 (column owner_node starts with 0, and so owner_node=1 is inst_id=2). Essentially, in this example, the resource is mastered by inst_id=2 as shown in the output of gv$ges_resuorce and lock on that resource is also owned by the same instance. (Note that SQL statements in the section are available in downloadable scripts Listing_10-1.sql).

[4]Note that in Database version 12c, the pluggable databases concept is introduced, and multiple pluggable databases can share the same buffer cache. Since (file_id , block_id) combination is unique even when multiple databases are plugged in a container database, so, BL resource_name format remains the same in 12c also.

Listing 10-1. Gv$ges_enqueue Output

```
col state format a15
RS@ORCL2:2>SELECT resource_name1, grant_level, state, owner_node
FROM v$ges_enqueue
WHERE resource_name1 LIKE '[0xb4][0x4],[BL]%'
/
RESOURCE_NAME1                 GRANT_LEV STATE      OWNER_NODE
------------------------------ --------- ---------- ----------
[0xb4][0x4],[BL][ext 0x0,0x0]  KJUSERPR  GRANTED             1
```

Connecting to instance ORCL1 (with inst_id=1), I will query the same row to read data blocks into instance 1 buffer cache.

```
RS@ORCL1:1> SELECT n1,
  dbms_rowid.rowid_to_absolute_fno (rowid, 'RS','T_ONE') fno,
  dbms_rowid.rowid_block_number(rowid) block,
  dbms_rowid.rowid_object(rowid) obj,
  LENGTH(v1) v1
FROM rs.t_one
WHERE n1=100;

N1         FNO        BLOCK      OBJ        V1
---------- ---------- ---------- ---------- ----------
       100          4        180      75742        250
```

Accessing gv$ges_resource, as printed in the following query output, we see that block is still mastered by instance 1.

```
RS@ORCL2:2> SELECT resource_name, ON_CONVERT_Q, ON_GRANT_Q, MASTER_NODE
FROM gv$ges_resource
WHERE resource_name LIKE '[0xb4][0x4],[BL]%';

RESOURCE_NAME                  ON_CONVERT_Q ON_GRANT_Q MASTER_NODE
------------------------------ ------------ ---------- -----------
[0xb4][0x4],[BL][ext 0x0,0x0]             0          1           1
```

Accessing gv$ges_enqueue, there are two locks on that resource, which are owned by instances 1 and 2 in KJUSERPR (PR) mode. Since sessions are accessing the block in read-only mode, locks are acquired in PR (KJUSERPR) mode. Both locks can be held in the KJUSERPR mode, as the lock mode KJUSERPR is compatible with another KJUSERPR mode.

KJUSERPR mode protects concurrent changes to the block. So, if another session connected to instance 3 tries to change the block, then both instances 1 and 2 must downgrade the block before instance 3 can acquire the lock on the resource in exclusive mode.

```
RS@ORCL2:2>SELECT resource_name1, grant_level, state, owner_node
FROM v$ges_enqueue
WHERE resource_name1 LIKE '[0xb4][0x4],[BL]%'
/
RESOURCE_NAME1                 GRANT_LEV STATE      OWNER_NODE
------------------------------ --------- ---------- ----------
[0xb4][0x4],[BL][ext 0x0,0x0]  KJUSERPR  GRANTED             1
[0xb4][0x4],[BL][ext 0x0,0x0]  KJUSERPR  GRANTED             0
```

Figure 10-2 represents the resource and locks discussed in this section. The resource master instance has a BL resource allocated to protect the block in discussion. Two GCS (Global Cache Services) locks also have been acquired. However, there is a minor distinction to be made between these two GCS locks, namely, GCS client and GCS shadow: if the buffer represented by the resource is resident in the local buffer cache, then the corresponding GCS lock structure is known as the GCS client; if the buffer is resident in the remote buffer cache, then the lock structure is known as GCS shadow. A GCS client points to lock element, which in turn points to a buffer header structure. The buffer header structure holds a pointer to the buffer in the buffer cache.

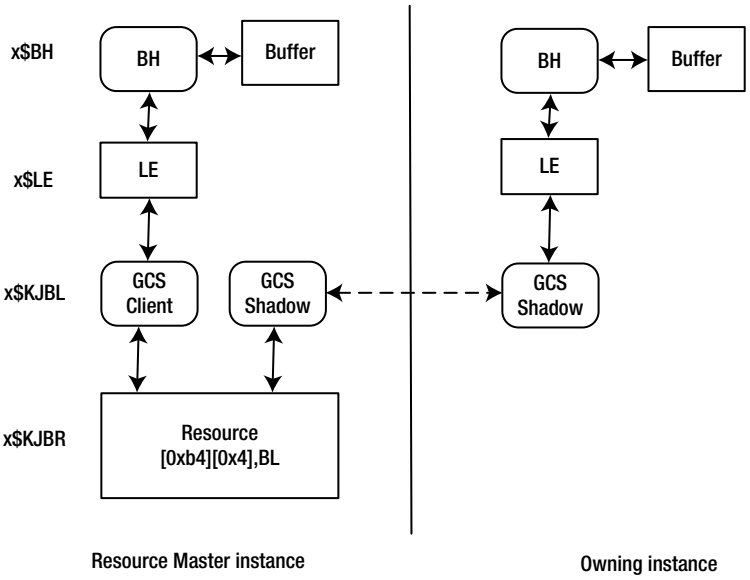


Figure 10-2. *Resources and locks*

Fixed table x$kjbr externalizes resource structures, x$kjbl externalizes BL locking structures, x$le externalizes the lock elements, and x$bh externalizes the buffer header structures.

The query in Listing 10-2 can be used to probe the internal structures further. (I should point out that this SQL statement has no practical value, it is provided purely to help you understand the internal structures.) In 12c, this statement must be executed in a container database (CDB) if the database is a CDB. You may be able to research the internal structures further using the script in Listing 10-2.

Listing 10-2. Lock_to_buffer

```
set lines 120
col kjblname format a30
col kjblname2 format a20
col kjblgrant format a10
col kjblrole format 9999
col kjblrequest format A10
set verify off
select /*+ leading (c a b ) */ b.kjblname, b.kjblname2 ,
b.kjblgrant, b.kjblrole, b.kjblrequest , b.kjblmaster, b.kjblowner
from x$le a , x$kjbl b, x$bh c
```

```
where a.le_kjbl = b.kjbllockp
and a.le_addr =c.le_addr
and dbablk=&&block and obj=&&obj
;
KJBLNAME                  KJBLNAME2           KJBLGRANT  KJBLROLE KJBLREQUES KJBLMASTER
------------------------- ------------------- ---------- -------- ---------- ----------
[0xb4][0x4],[BL][ext 0x0,0 180,4,BL           KJUSEREX          0 KJUSERNL            0
```

In summary, a BL resource protects a block, the action of reading a block into buffer cache creates a resource in the resource master instance, and a lock is added to that resource in a specific mode. In this example, the block is read with no intent to change and hence KJUSERPR locking mode (PR) is sufficient. We will see that locking mode will be different depending upon whether the block is altered or not. Locking modes are also discussed in Chapter 11.

Performance Analysis

Typically, performance tuning an application in a RAC database is geared towards identifying the layer and the instance causing slowdown. You should ask the following questions to understand the problem further. While few questions would generally apply to single-instance performance, these questions are specifically written for a RAC database.

1. What are the wait events the application is suffering from? Analysis of the waits suffered by the FG processes typically focus on the receiving side of global cache performance.
    a. Are there numerous occurrences of wait events?
    b. Is the average wait time for an event huge?
2. Is the *sending side* of global cache performing efficiently? Is there a delay in global cache processing?
3. Is the network latency causing performance issues? Network will be discussed in Chapter 9.
4. Can the performance problem isolated to an instance? Is that instance suffering delays in the receiving side or the sending side?

Each of these questions can be answered by reviewing wait events or statistics. You can isolate the problem to a specific instance, component, or object using the methods discussed in this section.

Analysis of the Receiving Side

This section focuses on the receiving side of global cache performance metrics. RAC introduces a few wait events for time model instrumentation. You would use AWR or another time-based model to understand the wait events causing performance issues. Then, tune the top event consuming time.

Generally, wait events can be grouped into the following categories:

1. **Current mode wait events:** This wait event is incurred if the version of transferred or requested block is the latest. Current mode wait events can be encountered for both read and write requests. You should note that the block version of the disk could be the latest version, and hence the sheer action of reading a block from the disk can acquire locks in the current mode.
2. **CR mode wait events:** If the current version of the block is ahead of Query environment System Change Number (SCN), then the LMS process must reconstruct a block version for read-only purposes and then send the block to the FG process. This type of block transfer is known as CR mode transfer.

3. **Busy wait events:** Time is accounted to these events if the block transfer suffered from additional processing delays encountered by LMS process. For example, LMS process may have applied undo records to create CR mode blocks.

4. **Congestion-related events:** These events imply that delays were incurred since the LMS process was too busy servicing other requests or the LMS process was suffering from a resource starvation.

Wait events can be further grouped depending on the number of instances participating in a block transfer. Up to three instances can participate in a block transfer. Requesting instance is the instance requesting the block, master instance tracks the resource status, and the owner instance currently holds a lock on the resource. If three instances participate in a block transfer, then that block transfer is tagged with 3-way wait events. In some cases, the resource master instance might be the owner instance too. In that case, only two instances participate in a block transfer; hence, those wait events are tagged with 2-way wait events.

A Generic Analysis for all Wait Events

You can identify the object, sql_id, or distribution of wait time for any wait event. This section provides general guidance to understand the details of all RAC wait events. Use this section in conjunction with individual RAC wait events to understand the performance bottlenecks better.

Identify Object

Active Session History (ASH) can be queried to identify the objects suffering from wait events. Listing 10-3 introduces script ash_gcwait_to_obj.sql to identify the object_name.

Listing 10-3. Script: ash_gcwait_to_obj.sql

```
col owner format a30
col object_name format a30
set lines 160
WITH ash_gc AS
  (SELECT inst_id, event, current_obj#, COUNT(*) cnt
  FROM gv$active_session_history
  WHERE event=lower('&event')
  GROUP BY inst_id, event, current_obj#
  HAVING COUNT (*) > &threshold )
SELECT * FROM
  (SELECT inst_id, nvl( owner,'Non-existent') owner ,
           nvl ( object_name,'Non-existent') object_name,
           nvl ( object_type, 'Non-existent') object_type,
           cnt
  FROM ash_gc a, dba_objects o
  WHERE (a.current_obj#=o.object_id (+))
  AND a.current_obj#  >=1
  UNION
  SELECT inst_id, '', '', 'Undo Header/Undo block', cnt
  FROM ash_gc a WHERE a.current_obj#=0
  UNION
  SELECT inst_id, '', '', 'Undo Block', cnt
```

```
  FROM ash_gc a
  WHERE a.current_obj#=-1
  )
ORDER BY cnt DESC
/
Enter value for event: gc current block 2-way
Enter value for threshold: 30

INST_ID OWNER                OBJECT_NAME                    OBJECT_TYPE          CNT
------- -------------------- ------------------------------ ------------- ----------
      3 PO                   RCV_SHIPMENT_LINES             TABLE               2228
      2 PO                   RCV_SHIPMENT_LINES             TABLE               2199
      1 PO                   RCV_SHIPMENT_LINES             TABLE               2197
      3 PO                   PO_REQUISITION_LINES_ALL       TABLE               2061
      2 PO                   PO_REQUISITION_LINES_ALL       TABLE               1985
      3                                                     Undo Block           120
...
```

The output of Listing 10-3 shows the tables suffering from waits for the `gc current block 2-way` wait event. You can supply any event name to identify the objects suffering from that event. If the object_id is 0 or -1, then it indicates undo block or undo segment header block. If the object was dropped, then the object_name will be marked as *non-existent*.

Also, to improve the performance of this query, set a higher threshold limit such as 100 or 1,000 in a busy database.

■ **Note** ASH data is sampled, so the number of samples must be high enough for the data to be accurate. Also, you will need a Diagnostic & Tuning Pack license to access ASH data in release 12c and earlier.

In version 12c, the Pluggable Database (PDB) feature is introduced. As the user objects can be in a PDB instead of the CDB, the code shown in Listing 10-3 needs to be adjusted to access cdb_objects view. Listing 10-4 shows the modified script, and in the output below, con_id column indicates the PDB container_id and object inducing global cache wait events. You can identify the PDB name by joining cdb_pdbs view. In my test cluster, container_id of hrdb1 PDB is 3.

Listing 10-4. Script: ash_gcwait_to_obj_12c.sql

```
col owner format a30
col object_name format a30
set lines 160
WITH ash_gc AS
  (SELECT inst_id, event, current_obj#,con_id, COUNT(*) cnt
  FROM gv$active_session_history
  WHERE event=lower('&event')
  GROUP BY inst_id, event, current_obj#, con_id
  HAVING COUNT (*) > &threshold )
SELECT * FROM
  (SELECT inst_id, a.con_id, nvl( owner,'Non-existent') owner ,
          nvl ( object_name,'Non-existent') object_name,
          nvl ( object_type, 'Non-existent') object_type,
          cnt
```

```
  FROM ash_gc a, cdb_objects o
  WHERE (a.current_obj#=o.object_id (+))
  AND a.current_obj#  >=1
  AND a.con_id =o.con_id (+)
  UNION
  SELECT inst_id,0, '', '', 'Undo Header/Undo block', cnt
  FROM ash_gc a WHERE a.current_obj#=0
  UNION
  SELECT inst_id,0, '', '', 'Undo Block', cnt
  FROM ash_gc a
  WHERE a.current_obj#=-1
  )
ORDER BY cnt DESC
/
Inst Cont OWNER OBJECT_NAME                    OBJECT_TYPE            CNT
---- ---- ----  ------------------------------ ---------------        ---
   2    3 RS    HUGETABLE                      TABLE                   21
   2    3 RS    HUGETABLE_HASH                 TABLE PARTITION          9
...
```

Identify SQL Statement

ASH data can also be queried to identify the SQL statement associated with events. Listing 10-5 shows the script ash_gcwait_to_sql_id.sql to identify the SQL statement associated with the wait event. The output in Listing 10-5 shows that some SQL statements were inducing waits for these events. Further review of these SQL statements, access plans, and performance metrics would be required to diagnose the performance problem.

Listing 10-5. Script: ash_gcwait_to_sql_id.sql

```
WITH ash_gc AS
  (SELECT    /*+ materialize */ inst_id, event, sql_id, COUNT(*) cnt
  FROM gv$active_session_history
  WHERE event=lower('&event')
  GROUP BY inst_id, event, sql_id
  HAVING COUNT (*) > &threshold )
SELECT inst_id, sql_id, cnt FROM ash_gc
ORDER BY cnt DESC
/
Enter value for event: gc current block 2-way
Enter value for threshold: 100

   INST_ID SQL_ID               CNT
---------- ------------- ----------
         3 26shktr5f1bqk       2717
         3 4rfpqz63y34rk       2332
         2 4rfpqz63y34rk       2294
 ...
```

With the PDB feature, the script shown in Listing 10-6 must also be modified to retrieve container_id of the PDB. Column con_id can be used to identify PDB executing costly SQL statements.

Listing 10-6. Script: ash_gcwait_to_sql_id_12c.sql

```
WITH ash_gc AS
  (SELECT    /*+ materialize */ inst_id, con_id,event, sql_id, COUNT(*) cnt
  FROM gv$active_session_history
  WHERE event=lower('&event')
  GROUP BY inst_id, event, sql_id, con_id
  HAVING COUNT (*) > &threshold )
SELECT inst_id,con_id, sql_id, cnt FROM ash_gc
ORDER BY cnt DESC
/
```

Note that PDB is an optional feature in version 12c, and if your database is a non-CDB, then the script in Listing 10-4 is sufficient.

Understanding Wait Distribution

You should also identify the histogram of wait time, as averages can be misleading. Are a few occurrences of longer waits for an event causing an increase in average response time? Or is the average wait time itself elevated? AWR reports provide a section for wait event histograms too.

In Listing 10-7, histogram information about wait event is retrieved by querying gv$event_histogram view. The following output shows that 99% of the waits for gc cr block 2-way wait event complete in under 4 ms. This output is queried from a healthy database, and wait time is within the acceptable range.

Listing 10-7. Script: event_histogram.sql

```
SELECT inst_id, event, wait_time_milli, wait_count,
      TRUNC(100*(wait_count/tot),2) per
FROM
  (SELECT inst_id, event, wait_time_milli, wait_count,
    SUM (wait_count) over(partition BY inst_id, event
                  order by inst_id
                  rows BETWEEN unbounded preceding AND unbounded following
                  ) tot
  FROM
    (SELECT * FROM gv$event_histogram
     WHERE event LIKE '%&event_name%'
     ORDER BY inst_id, event#, WAIT_TIME_MILLI
    )
  )
ORDER BY inst_id, event, WAIT_TIME_MILLI ;
Enter value for event_name: gc cr block 2-way

   INST_ID EVENT                          WAIT_TIME_MILLI WAIT_COUNT        PER
---------- ------------------------------ --------------- ---------- ----------
         1 gc cr block 2-way                            1     105310       2.03
         1 gc cr block 2-way                            2    3461802      66.87
         1 gc cr block 2-way                            4    1593929      30.79
         1 gc cr block 2-way                            8      15163        .29
         1 gc cr block 2-way                           16        340          0
         1 gc cr block 2-way                           32         26          0
...
```

Remember that gv$event_histogram gives performance metrics from the start of an instance and could be misleading. So, it is crucial to review the performance metrics during a short period of time using AWR reports or custom scripts. The query in Listing 10-8 shows the wait histogram for the past day. This query uses AWR tables to retrieve the wait_time for an event.

Listing 10-8. Script: event_hist_from_awr.sql

```
set lines 160
col begin_interval_time format a30
col event_name format a30
col instance_number format 99 head "Inst"
break on begin_interval_time
SELECT snaps.begin_interval_time,
  snaps.instance_number,
  hist.event_name,
  hist.wait_time_milli,
  hist.wait_count
FROM dba_hist_event_histogram hist, DBA_HIST_SNAPSHOT snaps
WHERE snaps.snap_id           = hist.snap_id
AND snaps.instance_number     = hist.instance_number
AND snaps.begin_interval_time > sysdate-1 -- modify to specify constrained time window
AND hist.event_name           = lower('&event_name')
ORDER BY snaps.snap_id ,
  instance_number,
  wait_time_milli
/

BEGIN_INTERVAL_TIME            Inst EVENT_NAME           WAIT_TIME_MILLI WAIT_COUNT
------------------------------ ---- -------------------- --------------- ----------
01-NOV-12 05.00.15.638 PM         1 gc cr block 2-way                 1     110970
                                  1 gc cr block 2-way                 2    3620790
                                  1 gc cr block 2-way                 4    1708432
                                  1 gc cr block 2-way                 8      15685
                                  1 gc cr block 2-way                16        340
                                  1 gc cr block 2-way                32         26
                                  1 gc cr block 2-way                64          8
                                  1 gc cr block 2-way               128          1
                                  1 gc cr block 2-way               256          1
                                  1 gc cr block 2-way              2048          1
```

Note that this query retrieves performance metrics for a period of one day. If you are analyzing a performance problem for a constrained time frame, then modify the query to retrieve data for that problem time frame. If the problem is constrained to a smaller window of time, then reviewing the performance metrics over a longer period of time can be misleading.

Eliminate the Network as Root Cause

Another action item is to eliminate the network as a problem. A few RAC wait events serve as a baseline and can be effectively used to eliminate the network layer being the probable root cause of a performance problem. There are two events that primarily can be used as baseline events. These events do not suffer from busy or congestion-related waits. So, if the average wait time for these baseline events is in a 2-4 ms range, that would imply that the network can be eliminated as an issue. This technique is useful when you are debugging a stressful production performance issue.

Event `gc cr grant 2-way` is a lightweight event. It involves a round-trip message to a remote LMS process and the remote LMS process responding with a message to read the block from the disk. Both messages are typically 300 bytes in size. The average wait time for this event is usually less than 2 ms. If the distribution of wait time for this event indicates that 90% of waits are less than 2 ms, then you can safely say that network *may* not be a problem. If the wait time is much higher than 2 ms, then it could be a network problem or GC processing problem. This event measures the baseline performance of a small packet transfer.

Another such event is `gc current block 2-way or gc cr block 2-way`. Time is accounted to these two events if a database block was transferred from one instance to the FG process by an LMS process without incurring any additional performance delays. Typically, the average wait time for this event is 2-4 ms. If the event histogram for this event is mostly in the lower wait buckets, then you can safely eliminate the network as a problem. The message size for these two wait events is usually big, as these messages will transfer one database block. As these two events generally transfer big packets (~1 database block), they will serve to verify if the problem is with the Jumbo Frames setup (if Jumbo Frames is in play).

Identify Problem Instances

When you have a performance problem, you can identify problem instances by reviewing gv$instance_cache_transfer view. This view provides detailed map of receive time by class, receiving, and sending instance level. Using the query listed in Listing 10-9, you can quickly identify the instance causing elevated GC latencies. In the following output, there is no problem: both average CR receive time and average CUR receive time have the approximately the same values in all instances.

Listing 10-9. Script: gc_problem_instance.sql

```
SELECT instance ||'->' || inst_id transfer,
  class,
  cr_block cr_blk,
  TRUNC(cr_block_time     /cr_block/1000,2) avg_Cr,
  current_block cur_blk,
  TRUNC(current_block_time/current_block/1000,2) avg_cur
FROM gv$instance_cache_transfer
WHERE cr_block >0 AND current_block>0
ORDER BY instance, inst_id, class
/

TRANS CLASS                  CR_BLK     AVG_CR    CUR_BLK    AVG_CUR
----- ------------------ ---------- ---------- ---------- ----------
...
1->2  data block           51035711       1.57  930591874        1.6
...
1->2  undo block           21548056       1.05         70       1.82
1->2  undo header          17073000       1.01     272653        1.2
...
1->3  data block           35554782       1.67  804269141       1.78
...
1->3  undo block           16125459       1.06          1       2.03
1->3  undo header          25643014        .99     270995       1.15
...
2->1  data block          178936080       1.44 1382236663       1.55
```

Gv$instance_cache_transfer view is an ultracritical view to quickly identify the instance suffering from the performance issue. If there are problems with a huge cluster, I usually drill down to the instance suffering from performance issues quickly using this view.

The following output shows that instance 3 is causing global cache slowdown. Blocks received from instance 3 have an average wait time of about 20 ms, while the average wait time for the blocks received from other instances is ~2 ms. You should probe instance 3 to identify the reason for slow block transfers from that instance.

```
TRANS CLASS                 CR_BLK     AVG_CR    CUR_BLK    AVG_CUR
----- ------------------ ---------- ---------- ---------- ----------
1->2  data block          87934887       1.23    9834152        1.8
2->1  data block          28392332       1.30    1764932        2.1
...
3->1  data block          12519985      11.57    2231921       21.6
...
3->2  undo block           4676398       8.85        320      27.82
```

Again, SQL statements shown in Listing 10-9 are accessing gv$instance_cache_transfer view, a view of metrics from the start of an instance. We need to review metrics in a more granular fashion, so review AWR report or use my script gc_instance_cache_tfr.sql in the downloadable script format.

RAC Wait Events

I will discuss important RAC wait events in this section. This is not a complete list of all wait events, but just a list of most frequently encountered wait events.

GC Current Block 2-Way/3-Way

Current block wait events indicate that the transferred block version is the latest version of the block. This wait event can be encountered for both read and write activities. If the block is accessed for read activity, then a lock on a resource is acquired in KJUSERPR (*PR*) mode. The example I discussed earlier in the "Resources and Locks" section shows the locks in KJUSERPR mode.

In the following example, I queried a row from table t_one, resulting in a disk read connecting to node 2. Reviewing the SQL trace file, there are no global cache wait events. The reasoning is that the block is mastered locally, and so the FG process can acquire locks on the resources directly without incurring any global cache wait. This type of locking is also called the affinity-locking scheme. The Dynamic Resource Mastering (DRM) section will discuss affinity locking in detail.

```
RS@ORCL2:2> @tc_one_row.sql

        N1        FNO      BLOCK        OBJ         V1
---------- ---------- ---------- ---------- ----------
       100          4        180      75742        250

Trace file:

nam='db file sequential read' ela= 563 file#=4 block#=180 blocks=1 obj#=75742
```

I queried the same row connecting to instance 1.[5] Since that block is cached in instance 2 already, the block is transferred from instance 2 to 1. Trace line shows that a wait event `gc current block 2-way` was encountered for the

[5]Notice that I am querying the block from a different instance. There is a RAC optimization by which block locks are kept local and globalized only when the block is accessed in a different instance. So, if you access the block in an instance, global cache structures may not be set up yet. That's the reasoning behind accessing the block in a different instance.

block with file_id=4, block_id=180. This block transfer is a 2-way block transfer, as the owning instance is same as the resource master instance (instance 2).

```
SYS@ORCL1:1> @tc_one_row.sql

        N1        FNO      BLOCK        OBJ         V1
---------- ---------- ---------- ---------- ----------
       100          4        180      75742        250

Trace file:

nam='gc current block 2-way' ela= 629 p1=4 p2=180 p3=1 obj#=75742 tim=1350440823837572
```

Next, I will create conditions for a 3-way wait event, but first let me flush buffer cache in all three instances to start with a clean buffer cache. In the following example,

1. I will query the row connecting to instance 1 (which would load the block into the buffer cache of instance 1).
2. I will connect to an instance and query the same row from instance 3. The FG process for my session will request LMS process running in instance 2 for the block (as instance 2 is the resource master).
3. Instance 2 LMS process forwards the request to LMS process in instance 1.
4. An LMS process in instance 1 will send the block to the FG process in instance 3.

Essentially, three instances participate in a block transfer; hence, this is a 3-way block transfer.

```
-- alter system flush buffer_cache; from all instances
RS@ORCL1:1> @tc_one_row.sql

        N1        FNO      BLOCK        OBJ         V1
---------- ---------- ---------- ---------- ----------
       100          4        180      75742        250

RS@ORCL3:3> @tc_one_row.sql
        N1        FNO      BLOCK        OBJ         V1
---------- ---------- ---------- ---------- ----------
       100          4        180      75742        250

    Trace file:

nam='gc current block 3-way' ela= 798 p1=4 p2=180 p3=1 obj#=75742
```

Reviewing locks on the resource connecting to instance 2, we see that there are two locks on the resources held for instance 1 and instance 3 (owner_node equals to 0 and 2).

```
RS@ORCL2:2> SELECT resource_name1, grant_level, state, owner_node
     FROM v$ges_enqueue
     WHERE resource_name1 LIKE '[0xb4][0x4],[BL]%';
```

```
RESOURCE_NAME1                 GRANT_LEV STATE         OWNER_NODE
------------------------------ --------- ------------- ----------
[0xb4][0x4],[BL][ext 0x0,0x0]  KJUSERPR  GRANTED                2
[0xb4][0x4],[BL][ext 0x0,0x0]  KJUSERPR  GRANTED                0
```

Excessive waits for gc current block 2-way or gc current block 3-way wait event are generally either due to (a) an inefficient execution plan, leading to numerous block visits, or (b) application affinity not being in play. Consider implementing application affinity if the object access is localized. Also, use the techniques discussed earlier in the section "A generic analysis for all wait events."

GC CR Block 2-Way/3-Way

CR mode block transfer is requested for read-only access. Consider that a block is resident in *current* mode in instance 2; instance 2 holds the exclusive mode BL lock on the resource. Another session connected to instance 1 requests that block. As the "readers do not see uncommitted changes" in Oracle Database, SELECT statements request a specific version of the block as of the query start time. SCN is employed for block versioning; essentially, SELECT statement requests a version of a block consistent with *a* SCN. LMS process servicing the request in instance 2 will clone current mode buffer, verify that the SCN version is consistent with the request, and then send the CR copy of the block to the FG process.

The primary difference between these CR mode transfers and current mode transfers is that, in the case of CR mode transfers, no resource or locks are maintained in GRD for the CR buffers. Essentially, CR mode blocks do not require global cache resources or locks. Received CR copy is usable only by the requesting session and only for that specific SQL execution. That's why Oracle Database does not acquire any lock on that BL resource for CR transfers.

Since there are no global cache locks protecting the buffer, the next execution of SQL statement accessing that block connected to instance 1 would suffer from wait for `gc cr block 2-way` or `gc cr block 3-way` event too. So, every access to the block from instance 1 would trigger a new CR buffer fabrication. Even if nothing has changed in the buffer in instance 2, still, the FG process in instance 1 would suffer from CR wait events. CR buffers residing in instance 1 are not reusable either, since the requested query SCN will be different for every SQL execution.

The following trace line shows that a block was transferred from a resource master instance to the requesting instance with a latency of 0.6 ms. Further, file_id, block_id, and object_id information in the trace file can be used to identify the objects suffering from these two wait events. Of course, ASH data can be queried to identify the object also.

```
nam='gc cr block 2-way' ela= 627 p1=7 p2=6852 p3=1 obj#=76483 tim=37221074057
```

After executing tc_one_row.sql five times and then querying buffer cache headers, you can see that there are five CR buffers for that block in both instances 1 and 2. Notice that CR_SCN_BAS and CR_SCN_WRP columns[6] have different values for each CR buffer copy. Querying gv$ges_resource and gv$ges_enqueue, you can see that there are no GC locks protecting these buffers though.

Listing 10-10. Buffer Status

```
SELECT
  DECODE(state,0,'free',1,'xcur',2,'scur',3,'cr', 4,'read',5,'mrec',
               6,'irec',7,'write',8,'pi', 9,'memory',10,'mwrite',
              11,'donated', 12,'protected', 13,'securefile', 14,'siop',
              15,'recckpt', 16, 'flashfree', 17, 'flashcur', 18, 'flashna') state,
  mode_held, le_addr, dbarfil, dbablk, cr_scn_bas, cr_scn_wrp , class
```

[6]Essentially, queries request a specific version of the block by specifying query SCN as block version. The combination of cr_scn_bas and cr_scn_wrp specifies SCN version of a CR buffer.

```
FROM sys.x$bh
WHERE obj= &&obj
AND dbablk= &&block
AND state!=0 ;
Enter value for obj: 75742
Enter value for block: 180

STATE       MODE_HELD LE        DBARFIL     DBABLK CR_SCN_BAS CR_SCN_WRP      CLASS
---------- ---------- -- -------------- ---------- ---------- ---------- ----------
cr                  0 00              1      75742  649314930       3015          1
cr                  0 00              1      75742  648947873       3015          1
cr                  0 00              1      75742  648926281       3015          1
cr                  0 00              1      75742  648810300       3015          1
cr                  0 00              1      75742 1177328436       3013          1
```

CR buffer generation is a special case, and no global cache locks are acquired to protect CR buffers. It is possible to create a CR storm if there are long-pending transactions on highly accessed objects. Hence, it is prudent to schedule batch processes updating heavily accessed tables to a less busy timeframe.

GC CR Grant 2-Way/Gc Current Grant 2-Way

gc cr grant 2-way and *gc current grant 2-way* wait events are encountered if the requested block is not resident in any buffer cache. The FG process requests LMS process for a block, but the block is not resident in any buffer cache. So, LMS process replies with a grant message to the FG process to read the block from disk. The FG process reads the block from disk and continues the processing.

The following line shows that for the block access with file_id=4 and lock_id=180, a grant response was received by the FG process from LMS process. The subsequent line shows that a physical read was performed to read the block from disk.

```
nam='gc cr grant 2-way' ela= 402 p1=4 p2=180 p3=1 obj#=75742
nam='db file sequential read' ela= 553 file#=4 block#=180 blocks=1 obj#=75742
```

Excessive waits for these events imply either that the buffer cache is undersized or that the SQL statements are aggressively flushing the buffer cache. Identify SQL statements and objects suffering from waits for the events, and tune those SQL statements.

The DRM feature is designed to reduce the occurrences of waits for the grants.

GC CR Block Busy/GC Current Block Busy

Busy events indicate that LMS performed additional work due to concurrency-related issues. For example, to build a CR block, LMS process may have to apply undo records to reconstruct a block consistent with query SCN. While transferring the block back to the FG process, LMS will mark the block transfer to have suffered from `gc cr block busy` event or `gc current block busy` event depending upon the type of block transfer.

GC CR Block Congested/GC Current Block Congested

If the LMS process did not process a request within 1 ms after the receipt of a request, then LMS process marks the response that block suffered from congestion-related wait events. There are many reasons for congestion-related wait events, such as that the LMS process is overwhelmed by numerous global cache requests, LMS process is suffering from CPU Scheduling delays, LMS process has suffered from another resource starvation (such as memory), etc.

Typically, the LMS process runs in a real-time CPU scheduling priority; therefore, CPU scheduling delays will be minimal. The excessive number of waits for this event would indicate that there was a sudden spike in global cache requests and that the LMS process was not able to process the requests quickly enough. Memory starvation in the server also can lead to paging of the LMS process, affecting global cache performance.

You can review to see why LMS process is not able to process requests efficiently.

Placeholder Wait Events

After sending a message to LMS process to access a block, an FG process must wait for a response back from LMS process. RAC code is instrumented such that the FG process will wait on a placeholder event such as `gc cr request` or `gc current request`. The response from LMS process will indicate the type of delay encountered while processing the block. After receiving the response, the FG will account the time correctly to an actual wait event.

Generally, these placeholder wait events have no significance, since actual wait time is recorded to another wait event. If there is no response after a 0.5-s wait (6 s in Windows platform), the FG process will declare that the block has been lost, accounts time to `gc lost block`[7] wait event, and then resends the message. These placeholder events are visible only in the v$session/v$session_wait family of views and indicate the real-time session waits. As 500 ms is a long time compared to 2–3 ms, you would probe to identify why the reply is not received in 500 ms.

In a healthy database, the amount of time accounted toward these two placeholder wait events will be negligible. If you see many processes waiting for these placeholder wait events, then review other events such as `gcs log flush sync` or background waits to identify the performance delay.

Sending-Side Analysis[8]

Many analysts focus on the performance metrics of the receiving side exclusively while troubleshooting a global cache performance issue. RAC wait events suffered by the FG processes are instrumenting the performance of the *receiving side*.

It is equally important to understand the performance metrics of the *sending side* too. You need to understand that slowdown in the sending side can lead to longer wait times for FG processes in other nodes. Focus on sending side is more important if the histogram of wait events indicates latencies in higher-order buckets.

Fortunately, Oracle Database is instrumented with extensive statistics to perform a breakdown analysis on the sending side too. Before you can understand the metrics, you should understand the internal mechanics involved in a global cache block transfer. Figure 10-3 provides a rough overview of global cache processing. In this Figure, only two instances are shown to improve readability. The FG process connected to instance 1 tries to access a block, constructs a GC message, and sends the message to the LMS process running in instance 2. Assume that the block is resident in the buffer cache of instance 2 and that block is also mastered by instance 2.

[7]Event gc lost block is discussed extensively in Chapter 9. I think it is repetitive to probe this event further in this chapter.
[8]Sending side refers to the instance and LMS processes sending the block. Receiving side refers to the FG process requesting the blocks.

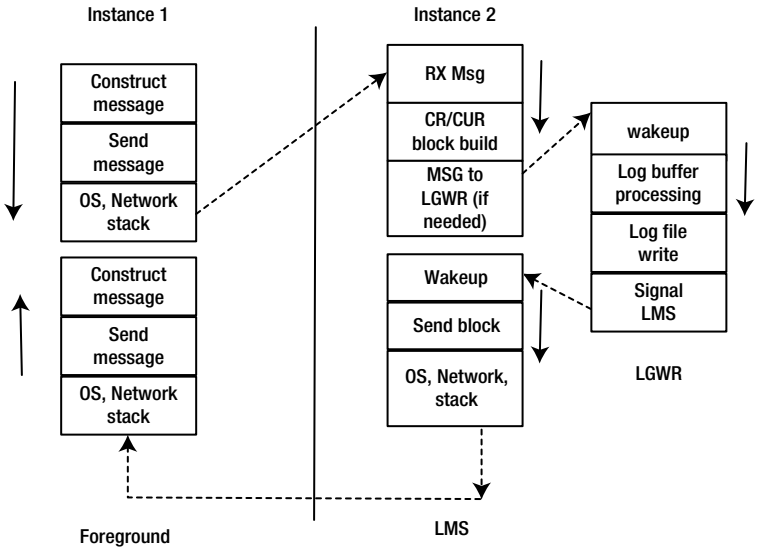


Figure 10-3. *LMS processing mechanics*

The FG process uses OS calls to send messages. These system calls in turn recursively call network layer kernel functions to send the message through the network hardware. These system calls are executed, accounting time to the kernel mode CPU, and higher global cache traffic can induce higher CPU usage in kernel mode too.

An LMS process in instance 2 receives the message. If the LMS process has generated any type of redo, to reconstruct the block either for CR or for current mode, LMS process will request LGWR process to do a log flush sync. While LGWR is performing a log flush, LMS process will account wait time to the `gcs log flush sync` event. After writing the log buffer to redo log files, LGWR wakes up LMS process, and LMS sends the block to the FG process in instance 2. The packet is delivered to the FG process through the interconnect.

The FG process reads the packets from network buffers, copies the block to buffer cache, and continues processing.

A block transfer involves numerous atomic steps. Time waited on each of these atomic steps is instrumented as a statistic. Latency to receive a block in CR mode in an instance can be written using the following formula.

gc cr block receive time= Time to send message to a remote LMS process by FG

+ Time taken by LMS to build block

(statistics: gc cr block build time)

+ LMS wait for LGWR latency

(statistics: gc cr block flush time)

+ LMS send time

(Statistics: gc cr block send time)

+ Wire latency.

The AWR report also captures these performance metrics and prints the average time taken for these atomic steps involved in a block transfer. The following section from AWR report shows an example. First, receive metrics are printed with an average of 1.9 ms and 1.8 ms for a block transfer in CR and current mode. The next few lines

print sending-side metrics suffered by LMS process. In this example, average CR block flush time is 8.3 ms and only 27% of the block transfers waited for LGWR to complete a log flush. The following AWR report shows no sign of a performance issue.

```
Global Cache and Enqueue Services - Workload Characteristics
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  Avg global cache cr block receive time (ms)              :      1.9
  Avg global cache current block receive time (ms)         :      1.8

    Avg global cache cr block build time (ms)              :      0.0
    Avg global cache cr block send time (ms)               :      0.0
    Global cache log flushes for Cr blocks served %        :     27.2
    Avg global cache cr block flush time (ms)              :      8.3
```

In contrast, the following AWR report was captured when there was a problem in the sending side. In this example, the average CR block receive time is 222 ms in instance 1. Since the receive time in instance 1 is higher, we need to review the sending-side metrics in other nodes; AWR report in instance 2 shows a glaring problem. Average global cache flush time is very high, at an average of 15.8 s per CR block transfer. From this AWR report, we can infer that the performance problems suffered by the LGWR process in instance 2 is affecting the receive time in other instances, notably in instance 1.

```
Instance 1:

  Avg global cache cr block receive time (ms)              :    222.1
  Avg global cache current block receive time (ms)         :     27.5

Instance 2:

    Avg global cache cr block build time (ms)              :      0.0
    Avg global cache cr block send time (ms)               :      0.1
    Global cache log flushes for cr blocks served %        :      2.7

    Avg global cache cr block flush time (ms)              :  15879.9
```

You may need to review performance metrics in instance 2. AWR reports from instance 2 confirm that the log writer was suffering from I/O-related latency, as the average latency for log file sync wait time is greater than 11 s. (Normally, this event has a latency of a few milliseconds). In a nutshell, LMS process was waiting for LGWR to complete log flush accounting wait time to `gc cr block flush time` statistics, and the FG process in instance 1 is waiting for the LMS process to respond.

```
Top 5 Timed FG Events
~~~~~~~~~~~~~~~~~~~~
                                                    Avg
                                                   wait   % DB
Event                                 Waits     Time(s)   (ms)   time Wait Class
------------------------------ ------------ ----------- ------ ------ ----------
log file sync                         2,054      23,720  11548   45.8 Commit
gc buffer busy acquire               19,505      10,382    532   20.0 Cluster
gc cr block busy                      5,407       4,655    861    9.0 Custer
enq: SQ - contention                    140       3,432  24514    6.6 Configurat
db file sequential read              38,062       1,305     34    2.5 User I/O
```

Similarly, sending-side metrics for current mode block transfer can be broken down to individual statistics too. The time taken by an LMS process to send a block to the remote side can be written as follows:

Gc current block receive time = Time take to send a message to LMS process by FG

+ (LMS) Time taken to pin block

(statistics: gc current block pin time)

+ (LMS) wait for log flush sync

(statistics: gc current block flush time)

+ (LMS) send time

(statistics: gc current block send time)

+ Wire latency.

Notice that the atomic steps involved in processing a current more transfer are slightly different from those involved in CR mode transfer. The statistic `gc current block pin time` indicates the time taken by the LMS process to pin a block. Higher latency for this statistic indicates that the blocks are currently pinned by another process and imply that block was busy.

Your strategy to debug cache fusion performance issue should cover performance metrics of both sending side and receiving side. Sending-side latency can lead to longer receiving-side latency.

Block Types Served

The pattern of block types served can be used to understand global cache workload patterns. `v$cr_block_server` provides a breakdown of CR requests into various types of requests. For example, the output in Listing 10-11 shows that 51% of CR requests were for data block, 31% of CR requests for undo blocks, and 15% for undo header block. From this output, you can infer that application affinity will greatly reduce the GC traffic, as 50% of the blocks served is for undo blocks.

Listing 10-11. Block Types Served

```
SELECT inst_id,
  TRUNC(data_requests /tot_req,2) * 100 data_per,
  TRUNC(undo_requests /tot_req,2) * 100 undo_per,
  TRUNC(tx_requests   /tot_req,2) * 100 tx_per,
  TRUNC(other_requests/tot_req,2) * 100 other_per
FROM
  (SELECT inst_id, cr_Requests + current_Requests tot_req,
    data_requests, undo_requests, tx_requests, other_requests
    FROM gv$cr_block_server
   )
ORDER BY inst_id;

   INST_ID   DATA_PER   UNDO_PER     TX_PER  OTHER_PER
---------- ---------- ---------- ---------- ----------
         1         51         31         15          0
         2         47         47          5          0
         3         49         42          7          0
```

GCS Log Flush Sync

When a session issues a `commit`, that action triggers LGWR to write log buffer entries to the redo log files. The FG process waits for LGWR to complete log file write. Until the LGWR process responds, the FG process will account wait time to `log file sync` event. The action of LGWR writes from the log buffer to the log file makes the transaction permanent. Even if the database crashes at that time, the transaction is not lost, as the transaction is permanently stored in the redo log file. Essentially, the durability in ACID properties of a database is satisfied with this mechanism (ACID = Atomicity, Consistency, Isolation, and Durability).

In RAC, since multiple buffer caches are fused, a mechanism similar to log file sync is employed by the LMS process to maintain durability. LMS will request a log flush before sending the block if undo records were applied to create a CR copy of the block or if the block has uncommitted recent transactions in the block. LMS process will wait for LGWR to complete the log flush and post LMS process to continue. LMS process accounts the wait time to `gcs log flush sync wait` event while waiting for LGWR process. The FG processes in other instances will be waiting for a global cache wait event while LMS process is waiting for LGWR process. Potentially, prolonged wait by LMS process explodes into massive waits for gc buffer busy waits.

Therefore, it is very important to have consistent LGWR performance in RAC. Any delay in LGWR processing will be magnified as an excessive amount of global cache waits.

If your database is suffering from high `gc buffer busy` waits or other RAC wait events, and if the average wait time for wait events is much higher, then probe to see if gcs log flush sync in *other* nodes is causing the problem, and also review the histogram of the wait time for gcs log flush sync wait event. Script event_histogram.sql in Listing 10-5 can be used to understand the waits by LMS process. The following output of the script shows that 91% of waits complete within 2 ms in a healthy database. (As always, use AWR report or custom script to review the histogram during a specific period of time.)

```
@event_histogram
Enter value for event_name: gcs log flush sync

   INST_ID EVENT                          WAIT_TIME_MILLI WAIT_COUNT        PER
---------- ------------------------------ --------------- ---------- ----------
         1 gcs log flush sync                           1  336241301      78.73
         1 gcs log flush sync                           2   58520524       13.7
         1 gcs log flush sync                           4   22517131       5.27
         1 gcs log flush sync                           8    7629771       1.78
         1 gcs log flush sync                          16    2103830        .49
         1 gcs log flush sync                          32      67629        .01
         1 gcs log flush sync                          64        608          0
         1 gcs log flush sync                         128         26          0
```

The following output shows a database with high wait time for gcs log flush event. This increase in wait time resulted in longer waits for the global cache waits from the perspective of application process.

```
@event_histogram.sql
Enter value for event_name: gcs log flush sync
INST_ID       EVENT                    WAIT_TIME_MILLI WAIT_COUNT   PER
--------- ---------------------------- --------------- ---------- -----
       1       gcs log flush sync                    1         28   .07
       1       gcs log flush sync                    2         24   .06
       1       gcs log flush sync                    4         31   .08
       1       gcs log flush sync                    8         33   .08
       1       gcs log flush sync                   16      35757 95.96
       1       gcs log flush sync                   32       1378  3.69
       1       gcs log flush sync                   64          6   .01
       1       gcs log flush sync                  128          2     0
```

If LMS database processes suffers from high gcs log flush sync wait event, then you would need to understand why LGWR is not responding fast enough. In most cases, that would require a review of the output of script lfsdiag.sql supplied by Oracle Support to understand the breakdown of the wait time suffered by LGWR process.

Defending LMS Process

As the performance of LMS process is quite essential, there are few features protecting LMS process overworking.

As I/O events usually exceed 5 ms, and since global cache latency is in the order of a few milliseconds, an I/O event initiated by LMS process can induce longer latencies to the FG processes. So, LMS process does not initiate any I/O calls at all. If LMS process has to initiate an I/O, for example, to read undo blocks required for CR fabrication as those undo block may not be in the cache anymore, then LMS process will downgrade the lock on the BL resource and send the buffer to the requestor. The requesting FG process will perform I/O calls and complete CR fabrication. This feature is known as the "Light Works Rule."

If a block is requested excessively, as detected by LMS process, then LMS process can downgrade the BL lock and transfer the block to the remote FG process. For example, if a block is requested in CR mode aggressively, then LMS process can downgrade the lock to a null mode and transfer the block to the FG process. This feature is known as "Fairness Down Convert."

An overworked LMS process can trigger delays in global cache transfer. These features protect LMS process from working too hard due to an onslaught of excessive global cache requests. View gv$cr_block_server provide statistics on these features.

GC Buffer Busy Acquire/Release

Wait events `gc buffer busy acquire` or `gc buffer busy release` are usually side effects of another root cause issue. In a single-instance database, if a session `s1` is trying to read a block from the disk to the buffer cache, and if another session s2 has issued a read call already for that block and has not completed the read call yet, then session s1 will wait for the read call issued by session s2 to complete. Session s1 will account wait time to an event, namely, `read by other session.` Additionally, subsequent sessions trying to read the same block, while the read call is still pending, will wait and account time to `read by other session` event. Further, if a buffer is pinned by a session `s2`, then another session `s1` trying to access the same buffer will wait for the `buffer busy wait` event.

In RAC, wait for `gc buffer busy` event is similar to single-instance counterparts `read by other session` and `buffer busy waits`. A wait for it indicates that there is a global cache operation pending on a buffer; subsequent processes must wait for that global cache operation to complete before accessing the buffer. For example, process p1 may have requested a BL lock on the block, but GCS lock has not been completely acquired. So, another process p2 trying to access that buffer should wait for `gc buffer busy` event, essentially yielding to process p1 to complete GC operation.

Rarely, gc buffer busy event itself is a root cause. Typically, it is a side effect of another problem. Here is an example—in the AWR report section for instance 1 as follows, 10,382s was spent waiting for `gc buffer busy acquire` during an AWR period of 30 min, and the root cause, LGWR, was not able to complete log flushes quick enough in instance 2. So, the LMS process in instance 2 was stuck waiting for LGWR to complete a log flush. Meanwhile, a few processes requesting blocks in instance 1 are waiting on `gc cr request` or `gc current request` waits. Subsequent processes accessing the same block are waiting for `gc buffer busy acquire` event, since a global cache request is pending and the request was initiated from the local node.

```
Top 5 Timed Foreground Events
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                                                    Avg
                                                   wait   % DB
Event                             Waits     Time(s)   (ms)   time Wait Class
------------------------------ ------------ ----------- ------ ------ ----------
log file sync                      2,054      23,720  11548   45.8 Commit
gc buffer busy acquire            19,505      10,382    532   20.0 Cluster
gc cr block busy                   5,407       4,655    861    9.0 Cluster
enq: SQ - contention                 140       3,432  24514    6.6 Configurat
db file sequential read           38,062       1,305     34    2.5 User I/O
```

To troubleshoot gc buffer busy wait events, initially review events suffered by the background processes. In many production problems, slowness in the database background processes is the root cause of gc buffer busy event waits. It is also important to understand the difference between `gc buffer busy acquire` and `gc buffer busy release` wait events. The following output from v$session_wait view differentiates these two events nicely:

1. Notice that session 53 is waiting to access the block with `file_id=10` and `block_id=560651` and that session is currently waiting for `gc current request` event (a placeholder wait event as discussed).
2. Session 47 connected to instance 2 is waiting for `log file sync` event. LMS process is also waiting for `log file sync` wait event (not shown in the output, though).
3. In instance 1, all other sessions trying to access that block with `file_id=10` and `block_id=560651` are waiting for `gc buffer busy acquire` wait event.
4. Meanwhile, sessions connected to instance 2, which are trying to access that block, are waiting for `gc buffer busy release` wait event.

```
INST_ID    SID EVENT                          STATE   WIS   P1_P2_P3_TEXT
------- ------ -------------------------- ------- ----- ------------------------------
      1     53 gc current request           WAITING    26 file# 10-block# 560651-...
      1     40 gc buffer busy acquire       WAITING    26 file# 10-block# 560651-class# 1
      1     60 gc buffer busy acquire       WAITING    26 file# 10-block# 560651-class# 1
      1     59 gc buffer busy acquire       WAITING    26 file# 10-block# 560651-class# 1
      1     58 gc buffer busy acquire       WAITING    26 file# 10-block# 560651-class# 1
...
      2      1 gc buffer busy release       WAITING     1 file# 10-block# 560651-class# 1
      2     68 gc buffer busy release       WAITING     1 file# 10-block# 560651-class# 1
      2     65 gc buffer busy release       WAITING     0 file# 10-block# 560651-class# 1
...
      2     47 log file sync                WAITING    22 buffer# 4450-sync scn 30839721-
      2     43 gc buffer busy release       WAITING     1 file# 10-block# 560651-class# 1
      2     36 gc buffer busy release       WAITING     0 file# 10-block# 560651-class# 1
...
```

Essentially, sessions connected to instance 1 are waiting for the global cache locks to be acquired by instance 1, and so they account wait time to event `gc buffer busy acquire`. Sessions connected to instance 2 are waiting for instance 1 to release the gc lock, and so time is accounted to `gc buffer busy release` event. As global cache BL locks are considered to be owned by an instance, not by an individual session, in this example, sessions connected to instance 1 are waiting for instance 1 to acquire the gc lock, and sessions connected to instance 2 are waiting for instance 1 to release the gc lock.

This differentiation is useful to identify the instance that initiated a block request. Next, we need to identify the object suffering from an excessive amount of `gc buffer busy` events. SQL in Listing 10-3 is handy to identify the objects inducing this wait event. The following output shows that index `lines_u1` is suffering from an excessive number of `gc buffer busy acquire` wait events. Depending upon the type of object, a different troubleshooting method will be employed. Again, if the problem is constrained to a specific time interval, then you should review the AWR report for that time frame and focus on the section listing segments suffering from gc buffer busy waits.

```
@ash_gcwait_to_obj.sql
Enter value for event: gc buffer busy acquire
Enter value for threshold: 1000
/
      INST OWNER      OBJECT_NAME                    OBJECT_TYP        CNT
---------- ---------- ------------------------------ ---------- ----------
         1 RS         LINES_U1                       INDEX         1205710
         2 AR         RA_CUSTOMER_TRX_ALL            TABLE            4399
         1 AR         RA_CUSTOMER_TRX_ALL            TABLE            4241
         1 WSH        WSH_DELIVERY_DETAILS_IDX1      INDEX            4106
...
         1                                           Undo Header      1481
                                                     /Undo block
```

The following section discusses the common object types causing the gc buffer busy wait event and a method to diagnose further.

Unique Indexes

In a B-tree index, indexed column values are stored in (key[s], rowid) format in ascending or descending key column order. If the column values are populated using sequence-generated values, then all new rows will be populated in the rightmost leaf block of the index. This type of index growth is **right-hand index growth**. If many sessions are trying to insert rows into the table, then the rightmost leaf block will be a hot block. In the case of a single-instance database, this concurrency issue will cause waits for read by other session and buffer busy wait events. In a RAC database, this problem will be magnified as gc buffer busy waits. If the application inserts into that table from all instances, then the rightmost leaf block has to be transferred between the instances aggressively, similar to a Hot Potato game played by children in the United States.

Almost-unique indexes can cause `gc buffer busy` waits too; indexes on timestamp column are an example. Indexed column values inserted concurrently will have the same timestamp or timestamp values close enough that those values will be stored in fewer leaf blocks of the index B-tree. So, concurrent inserts into that table will induce contention for few leaf blocks.

You will also notice that contention will be moving from one block to another block, as the leaf block will be filled, and new blocks added to the index structure of their rightmost part of the index.

Right-hand growth index contention can be resolved by hash partitioning the index on key column values. By hash partitioning the index, values are distributed among multiple leaf blocks, thereby effectively spreading the activity to many blocks. From 10g onward, indexes can be partitioned even if the table is not partitioned.

`Index unique scan` and `index range scan` access paths on hash-partitioned indexes specifying, at least, all partitioning key column values will perform similar to a nonpartitioned index. However, if your application performs numerous index range scans without providing all partitioning key columns, then hash partitioning will increase logical I/O per execution.

Reverse key indexes also can be used to combat right-hand growth contention. But reverse key indexes do not allow a range scan access path, and so there are only a very few reasons as to why you would choose a reverse key index over hash-partitioning indexes. One of those reasons is that partitioning requires a license for partitioning option.

Table Blocks

If the object_type suffering from gc buffer busy wait event is a table block, then it is imperative that you identify the type of block inducing gc buffer busy wait event. Listing 10-6 provides a SQL statement querying gv$active_session_history data and identifies the blocks inducing gc buffer busy acquire waits.

After identifying the specific block, you must identify the type of the block suffering from contention. The output in Listing 10-12 shows that block with file_id=49 and block_id= 635413 belongs to the fnd_concurrent_processes table (a familiar E-Business suite table). You may have to dump the data block to identify the block type.

Listing 10-12. Ash_gcwait_to_block.sql

```
col inst format 9999
col current_file# format 99999  head file
col current_block# format 9999999 head blk
WITH ash_gc AS
  (SELECT * FROM
    (SELECT /*+ materialize */ inst_id, event, current_obj#, current_file#,
      current_block#, COUNT(*) cnt
    FROM gv$active_session_history
    WHERE event=lower('&event')
    GROUP BY inst_id, event, current_obj#,
      current_file#, current_block#
    HAVING COUNT(*) >5
    )
  WHERE rownum <101
  )
SELECT * FROM
  (SELECT inst_id, owner, object_name, object_type, current_file#,
    current_block#, cnt
  FROM ash_gc a, dba_objects o
  WHERE (a.current_obj#  =o.object_id (+))
  AND a.current_obj#  >=1
  UNION
  SELECT inst_id, '', '', 'Undo Header/Undo block' ,
    current_file#, current_block#, cnt
  FROM ash_gc a
  WHERE a.current_obj#=0
  UNION
  SELECT inst_id, '', '', 'Undo Block' ,
    current_file#, current_block#, cnt
  FROM ash_gc a
  WHERE a.current_obj#=-1
  )
ORDER BY 7 DESC
/
Enter value for event: gc buffer busy acquire
```

```
     INST OWNER      OBJECT_NAME                    OBJECT_TYP   file     blk     CNT
--------- ---------- ------------------------------ ---------- ------ ------- -------
        1 APPLSYS    FND_CONCURRENT_PROCESSES       TABLE          49  635413    2039
        1 APPLSYS    FND_CONCURRENT_PROCESSES       TABLE         301  954692      18
        1 SYSTEM     AQ$_QUEUE_TABLES               TABLE           4    1129      18
        1 APPLSYS    FND_CONCURRENT_QUEUES          TABLE         382  123951      17
        1                                           Undo Block      0       0      15
...
```

Dumping the data block using the following command[9] and reading through trace file, you can identify the type of block. Oracle Support will help to read the trace file too. Table 10-1 shows possible block types and methods to resolve the contention.

```
alter system dump datafile 49 block 635413 block max 49 635413   ;
```

Table 10-1. *Block Type and Possible Issues*

Block type	Comment
Segment Header	Probable freelists contention. Alter the table to have multiple freelist groups.
Table block	Possibly code updating a few rows of the table aggressively. Concurrent deletes on inserts of a few rows. SQL statements scanning objects aggressively. Partitioning or code change may be required.
ASSM blocks	ASSM bitmap block contention. Usually would require further analysis from Oracle Support.
Index root or branch blocks	Heavy concurrent inserts and deletes.
Undo header blocks	Numerous short transactions?
Undo blocks	Long-pending transactions and excessive CR generation.
Dictionary	Possible sequence issues if the block belongs to SEQ$ table. Requires further analysis.

If your database is a 12c CDB, then join to cdb_objects to identify PDB associated with the object. The following script is useful in a database container to identify the block in a CDB.

Listing 10-13. Ash_gcwait_to_block_12c.sql

```
col inst format 9999
col current_file# format 99999  head file
col current_block# format 9999999 head blk
WITH ash_gc AS
  (SELECT * FROM
    (SELECT /*+ materialize */ inst_id, con_id, event, current_obj#, current_file#,
      current_block#, COUNT(*) cnt
```

[9]Another cautionary tale: do not dump numerous blocks in a RAC database from version 11.2 onward. For example, dumping all blocks of an extent overloads the LMS processes and can lead to node crash. So, try to dump few blocks at a time if required.

```
    FROM gv$active_session_history
    WHERE event=lower('&event')
    GROUP BY inst_id, event, current_obj#,
      current_file#, current_block#, con_id
    HAVING COUNT(*) >5
    )
  WHERE rownum <101
  )
SELECT * FROM
  (SELECT a.inst_id, a.con_id, owner, object_name, object_type, current_file#,
    current_block#, cnt
  FROM ash_gc a, cdb_objects o
  WHERE (a.current_obj#  =o.object_id (+) and a.con_id=o.con_id(+) )
  AND a.current_obj#  >=1
  UNION
  SELECT inst_id, a.con_id, '', '', 'Undo Header/Undo block' ,
    current_file#, current_block#, cnt
  FROM ash_gc a
  WHERE a.current_obj#=0
  UNION
  SELECT inst_id,a.con_id, '', '', 'Undo Block' ,
    current_file#, current_block#, cnt
  FROM ash_gc a
  WHERE a.current_obj#=-1
  )
ORDER BY 7 DESC
/
```

In summary, event waits for gc buffer busy acquire/release is a symptom of another problem, and you need to investigate the layers underneath to identify the root cause.

DRM

As discussed earlier in this chapter, the mastership of gc resources is distributed among all active instances. If a resource is locally mastered, then the lock on that resource can be acquired using the affinity-locking scheme. With the affinity-locking scheme, the FG process can query GRD in the local instance, identify current locks, and acquire locks typically without any involvement of the background processes. This affinity-locking scheme improves the performance of the application, as the waits for grants are eliminated or reduced.

What if an object obj1 is accessed heavily by an instance? In that scenario, wouldn't it be better to remaster that object to that instance so that every block of that object obj1 is locally mastered? With locally mastered locks, the application can perform better, since waits for grants will be minimal. That's the idea behind the DRM feature. For example, a tkprof output file of a batch program accessing few tables aggressively is shown as follows. Over 1,050 s was spent waiting for grants. These waits for grants can be avoided if those heavily accessed objects are locally remastered.

Elapsed times include waiting on the following events:

```
Event waited on                          Times   Max. Wait   Total       Waited
---------------------------------------  Waited        ----------  -------------
gc cr grant 2-way                        271931              1.39         905.49
...
gc cr grant congested                      4249              1.47         150.58
...
gc current grant 2-way                       98              0.00           0.22
```

Overview of DRM Processing

RAC background processes keep track of global cache lock requests at an object level. I should emphasize that it is not object access but rather gc lock requests on an object that are tracked. If the number of BL lock requests from an instance exceeds a threshold in an observation window, then DRM is triggered for that object. After the completion of DRM, the object will be locally mastered and future access to that object from that instance will use affinity-locking scheme, reducing waits for grants. Three RAC background processes are at the heart of DRM implementation:

1. LCK0 process maintains object-level statistics and decides if remastering should be triggered on an object.
2. If an object is chosen, a request is queued. LMD process reads the request queue and initiates GRD freeze. In the following LMD trace file, pkey indicates the object_id of the remastered object. Dissolve pkey means that all resources currently mastered by the local instance must be removed from local instance memory.

   ```
   *** 2010-01-08 19:41:26.726
   * kjdrchkdrm: found an RM request in the request queue
   Dissolve pkey 6984390
   *** 2010-01-08 19:41:26.727
   Begin DRM(189) - dissolve pkey 6984390 from 2 oscan 1.1
   ftd received from node 1 (8/0.30.0)
   ftd received from node 0 (8/0.30.0)
   ftd received from node 3 (8/0.30.0)
   all ftds received
   ```

3. LMON coordinates reconfiguration using LMS processes. Reconfiguration goes through many stages. DRM reconfiguration is similar to the reconfiguration triggered after the instance crash, but less severe. Reconfiguration triggered after an instance crash will remaster every resource mastered by the failed instance, whereas reconfiguration triggered for DRM will remaster only the resource identified by LCK0 process.

   ```
   *** 2010-01-08 19:41:26.793
   Begin DRM(189)
   sent syncr inc 8 lvl 5577 to 0 (8,0/31/0)
   synca inc 8 lvl 5577 rcvd (8.0)
   ```

View x$object_policy_statistics externalizes the BL lock requests and shows locking activity at the object level. As visible in the query output from x$object_policy_statistics, there were many Shared BL lock requests on object 12189604. In the following output, columns SOPENS and XOPENS track counts of BL lock requests in shared and exclusive mode, respectively.

```
SYS@ORCL1:> SELECT node, object, sopens, xopens, xfers, dirty
    FROM x$object_policy_statistics
    ORDER BY sopens+xopens
/
      NODE     OBJECT     SOPENS     XOPENS      XFERS      DIRTY
---------- ---------- ---------- ---------- ---------- ----------
...
         1   12189628     203303          1     127563         43
         3   11877357     247303        249      53641         16
         1    1698508     332013          0          0          0
         1    9972722     367830          0        181          0
         1   12189604     383890          2      22008          0
```

View gv$policy_history shows details about DRM activity. This view provides the object that was involved in a DRM activity, target instance that object was remastered to, and the date of the remastering event. In the following output, object_id 76739 was remastered on 12/30/2011 to instance 1. Initiate affinity is the policy event for remastering activity.

```
RS@ORCL1> select * from gv$policy_history;

INST_ID    POLICY_EVENT         DATA_OBJECT_ID TARGET_INSTANCE_NUMBER EVENT_DATE
---------- -------------------- -------------- ---------------------- --------------------
         1 initiate_affinity             76739                      1  12/30/2011 11:43:06
```

View gv$gcspfmaster_info shows current mastership information about the remastered object. Underlying x$table is used by the Oracle code to check if an object is mastered or not. For every BL lock request, RDBMS code checks if the object is locally mastered; if the object is visible in the underlying x$ table of v$gcspfmaster_info, then affinity locking code is executed.

```
RS@ORCL1>select * from v$gcspfmaster_info where object_id=6984154;

FILE_ID    OBJECT_ID  CURRENT_MASTER PREVIOUS_MASTER REMASTER_CNT
---------- ---------- -------------- --------------- ------------
         0    6984154              1               0            2
```

In release 12c, the con_id column is populated with a value of 0 in x$object_policy_statistics, gv$policy_history, and gv$gcspfmaster_info views, indicating that these rows are associated with the whole CDB. Hopefully, this is just a bug to be fixed in future patches. Using con_id, you can identify PDB associated with the object.

I triggered object remastering by accessing the object in a loop and initiating numerous BL lock requests. (Again, note that it is the number of BL lock requests that is used to trigger DRM activity, not the number of accesses to a block, two completely different metrics.) After remastering the object again, you can see that current_master column value changed from 1 to 2 and remaster_cnt increased to 3.

```
RS@ORCL1>select * from v$gcspfmaster_info where object_id=6984154;

FILE_ID    OBJECT_ID  CURRENT_MASTER PREVIOUS_MASTER REMASTER_CNT
---------- ---------- -------------- --------------- ------------
         0    6984154              2               1            3
```

DRM Stages

LMON goes through various stages to complete DRM activity. One major phase of DRM activity is GRD freeze. While GRD is frozen, no BL locks can be acquired or released. Existing buffers in SGA can be accessed, but new global locking requests will be stuck waiting for gc remaster event. At the end of DRM activity, GRD is unfrozen and the global cache locking activity continues. This GRD freeze and unfreeze is performed in a synchronized fashion by all instances. Instance triggering the remastering event acts as a co-coordinator instance. It coordinates with other instances to complete a specific sequence of tasks to complete the remastering event.

The following output from the LMON trace file shows an overview of DRM activity in a database. As a first step, GRD is frozen and no new BL locks can be acquired after the completion of freeze. Then, old locks and resources of the remastered objects are cleaned out. This cleanup activity must happen in all instances so that there is a synchronization mechanism after each step. The remastering instance waits for other instances to complete the task before proceeding to the next step. Finally, locks and resources are transferred to the instance that is about to receive the remastered objects.

```
*** 2011-08-12 15:24:33.146
* drm freeze : Quiesce- No new opens or converts
* drm cleanup: All old locks and resources closed for the objects
* drm sync 2: wait for sync from all nodes
* drm replay: transfer resource/lock states to new nodes
* drm sync 3: wait for sync
* drm fix writes
```

You have probably guessed the pain point of DRM processing. GRD freeze means that no new BL resources or locks can be acquired during the initial phases of DRM. This GRD freeze can induce a complete application hang.

GRD Freeze

GRD is frozen briefly during a resource remastering event. No new global resources and locks can be acquired while GRD is frozen, and in a few scenarios, this freeze can lead to a hung application. Worse yet, DRM can happen in cycles; for example, DRM started for object 1, at the completion of DRM, DRM started for object 2, etc. In a busy database, even a minute of GRD freeze can throw instability into application availability and performance. Further, as the size of SGA grows, the severity of GRD freeze increases. Typically, business-critical applications tend to have bigger SGA, and so the effect of GRD freeze can be severe in those business-critical applications. Databases with less activity might not suffer excessively due to GRD freeze.

The following output shows a few lines from an AWR report during DRM-related freeze. Event `gcs drm freeze enter server` indicates that the problem is due to DRM activity. The first process requesting a block will wait for `gc remaster` or `gcs drm freeze enter server` event. Subsequent processes accessing the same block will wait for `gc buffer busy` event, since there is an outstanding BL lock request already. So, in the following example, the actual impact of the application due to DRM activity should be considered 70% of response time. Note that average CR receive time is also at an elevated level of 17.1 ms and that elevated latency is due to DRM freeze.

```
Top Five Timed Events                                             Avg     %Total
~~~~~~~~~~~~~~~~~~~~                                              wait    Call
Event                            Waits           Time (s)         (ms)    Time    Wait Class
------------------------------  ------------    -----------       ------  ------  ----------
gc buffer busy                     1,826,073        152,415           83    62.0  Cluster
CPU time                                             30,192         12.3
enq: TX - index contention             34,332         15,535          453     6.3  Concurrency
gcs drm freeze in enter server         22,789         11,279          495     4.6  Other
enq: TX - row lock contention          46,926          4,493           96     1.8  Applicatio
```

```
Global Cache and Enqueue Services - Workload Characteristics
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Avg global enqueue get time (ms)                : 16.8
Avg global cache cr block receive time (ms)     : 17.1
Avg global cache current block receive time (ms): 14.9
```

Parameters

There are a few underscore parameters controlling the behavior of DRM activity. Parameter _gc_policy_minimum decides the minimum gc activity per minute of an object to be considered for resource mastering (default value, 1,500). Parameter _gc_policy_time governs the duration of an observation window (default value, 10 minutes). During the observation window, objects are observed, objects exceeding a minimum threshold are considered for resource mastering, and DRM is triggered.

There are many other underscore parameters controlling the behavior, and those parameters usually start with _lm_drm_ and do not normally need any adjustments. However, if you find that DRM activity is slower, it is possible to tune DRM speed using these parameters, but you must contact Oracle Support before changing these initialization parameters.

If your database performance suffers from excessive amount of DRM activity, you can tune that by adjusting `_gc_policy_minimum` parameter to a much higher value, such as 1 million. While DRM can be disabled by adjusting _gc_policy_time to zero, this choice disallows manual remastering too. So, the preferred action is to increase `_gc_policy_minimum` to a bigger value.

Changes in 12c

DRM has slight changes in release 12c, mostly for the PDB feature. Every PDB generates its own object_ids, and so pkey format has been changed to accommodate plugged-in databases. For example, remastering of an object with object_id=89911 uses pkey including con_id of plugged-in database. A snippet from LMD trace file follows.

```
*** 2013-03-17 21:57:49.718
Rcvd DRM(3) AFFINITY Transfer pkey 4.4.89911 to 1 oscan 1.1
ftd (30) received from node 1 (4 0.0/0.0)
all ftds received
```

In this example, pkey is 4.4.89911, where the first 4 is con_id of the object, my guess is that the second 4 is also a container_id, and 89911 is the object_id of the object. You can query v$container or cdb_pdbs to identify the plugged-in database of this object.

DRM and Undo

Undo segments also can be remastered. In RAC, every instance has its own undo tablespace. For example, if undo_tablespace parameter is set to undots1, then all undo segments onlined in instance 1 will be allocated in undots1 tablespace. So, by default, an instance onlining an undo segment will also be the resource master for that segment. But, if another instance accesses an undo segment excessively, then resource mastership for an undo segment can be remastered to another instance.

LMD0 trace file will show an object_id exceeding 4 billion. Here is an example of an undo remastering event:

```
Begin DRM(104) - transfer pkey 4294951403 to 1 oscan 0.0*** 2011-08-19
```

In version 12c, pkey format has been changed and pkey consists of two parts: value of 4 billion coloring the pkey as an undo segment, and 65541 identifying a specific undo segment.

```
*** 2013-03-17 19:29:41.008
Begin DRM(2) (swin 1) - AFFINITY transfer pkey 4294967295.65541 to 2 oscan 1.1
kjiobjscn 1
ftd (30) received from node 1 (4 0.0/0.0)
all ftds received
```

Troubleshooting DRM

If your database is suffering from excessive issues due to GRD freeze (and AWR wait events supports that assertion), then use the following guidelines to debug the problem.

1. Identify objects involved in DRM activity. Views gv$gcspfmaster_info and gv$policy_history can be used to identify objects triggered for DRM activity. LMD trace file will show object_id involved in a DRM with pkey as object_id. Verify if these object_ids are valid object_ids. In the following example, DRM is triggered on an object 27707121. Verify this object_id with dba_objects/cdb_objects(12c) and check if the object is valid. In 12c CDB, you need to identify the PDB associated with the object.

   ```
   Rcvd DRM(5) AFFINITY Transfer pkey 27707121.0 to 3 oscan 1.1
   ```

2. Understand if the activity on that object is justified by reviewing application workload pattern. If the activity is justified, then try to tune the SQL statement accessing those objects to reduce object access.
3. Identify the programs/application processes accessing those objects heavily. Colocate those programs to the same instance.
4. If none of these actions help, then contact Oracle Support and check if you can reduce DRM activity by tuning _gc_policy_minimum and _gc_policy_time parameters.

In summary, dynamic remastering is a good feature to improve performance of applications in a RAC database if the application affinity is in play. You can reduce the RAC tax to a minimal amount with this great feature. Also, in a RAC one-node database or a RAC cluster with active/passive instances, you can avoid RAC tax completely. It is a pity that this feature doesn't work for all workloads and especially that this feature starts to be problematic if the SGA size exceeds 50GB+.

AWR Reports and ADDM

AWR reports[10] and Automatic Database Diagnostic Monitor (ADDM) are the first step in performance diagnosis in a RAC cluster. There are multiple ways you can generate AWR reports in a RAC environment. You can also use OEM to generate an AWR report.

To generate an AWR report in a specific instance, use the awrrpti script. You can supply instance and database to generate AWR reports. In the following example, I am generating AWR report for instance 3.

[10]You need a Diagnostics and Tuning pack license to access AWR tables or report.

```
@$ORACLE_HOME/rdbms/admin/awrrpti
...
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   DB Id     Inst Num DB Name      Instance     Host
------------ -------- ------------ ------------ ------------
* 3821847422        2 PROD         PROD2        db1
  3821847422        3 PROD         PROD3        db2
  3821847422        1 PROD         PROD1        db3

Enter value for dbid: 3821847422
Using 3821847422 for database Id
Enter value for inst_num: 3
Using 3 for instance number
```

To generate an AWR report for the current instance, you can call awrrpt.sql file.

```
@$ORACLE_HOME/rdbms/admin/awrrpt.sql
```

To generate an AWR report for all instances, you can execute awrrptg.sql script. This script pulls the data for all active instances, prints metrics at both instance levels, and groups at cluster level. The AWR report generated from this script is useful to compare the performance of instances quickly.

```
@$ORACLE_HOME/rdbms/admin/awrrptg.sql
...
```

ADDM is executed after AWR report capture automatically. You can review the recommendations from ADDM with ease using the OEM tool. You can also use SQL*Plus tool to retrieve last task_name and retrieve the ADDM report.

```
SQL> select  task_name, status from dba_advisor_tasks where task_id=
  (select max(task_id) from dba_advisor_tasks where status='COMPLETED' );

ADDM:4227381283_3_59788        COMPLETED


SQL> SET LONG 1000000 PAGESIZE 0;
SQL> SELECT DBMS_ADDM.GET_REPORT('ADDM:4227381283_3_59788') from dual;
```

ASH Reports

Similar to AWR reports, you can also generate ASH reports for the local instance or a specific instance. To generate an ASH report for a specific instance, use ashrpti.sql script.

```
@$ORACLE_HOME/rdbms/admin/ashrpti.sql
```

To generate an ASH report for a current instance, use ashrpt.sql.

```
@$ORACLE_HOME/rdbms/admin/ashrpt.sql
```

Summary

The methodology to optimize an application in a RAC database is not vastly different from the single-instance methodology. Understand which layer is causing performance issues and tune that layer to resolve the root cause of the performance issue.

In most cases, RAC performance problems are due to single-instance database problems hiding behind RAC wait events.

This chapter describes the internals of cache fusion processing in an easy-to-understand format. You should be able to understand the meaning of the RAC wait events, identify objects inducing those RAC wait events, and identify SQL statements suffering from or inducing RAC wait events after having read this chapter. Further, you should be able to understand the details about LMS processing for a block transfer. Placeholder wait events indicate the current state of the session, but you should identify the final wait event to which the wait time was accounted.

DRM is an excellent feature. It is very useful in environments where the application affinity is implemented. It is a pity that resource mastering freezes GRD completely, and hopefully in a future release, this freeze will be minimized to freeze just a small portion of GRD.

Of course, a review of AWR reports and ASH reports of the problem time frame is an essential step in understanding the root cause of a performance issue.

CHAPTER 11

Locks and Deadlocks

by Riyaj Shamsudeen

Oracle database uses a locking scheme as a coordination mechanism to protect critical resources from concurrent changes. In a single-instance database, instance-level locks are sufficient as the locks need to be visible in the local instance only, so local locks are sufficient. In a RAC database, resources must be globally visible and so a new layer—Global Enqueue Services (GES)—is employed to implement a global locking scheme.

Global Resource Directory (GRD) is a memory area designed to keep track of the state of resources in a RAC database. GRD is distributed across all active instances of a database. This distribution removes a single point of failure and resource contention.

Resources and Locks

In this context, a resource is a structure designed to protect an entity from concurrent changes and act as a coordination mechanism for concurrency control. Locks are acquired on that resource before altering the entity. For example, a database table is a lockable entity, and before altering a table, a *resource* is allocated uniquely representing that table. Then, a *lock* is acquired on that resource in a specific mode. Another session modifying the same table must acquire a lock on the resource before altering the table. If the requested lock mode is incompatible with mode already held, then the requesting session must wait for the locks to be available. With this locking scheme, concurrent changes are coordinated to avoid consistency issues.

In a single-instance database, entities are modified only by a single instance, so single-instance resources and locks are sufficient to implement a locking scheme. In a RAC database, an entity can be modified by any instance of a database and hence changes must be globally coordinated. Globally maintained structures are required, as local resources and locks are not sufficient. For example, a session in instance 1 locks a row and another session in instance 2 tries to modify the same row concurrently while the transaction initiated by the session 1 is still active. A global locking scheme, GES layer, is employed to protect critical resources from concurrent changes globally. As in a single-instance locking scheme, every global resource has an associated resource structure: locking a global resource means that a global locking structure is associated with a global resource.

Figure 11-1 shows the structures employed in this global locking scheme. In this figure, a database table is to be locked. A resource uniquely representing that table is allocated to protect it. The resource uniquely represents the table by coding object_id of the table in the resource_name. Further, a resource structure has two queues: converting queue and granted queue. If a process succeeds in acquiring a lock on a resource, then that process is added to the granted queue of the resource. If the resource is not available or if the requested lock mode is incompatible with lock mode held already, then the requesting process is added to the converting queue. In Figure 11-1, one process is holding a lock on the resource, and two processes are waiting for the resource to be available.

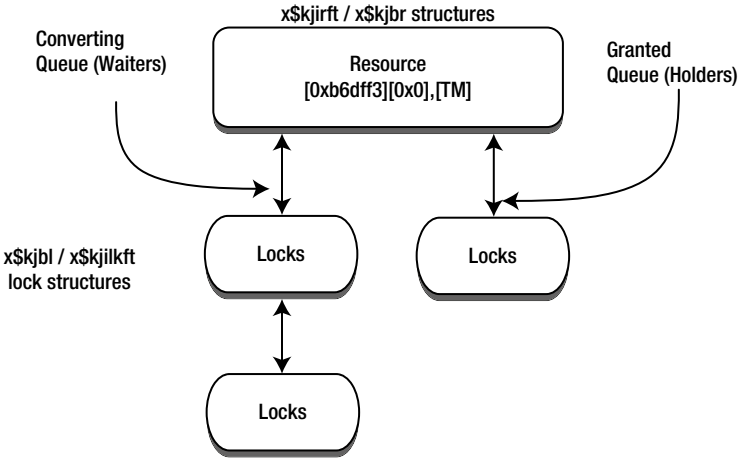


Figure 11-1. *Resources and locks in GRD*

■ **Note** Figure 11-1 shows locks and resources for both GES structures and Global Cache Services (GCS) structures. Internally, GCS manages cache fusion-related resources. GCS resources are externalized in x$kjbr fixed table, GCS locks are externalized in x$kjbl fixed table, GES resources are externalized in x$kjirft, and GES locks are externalized in x$kjilkft table. There are a few more structures implementing resources and locks too.

GRD is distributed among active instances of a database. For example, if there are three active instances, then one-third of GRD is resident in each instance. As resources and locks are integral components of GRD, resources are also distributed among active instances, and an instance is assigned to be the resource master; essentially, a resource master keeps track of the state of the resource at any given time. As searching for a resource in all instances will be cost-prohibitive, by employing the resource mastering scheme, search for a resource is optimized. Essentially, locking a resource has three steps: identify the resource master instance; request for locks on that resource to the resource master instance; and then the resource master instance grants lock or adds the request to the converting queue. The following is an elaboration of these steps:

1. Identify the master instance of a resource using the resource name. Typically, a variant of hash function is applied on the resource name to identify the master instance.
2. Send a message to resource master instance to acquire a lock on the resource. If the resource is available, then the remote instance will allocate a locking structure and associate the locking structure to the *granted* queue of that resource. A message is sent back to the requesting process with a grant. This action implies that requested process now holds a lock on that resource.
3. If the resource is not available, then the resource master instance will allocate a locking structure and associate the locking structure to the converting queue. A process in the *converting* queue waits for the resource to be available.

Resources and locks are allocated in GRD and maintained by GES layer processes. These components are the key pieces for the global locking scheme.[1]

SGA Memory Allocation

GRD is stored in the shared_pool area of instance SGA. The memory allocations for GES resources and locks are tagged as `ges resources` and `ges enqueues`. As the size of SGA and activity grows, these memory areas can grow dynamically too. In the following output, 65MB is allocated for `ges resource` and 150MB for `ges enqueues`. The size of GCS layer resources and locks is proportional to the size of SGA.

```
SELECT name, TRUNC(bytes/1024/1024, 2) size_MB
FROM v$sgastat
WHERE name LIKE 'ges resource%'
OR name LIKE 'ges enqueues'
OR name LIKE 'gcs resources'
OR name LIKE 'gcs enqueues'
OR name LIKE 'gcs shadows';

NAME                         SIZE_MB
-------------------------- ----------
ges resource                   65.77
ges enqueues                  150.37
gcs resources                 599.44
gcs shadows                   441.69
```

In version 12c, ges resource memory allocations are fine-grained into permanent and dynamic resource allocations. The following shows the output of the earlier query.

```
NAME                         SIZE_MB
-------------------------- ----------
ges resource dynamic            10.81
ges resource permanent          45.87
...
```

You can gather statistics about dynamic growth or shrinkage by reviewing v$resource_limit view. In the following output, resources grew up to a count of 689,253. However, current utilization is 123,090. This dynamic growth or shrinkage is not necessarily a concern unless SGA memory allocations for resources and enqueues are growing rapidly.

```
SELECT * FROM v$resource_limit WHERE resource_name IN
('ges_ress','ges_locks', 'gcs_resources','gcs_shadows');
```

[1]In version 12c, the pluggable database feature is introduced. GES and GCS services are common services for both pluggable databases and container databases, meaning both memory and background processes are shared between container and plugged databases. So, structures discussed in this chapter have a new column, con_id, to support the pluggable database feature.

```
                                  Current        Max Initial
RESOURCE_NAME                       Util       Util Alloc           Limit
------------------------------ ---------- ---------- --------------- ----------
ges_ress                           123090     689253     131128       UNLIMITED
ges_locks                           91215     524188     293045       UNLIMITED
gcs_resources                      755053    1167954    4135261       4135261
gcs_shadows                        823219     874225    4135261       4135261
```

If you have access and license to performance metrics stored in AWR (Automatic Workload Repository), you can review the growth and shrink of these areas over a period using the code listed in Listing 11-1.

Listing 11-1. History of SGA Allocations for RAC-Specific Allocations

```
select
    to_date(to_char(trunc(begin_interval_time), 'DD-MON-YYYY'), 'DD-MON-YYYY') DAY,
    instance_number,
    trunc(max(bytes/1024/1024),2)  sz_MB
  from
   (select begin_interval_time, s.instance_number, sum(bytes) bytes
    from
      dba_hist_sgastat g, dba_hist_snapshot s
   where (name like '%ges%' or name like '%gcs%')
    and trunc(begin_interval_time) >= sysdate -30
    and s.snap_id = g.snap_id
    and s.instance_number = g.instance_number
    group by begin_interval_time, s.instance_number
   )
  group by
    to_date(to_char(trunc(begin_interval_time), 'DD-MON-YYYY'), 'DD-MON-YYYY'),
    instance_number
order by 1
/
DAY             INSTANCE_NUMBER      SZ_MB
--------------- --------------- ----------
...
22-SEP-12                     1    1347.56
22-SEP-12                     2     599.44
22-SEP-12                     3    1358.17
```

To accommodate the increase in RAC-specific memory allocations, you should consider increasing the size of shared pool (shared_pool_size parameter) when you convert the database from single instance to a RAC database.[2] Along the same line of reasoning, you should consider increasing the size of shared pool if you increase buffer cache size to accommodate the increase in GCS allocation size.

[2]Required increase in shared_pool_size depends upon your SGA size, current shared_pool size, Database software version, etc. However, a 20% increase in single-instance shared_pool _size and a 10% increase in buffer cache is a good preliminary design. Sandesh Rao, Director of RAC Assurance Team, agrees with this recommendation.

Resource Types

There are two categories of global resource types.

1. BL resource: BL (Buffer Lock) resources are at the heart of cache fusion processing and protect buffer cache buffers in the SGA from concurrent modifications by multiple instances at any given time. Before a buffer of a database block is modified, lock on a BL resource is acquired in a specific lock mode. GCS layer manages BL resources.
2. Non-BL resources: Resources such as TX, TM, SQ, etc., are few examples of non-BL resources. GES layer manages non-BL resources.

There are subtle differences between the implementation of these two types of resources, and I will discuss non-BL resources in this chapter. BL resource specifics are discussed in Chapter 10.

Let me explain the locking scheme with an example. Consider a table T1 with object_id of 11984883. Before altering table T1 with a DDL statement, such as dropping a column from the table or locking the table in exclusive mode, a lock on a TM resource must be acquired. A TM resource is allocated to protect this table T1 and uniquely represents table T1, as id1 attribute of that resource is set to object_id of the table.

In the following code, table T1 is locked in exclusive mode. A lock structure is visible in gv$lock view with id1 column value set to 11984883. Also, note that locking mode of the resource is set to 6 (exclusive mode). The following output indicates single-instance locking scheme.

```
REM First let us lock a small table in exclusive mode.
SQL> LOCK TABLE t1 IN exclusive MODE;
Table locked.

REM Query gv$lock to review single-instance locks.
SQL> SELECT sid, type, id1, id2, lmode, request FROM gv$lock WHERE type='TM';

       SID TY        ID1        ID2      LMODE    REQUEST
---------- -- ---------- ---------- ---------- ----------
         4 TM   11984883          0          6          0
```

In RAC, single-instance resources and locks are globalized as global resources. In Listing 11-2, a global resource is seen in gv$ges_resource with a resource name [0xb6dff3][0x0],[TM] to protect table T1. This resource_name is coined using object_id of the table, providing a one-to-one mapping between the resource and the table T1. Resource name is a string concatenated with three components: [0xb6dff3] is object_id in hexadecimal representation, the second component [0x0] is not applicable to TM resource type (set to zero), and the third component [TM] is a string identifying resource type.

Listing 11-2. Resource Name in ges_resource

```
REM Global resource is created with an unique resource name referring to
REM object_id of the table.

SQL> SELECT inst_id, resource_name, master_node, on_convert_q, on_grant_q
     FROM gv$ges_resource WHERE resource_name LIKE '[0xb6dff3]%TM%' ;

   INST_ID RESOURCE_NAME                  MASTER_NODE ON_CONVERT_Q ON_GRANT_Q
---------- ------------------------------ ----------- ------------ ----------
         1 [0xb6dff3][0x0],[TM][ext 0x0,0           0            0          1
```

RESOURCE_NAME

Interestingly, you can also identify the resource_name using the oradebug command. The ffollowing code uses the oradebug command[3] to print the resource details to a trace file by accepting decimal representation. After connecting to the database as SYS user:

```
REM Set my process for Oradebug
SQL>oradebug setmypid

REM Print the resource details for the object_id (three components)

REM Use the object_id of your table, in this case, object_id=11984883 for table T1.
SQL>oradebug lkdebug -O 11984883 0 TM

REM Print the trace file name
SQL>oradebug tracefile_name
/opt/app/product/11.2.0.2/admin/diag/rdbms/RACD/trace/RACD_ora_920.trc
```

Review of the trace file shows the details about resource_name: [0xb6dff3][0x0],[TM] is printed in the trace file.

Note that column `master_node` indicates the master node of the resource. This resource is mastered by instance with inst_id=1. Also, `master_node` column values start at 0; in contrast, inst_id column values start with 1. Hence, `master_node=0` indicates inst_id=1.

Next, I will query gv$ges_enqueue to review locking structures. Locks associated with the global resource `[0xb6dff3][0x0],[TM]` are visible in the following code output. The grant_level of the lock is KJUSEREX, exclusive mode. I will discuss locking modes in the next section.

```
REM Locks are acquired on that global resource in an exclusive mode.
Set serveroutput on size 100000
BEGIN
   print_Table (q'#
    select  inst_id, resource_name1, grant_level, request_level,state, blocked, blocker from
    gv$ges_enqueue where resource_name1 like '[0xb6dff3]%TM%'
    #');
 END;
/
INST_ID                       : 1
RESOURCE_NAME1                : [0xb6dff3][0x0],[TM][ext 0x0,0
GRANT_LEVEL                   : KJUSEREX
REQUEST_LEVEL                 : KJUSEREX
STATE                         : GRANTED
BLOCKED                       : 0
BLOCKER                       : 0
```

[3]This section is mostly academic and merely provided to improve understanding of RAC internal structures. Use of oradebug command in a production environment is not advised, and do not try these commands in a critical database.

■ **Note** I am using a procedure named print_table to pretty print the columns. Print_table is a PL/SQL procedure written by Tom Kyte. In a nutshell, that procedure prints the column values in a row format. We will be using the print_table procedure in our book to improve readability of the code output. Search on your favorite search engine for "print_table tom kyte" or it can also be downloaded from the following URL: http://asktom.oracle.com/pls/apex/f?p=100:11:0::::P11_QUESTION_ID:1035431863958.

Lock Modes

Resources can be locked in various modes in RAC too. In single instance, six locking modes (1 through 6) are available to lock a resource. Similarly, RAC too has six locking modes. In fact, there is a one-to-one mapping between a single-instance locking mode and RAC locking mode, as shown in Table 11-1. Semantics of these single-instance locking modes are functionally equivalent to locking modes in RAC. Table 11-1 shows locking modes and their meaning.

Table 11-1. *Lock Modes*

Single Instance	Single-Instance Comment	RAC	RAC Comment
1	NULL	KJUSERNL	Null mode. Placeholder lock, acquired so that process can be notified if there is any change in resource state.
2	Row Share	KJUSERCR	Row Share—Concurrent read.
3	Row Exclusive	KJUSERCW	Row Exclusive—Concurrent write.
4	Share	KJUSERPR	Protected read, write allowed if only one holds share mode.
5	Share Row Exclusive	KJUSERPW	One process can modify, others can read.
6	Exclusive	KJUSEREX	One process holds exclusive lock. Others can't access.

It is important to understand most common locking modes for TX/TM resources. I will discuss the meaning of TX/TM lock modes later in this chapter.

Lock-Related Views

Dynamic view gv$lock shows all *single*-instance locks *globally*. View gv$lock is based upon the fixed table x$ksqst. Accessing gv$lock view retrieves data from x$ksqst table from all instances, processes rows, and returns the result set. Rows from the gv$lock and gv$resource views do not retrieve any information from GRD.

In contrast, Dynamic view gv$ges_resource shows global resource and it is based upon two fixed tables: X$KJBR and X$KJIRFT. Fixed table X$KJBR externalizes resources structures of BL resource type and fixed table X$KJIRFT externalizes resource structures of non-BL resource types such as TX, TM, etc. Similarly, gv$ges_enqueue is based upon the fixed tables X$KJBL and X$KJILKFT. Fixed table X$KJBL externalizes BL locks and X$KJILKFT externalizes non-BL locks. These four fixed tables are at the heart of the global locking scheme in RAC.

■ **Note** Parallel servers named PZxx are employed to retrieve data from other instances if you query gv$ views. Session querying gv$ views acts as a query coordinator, parallel servers named PZxx (xx is number between 99 and 01) retrieve data from other instances, and the query coordinator process receives data and returns it to the user. From version 11g onward, even if you disable parallelism completely, PZ process can be spawned in other instances. Also, PZ parallel server process starts with 99 and decrements for each additional PZ process. For example, the first query accessing gv$ views will use PZ99 server processes in all nodes, and the second *concurrent* query accessing gv$ views will use PZ98 parallel server process.

To review global locks, we need to access GES layer-level views. All currently held locks can be reviewed by querying gv$ges_enqueue. Currently blocked locks can be reviewed by querying v$ges_blocking_enqueue. Interpretations and methods will be discussed while discussing individual resource types.

Pluggable Databases (12c)

Oracle Database version 12c introduces the pluggable databases feature, which allows many pluggable databases to be resident in a container database.

Pluggable databases bring an interesting problem for resource names. For example, for the table resources (TM resource type), object_id is used to coin the resource name, as object_id is unique prior to 12c. But in 12c, object_id is unique only within a pluggable database. As GES layer is a common service to all pluggable databases, it is not sufficient to just use object_id alone to coin the resource_name.

So, from version 12c onward, resource_name is coded with container_id of the PDB also. For example, I created two PDBs (hrdb1 and hrdb2) in a container database and managed to create a table GSTEST with the same object_id:89897. As you can see in the following, the ext attribute of the resource_name is coded with container_id of the PDB to make the resource_name unique. In the following, the first resource is for GSTEST table created in hrdb1 PDB with container_id equal to 3 and second resource is for the GSTEST table created in hrdb2 PDB with container_id equal to 4.

```
select resource_name from gv$ges_resource where resource_name like '[0x15f29][0x0],[TM]%'
/
RESOURCE_NAME
------------------------------
[0x15f29][0x0],[TM][ext 0x3,0x  ← hrdb1 PDB – GSTEST table –object_id=89897
[0x15f29][0x0],[TM][ext 0x4,0x  ← hrdb2 PDB – GSTEST table –object_id=89897

select resource_name1, con_id, state, grant_level
from gv$ges_enqueue where resource_name1 like '[0x15f29][0x0],[TM]%';

RESOURCE_NAME1                   CON_ID STATE              GRANT_LEV
------------------------------ ---------- ------------------- ---------
[0x15f29][0x0],[TM][ext 0x4,0x        0 GRANTED             KJUSEREX
[0x15f29][0x0],[TM][ext 0x3,0x        0 GRANTED             KJUSEREX
```

So, from version 12c onward, you should pay close attention to ext attribute of resource_name to identify PDB of the object.

Troubleshooting Locking Contention

To understand locking contention, we need to understand holders and waiters of global locks. A handful of global GES views are useful to understand the locking contention. I will use a small example to illustrate the views and methods to interpret the data from global views.

In this example, I will create a blocking lock between two sessions connecting to two different instances. Refer to the following code: session 1 locked the table T1 in exclusive mode. Session 2 tries to lock the table in exclusive mode while session 1 is holding the lock. Since there is an exclusive lock already held on the table T1, session 2 enqueues waiting for session 1 to release the lock.

```
REM From session 1, lock table t1 from instance 1.

SQL> lock table t1 in exclusive mode;
Table locked.

REM From session 2, try to lock table t1 from instance 2.

SQL> lock table t1 in exclusive mode;
<.. Session 2 is waiting for the lock..>
```

Output of gv$ges_blocking_enqueue is printed in Listing 11-3. Querying gv$ges_blocking_enqueue, you can see the following details about all blocked locks:

1. Column `resource_name` uniquely identifies the table T1, as the first part of the string 0xb6dff3 is coded with object_id of the table in hexadecimal format.
2. Column `STATE` is set to `GRANTED`, implying that lock on the resource has been granted to PID:17665. Column value OPENING means that the process is waiting for the lock to be available.
3. Column `BLOCKED` indicates if the process is blocked or not. A value of 1 in the BLOCKED column indicates that process is blocked and waiting for lock to be available.
4. Column `BLOCKER` indicates if the process is blocking another process or not. A value of 1 in the `BLOCKER` column indicates that the process is blocking another process.
5. Column `owner_node` indicates the owner node for that lock. PID 3914 is connected to instance 1 (inst_id=2) and PID 17665 is connected to instance 0 (inst_id=1).

Listing 11-3. gv$ges_blocking_enqueue

```
REM At this time, session 2 will be waiting for locks. Let's review global locks.

SQL> col inst_id format 99
SQL> col owner_node format 99 head 'Owner|Node'

SQL> SELECT inst_id, pid, resource_name1, state, owner_node, blocked, blocker
     FROM gv$ges_blocking_enqueue
     ORDER BY resource_name1;
                                                          Owner
INST_ID        PID RESOURCE_NAME1                STATE       Node    BLOCKED    BLOCKER
------- ---------- ------------------------------ ---------- ----- ---------- ----------
      1      17665 [0xb6dff3][0x0],[TM][ext 0x0,0 GRANTED        0          0          1
      2       3914 [0xb6dff3][0x0],[TM][ext 0x0,0 OPENING        1          1          0
```

View gv$ges_blocking_enqueue can be joined to gv$process to retrieve session-level details. Listing 11-4 prints a useful script to retrieve session-level details. Session with a SID of 943 is holding a lock in KJUSEREX mode and session 4312 is waiting for the resource to be available. In this example, the resource type is TM, since we are locking the table object.

Listing 11-4. ges_blocking_locks.sql Script

```
REM Author: Riyaj Shamsudeen
REM ges_blocking_locks.sql script
col state format A10
col pid format 99999999
set serveroutput on size 100000
begin
 print_Table ('
        with dl as (
        SELECT inst_id, resource_name1, grant_level, request_level,
          transaction_id0, which_queue, state, pid, blocked ,
          blocker
        FROM gv$ges_blocking_enqueue
        )
        SELECT dl.inst_id, dl.resource_name1, dl.grant_level,
          dl.request_level, dl.state, s.sid, sw.event,
          sw.seconds_in_wait sec
        FROM dl,
          gv$process p, gv$session s, gv$session_wait sw
        WHERE (dl.inst_id = p.inst_id AND dl.pid        = p.spid)
        AND (p.inst_id    = s.inst_id AND p.addr        = s.paddr)
        AND (s.inst_id    = sw.inst_id AND s.sid        = sw.sid)
        ORDER BY sw.seconds_in_wait DESC
');
end;
/

INST_ID                       : 1
RESOURCE_NAME1                : [0xb6dff3][0x0],[TM][ext 0x0,0
GRANT_LEVEL                   : KJUSEREX
REQUEST_LEVEL                 : KJUSEREX
STATE                         : GRANTED
SID                           : 943
EVENT                         : PX Deq: Execute Reply
SEC                           : 0
-----------------
INST_ID                       : 2
RESOURCE_NAME1                : [0xb6dff3][0x0],[TM][ext 0x0,0
GRANT_LEVEL                   : KJUSERNL
REQUEST_LEVEL                 : KJUSEREX
STATE                         : OPENING
SID                           : 4312
EVENT                         : enq: TM - contention
SEC                           : 1798
```

Enqueue Contention

In this section, I will discuss locking contention, which is frequently seen in a typical RAC database.

TX Enqueue Contention

Oracle database provide row-level locking abilities for a fine-grained locking granularity. Sessions trying to modify a row locked by another transaction must wait for the transaction to commit or rollback before modifying the row. In a single-instance database, row-level locking scheme is implemented using TX resources and locks; *a* TX resource protects *one* transaction. A session initiating a transaction will hold exclusive mode lock on a TX resource, and a session waiting to modify the row locked by the first transaction will request exclusive mode lock on the lock holder's TX resource. After the lock holder completes the transaction, that is, commit or rollback, holding session will release the lock on TX resource. As the lock request is complete, lock waiter will continue to modify the row.

In a RAC database, sessions connected to two different instances could try to modify *a* row. So, TX resources and locks are globalized in a RAC database. A global locking scheme is employed to protect the TX resources globally. I will explain TX resource contention in a RAC database using a simple example: In the following code, session #1 updated a row, thereby initiating a new transaction. Transaction_id of the transaction is `1557.11.9239` as retrieved using `dbms_transaction` packaged call. From session #2, I will try to update the same row and since the row is locked by session #1, session #2 will wait for the lock to be available. Essentially, session #2 will wait for the transaction initiated by session #1 to complete.

```
REM From session 1, I will lock a row. SID=6445
SQL> update t1 set n1=n1 where n1=100;

select dbms_Transaction.LOCAL_TRANSACTION_ID from dual;

LOCAL_TRANSACTION_ID
--------------------
1557.11.9239

REM From session 2 connected to a different instance, I will update the same row.
REM  SID=12229
SQL> update t1 set n1=n1 where n1=100;
```

In single-instance views, TX resources are visible in v$lock family of views. In the following output, a TX resource with id1=102039563 and id2=9239 is held in exclusive mode by session 6445. Session with SID=12229 is requesting a lock on the same TX resource in exclusive mode. The combination of enqueue types id1 and id2 uniquely identifies a transaction.

```
SELECT sid, type, id1, id2, lmode, request
FROM gv$lock WHERE (type, id1, id2) IN
  (SELECT type, id1, id2 FROM gv$lock WHERE request>0) ;
   SID TY        ID1        ID2 LMODE REQUEST
------ -- ---------- ---------- ----- -------
  6445 TX  102039563       9239     6       0
 12229 TX  102039563       9239     0       6
```

Now, let's review the global locks. In Listing 11-5, output of ges_blocking_locks.sql script shows that session 6445 is holding a lock on a global TX resource in KJUSEREX mode, and session 12229 is waiting for the lock and has requested the lock in KJUSEREX mode. State `OPENING` indicates that the session is waiting for the resource to be available. Wait event confirms the row-level lock wait.

Listing 11-5. TX Enqueue Contention

```
REM Reviewing the output of ges_blocking_locks.sql script, session 12229 is
REM waitingfor TX lock. Session 6445 is holding a TX enqueue.
REM Script is printed in Listing 11-4.
SQL> @ges_blocking_locks.sql

INST_ID                        : 2
RESOURCE_NAME1                 : [0x615000b][0x2417],[TX][ext 0
GRANT_LEVEL                    : KJUSEREX
REQUEST_LEVEL                  : KJUSEREX
STATE                          : GRANTED
SID                            : 6445
EVENT                          : ges remote message
SEC                            : 0
-----------------
INST_ID                        : 1
RESOURCE_NAME1                 : [0x615000b][0x2417],[TX][ext 0
GRANT_LEVEL                    : KJUSERNL
REQUEST_LEVEL                  : KJUSEREX
STATE                          : OPENING
SID                            : 12229
EVENT                          : enq: TX - row lock contention
SEC                            : 462
```

Transaction_id is used to coin a unique resource_name [0x615000b][0x2417],[TX] in GRD. The first part of string [0x615000b] is the concatenation of strings 615 and 000b. 0x615 is a hexadecimal representation of decimal 1557, and 0x000b is the hexadecimal representation of 11. The second part of string 0x2417 is the hexadecimal representation of 9239. These three parts, 1557.11.9239, combined together constitute the transaction_id we queried using dbms_transaction package call.

■ **Note** gv$ges_blocking_enqueue encompasses both GES and GCS layer locks. If your buffer cache is huge, in the order of hundreds of gigabytes, accessing this gv$ view can be slower.

Oracle Development introduces code optimization in new releases, and some of those features are quietly introduced. For example, from version 11g, TX resource visibility stays local until a session in another instance tries to acquire a lock on that TX resource. This optimistic strategy improves RDBMS efficiency, as most transactions will complete without inducing or suffering from locking waits. Only a few percent of transactions will suffer from locking contention and hence it is efficient to defer maintenance of GES structures until it is absolutely necessary to do so. With this optimization, code path is reduced as the maintenance of global structures is completely avoided for most transactions. Further, this optimization is very useful in reducing global cache workload in the database using application affinity.

In addition to TX resources, the row-level locking scheme must protect the base table from any incompatible change also. So, lock on a TM resource protecting the table is acquired in a single-instance database. In RAC, TM locks are globalized as GRD resources, and locks are acquired in row share mode.

I will a fabricate TM resource_name by converting object_id of T1 table to a string of format [0x*obj_id*] [0x0],[TM]. This resource_name will be used to search in gv$ges_resource in Listing 11-6. The following code uses built-in functions to coin the resource_name.

```
REM Derive a resource_name string from object_id.
SELECT DISTINCT '[0x'
    ||trim(TO_CHAR(object_id, 'xxxxxxxx'))
    ||'][0x'
    || trim(TO_CHAR(0,'xxxx'))
    || '],[TM]' res
FROM dba_objects WHERE object_name=upper('&objname')
AND owner=upper('&owner') AND object_type LIKE 'TABLE%';
Enter value for objname: T1
Enter value for owner: RS
RES
---------------------------
[0xb6dff3][0x0],[TM]
```

In Listing 11-6, a SQL statement queries gv$ges_resource and searches for the resources with the coined resource name. A global resource is visible in both nodes. Also, note that master_node of the resource is set to 0 (inst_id=1). Querying gv$ges_enqueue, we see that two locks have been acquired on this resource, in KJUSERCW mode (Mode KJUSERCW—Row Exclusive or Concurrent Write—is compatible with another KJUSERCW mode).

Listing 11-6. TM Resources

```
REM Using the derived resource_name, identify all GES resources.

SELECT inst_id, resource_name, master_node
FROM gv$ges_resource WHERE resource_name LIKE '[0xb6dff3][0x0],[TM]%' ;

   INST_ID RESOURCE_NAME                  MASTER_NODE
---------- ------------------------------ -----------
         1 [0xb6dff3][0x0],[TM][ext 0x0,0           0
         2 [0xb6dff3][0x0],[TM][ext 0x0,0           0

REM Identify all GES locks for that resource.

col state format a10
col inst_id format 99 head Inst
col owner_node format 99 head 'Owner|Node'

SQL> SELECT inst_id, resource_name1, pid, state,
            owner_node , grant_level, request_level
     FROM gv$ges_enqueue
     WHERE resource_name1 LIKE '[0xb6dff3][0x0],[TM]%' ;
                                                          Owner
Inst RESOURCE_NAME1                   PID STATE       Node GRANT_LEV REQUEST_L
---- ------------------------------ ---------- ---------- ----- --------- ---------
   2 [0xb6dff3][0x0],[TM][ext 0x0,0      20444 GRANTED        1 KJUSERCW  KJUSERCW
   1 [0xb6dff3][0x0],[TM][ext 0x0,0      21877 GRANTED        0 KJUSERCW  KJUSERCW
```

In summary, in a RAC database, both TX and TM resources are used to implement the row-level locking scheme, a scheme similar to the single-instance database locking scheme. While it is possible to understand row-level locks by querying the gv$lock family of views, it is a better practice to review GRD-level locks also.

Table 11-2 provides a few common reasons for TX enqueue contention and possible causes that can help you to resolve the contention.

Table 11-2. *TX Lock Contention and Modes*

Lock Mode	Request Mode	Comment	RAC Equivalent (request mode)	Possible causes
6	6	Exclusive	KJUSEREX	Classic row-level locking contention. Usually, an application coding or concurrency issue. Also, check holder to review why the lock is not released quickly.
6	4	Share	KJUSERPR	1. Block-level concurrency issue (initrans is lower). 2. Concurrent inserts and updates in the presence of a unique/primary key index or constraint. 3. Updates to column(s) with bitmap indexes.

TM Enqueue Contention

TM resource and enqueue protects table-level concurrency, as we saw in the previous section. To further your understanding of TM resource contention, I will create TM enqueue contention and review the resource structures.

In Listing 11-6, I explained how TM locks are acquired to implement row-level locking strategy. I will continue with the same setup as a starting point for this example. While session 12229 is waiting to lock a row, I will try to lock the table in exclusive mode from a different session connected to instance 3. Since the request mode is incompatible with other existing locking modes, session 3 will wait for the lock to be available.

```
REM This listing continues from Listing 11-4.
REM From session 3, we will try to lock the table.
SQL> lock table t1 in exclusive mode;
...
REM From another session, let's review resources and locks.

SQL> SELECT DISTINCT '[0x'
    ||trim(TO_CHAR(object_id, 'xxxxxxxx'))
    ||'][0x'
    || trim(TO_CHAR(0,'xxxx'))
    || '],[TM]' res
FROM dba_objects WHERE object_name=upper('&objname');
RES
--------------------------
[0xb6dff3][0x0],[TM]
```

Output of gv$ges_resource shows that resource is set up in all instances.

```
SQL> SELECT inst_id, resource_name, master_node
     FROM gv$ges_resource WHERE resource_name LIKE '[0xb6dff3][0x0],[TM]%' ;

Inst RESOURCE_NAME                  MASTER_NODE
---- ------------------------------ -----------
   1 [0xb6dff3][0x0],[TM][ext 0x0,0           0
   2 [0xb6dff3][0x0],[TM][ext 0x0,0           0
   3 [0xb6dff3][0x0],[TM][ext 0x0,0           0
```

Further, from the output of gv$ges_enqueue, we can see that Process with PID 404 is requesting a lock on the resource [0xb6dff3][0x0],[TM] in KJUSEREX mode. Since that resource is already held by other two processes in KJUSERCW mode (which is incompatible with KJUSEREX mode), PID 404 is waiting for the lock to be available. State of the lock for PID 404 is set to OPENING, indicating that lock is not granted yet.

```
col state format a10
col inst_id format 99 head Inst
col owner_node format 99 head 'Owner|Node'

SQL> SELECT inst_id, resource_name1, pid, state,
            owner_node , grant_level, request_level
     FROM gv$ges_enqueue
     WHERE resource_name1 LIKE '[0xb6dff3][0x0],[TM]%' ;
                                                          Owner
Inst RESOURCE_NAME1                          PID STATE       Node GRANT_LEV REQUEST_L
---- ------------------------------ ---------- ---------- ----- --------- ---------
   1 [0xb6dff3][0x0],[TM][ext 0x0,0      21877 GRANTED         0 KJUSERCW  KJUSERCW
   2 [0xb6dff3][0x0],[TM][ext 0x0,0      20444 GRANTED         1 KJUSERCW  KJUSERCW
   3 [0xb6dff3][0x0],[TM][ext 0x0,0        404 OPENING         2 KJUSERNL  KJUSEREX
```

Reviewing the output of ges_blocking_locks.sql script, we can identify that SID 8013 is requesting the lock in KJUSEREX mode and waiting on event "enq: TM – contention."

```
INST_ID                   : 3
RESOURCE_NAME1            : [0xb6dff3][0x0],[TM][ext 0x0,0
GRANT_LEVEL               : KJUSERNL
REQUEST_LEVEL             : KJUSEREX
STATE                     : OPENING
SID                       : 8013
EVENT                     : enq: TM - contention
SEC                       : 3516
```

Table 11-3 lists the most commonly seen TM resource request modes in a typical application life cycle. This table also guides common solutions to the locking contention with the listed request modes.

Table 11-3. *TM Lock Contention and Modes*

Lock Mode	Request Mode	Comment	RAC Equivalent	Possible causes
3	6	Exclusive	KJUSEREX	Identify why a table is locked in exclusive mode. Possibly, DDL statements on the table.
3, 4	4	Share	KJUSERPR	Unindexed foreign keys, lock in share mode, etc.

HW Enqueue Contention

High-Water Mark (HWM) of a segment indicates the highest-ever used block of that segment; essentially, a marker between used space and unused space. Full Table Scan access path reads all blocks below HWM. During a DML operation, to populate rows, new blocks may need to be formatted and that would require HWM to be increased. Changes to the HWM are performed under the protection of HW enqueues. During excessive concurrent insert/update workload, processes can wait for HW enqueue contention.

■ **Note** HWM is increased five blocks at a time. Undocumented parameter _bump_highwater_mark controls the number of blocks to increase. There is no reason to modify this parameter for normal workload, though.

Since HW enqueues are also globalized in a RAC environment, HW enqueue contention can be magnified. Generally, this problem is not prevalent in ASSM tablespaces. If you encounter HW enqueue contention in an ASSM tablespace, then that would generally indicate that the object growth rate is higher than the speed of ASSM metadata manipulation. You might have to either preallocate the extents or partition the object to reduce the severity of HW enqueue contention.

There are a few common reasons for HW contention:

1. Heavy insert activity in to a non-partitioned table: Numerous processes insert in one segment and that segment grows by leaps and bounds, leading to contention issues with HW enqueues. Partitioning the table is a better practice, so that pressure on HW enqueue is distributed across many segments.
2. Indexes splitting and growing faster: Determine if indexes are needed during insert activity. If so, hash partitioning indexes might be helpful.
3. Prior to Oracle Database 10g, LOB column segments were extended one block at a time. This bug has been fixed in Oracle Database version 11g, and so upgrading to the latest version is useful to resolve the problem if the segment is a LOB segment.

DFS Lock Handle

DFS stands for Distributed File System, an older name from Oracle Parallel Server (predecessor to Oracle RAC) versions. However, in RAC, this wait event is overused in various contexts. For example, DFS lock handle mechanism is also used to trigger a predefined action, such as global checkpoint, in an instance background process (both remote and local instances).

Consider a DDL statement truncating a table T1. Table blocks of T1 can be resident in the buffer cache of any instance. So, before the truncate statement is successful, buffers belonging to that table must be invalidated[4] from all instances. This invalidation would require a message to be sent to the remote DBW process, and that communication mechanism is implemented using DFS lock handle mechanism. Essentially, requesting a lock handle on a lock type will trigger predefined action in the background process indirectly. While the background process is working to complete the triggered action, foreground process is instrumented to wait on DFS lock handle wait event. The exact sequence of messaging between the background processes is version dependent and so, we will see only practical methods to debug "DFS lock handle" waits.

A resource and lock lurks underneath DFS lock handle mechanism. To understand the root cause of DFS lock handle waits, you need to identify the resource type and mode requested. In Listing 11-7, a line from a SQL trace file is shows a wait time of 4.6 ms for `DFS lock handle` wait event. Attribute p1 of the event is set to 1398145029. That innocuous-looking number can be used to understand more details. Hexadecimal representation of 1398145029 is 0x53560005. The first part of the string 5356 represents the lock type in ASCII format. In this example, the value of 53 maps to ASCII character S, and the value of 56 maps to ASCII character V; therefore, 5356 represents lock type SV. Lock mode is 0005, which is KJUSERPW mode. Listing 11-7 also provides a SQL statement to convert the value of p1 from decimal to lock type and mode.

[4]Exact terminology is *Checkpoint Range and Invalidate*.

Listing 11-7. DFS Lock Handle Trace File Output

```
REM A line from a trace file.
nam='DFS lock handle' ela= 4362 type|mode=1398145029 id1=86033 id2=0 obj#=-1 ...

SQL> SELECT chr(bitand(&&p1,-16777216)/16777215) || chr(bitand(&&p1,16711680)/65535) type,
     mod(&&p1, 16) md
     FROM dual
/
Enter value for p1: 1398145029
TY MD
-- ----------
SV 5
```

You can query v$lock_type to understand the functionality of a lock type. In this example, SV lock type is used for Sequence Ordering, and `id1` column value indicates the object_id. Lock type SV will be covered later in this chapter.

```
exec print_table ( 'select * from v$lock_type where type=''SV''');
TYPE                          : SV
NAME                          : Sequence Ordering
ID1_TAG                       : object #
ID2_TAG                       : 0
IS_USER                       : NO
DESCRIPTION                   : Lock to ensure ordered sequence allocation in RAC mode
```

After identifying the lock type and mode, you can probe further to understand the root cause of DFS lock handle contention. In the next section, I will probe individual lock types and discuss practical methods to resolve the locking contention.

SV Resources

As discussed in the preceding section, SV enqueue is used for sequence-related actions. Each instance caches sequence values in SGA. In RAC, due to instance-level caching, queried sequence values may not correlate well with time-based column values. Designers typically tend to resolve this problem by creating sequences with nocache or ORDER attributes.

The CACHE attribute of a sequence dictates that sequence values are cached in instance SGA. The ORDER attribute of a sequence dictates that values will be retrieved in strict sequential order. Every retrieval from a sequence with ORDER NOCACHE attribute modifies the seq$ dictionary table to guarantee a strict sequential order. This method of retrieval leads to excessive amount of "row cache lock" waits (changes to data dictionary tables require row cache locks to be held) and global cache wait events.

If a sequence is defined with ORDER CACHE attributes, then GES layer is used to maintain the strict sequential order. Essentially, DFS lock handle mechanism with an SV resource is used for this purpose. To explain DFS lock handle contention for SV resource, I will create a sequence `T1_SEQ` with `ORDER` and `CACHE` attributes. Then, I will access the sequence a few times and increase the sequence value to 9.

```
REM Create a sequence with cache and order attributes.
create sequence T1_SEQ order cache 100;

SQL> select t1_seq.nextval from dual;
```

```
   NEXTVAL
----------
         1
...

REM selecting couple of times from this sequence from two nodes to bump up the value to 9
REM Note: Not all SELECT statements are printed for brevity.
SQL> select t1_seq.nextval from dual;

   NEXTVAL
----------
         9
```

Resources created in GRD for sequences with CACHE ORDER attributes uses the familiar naming pattern discussed earlier in this chapter. Object_id of the sequence and SV lock type is used to fabricate a resource name.[5] In this example, resource_name is `[0xb6e64b][0x0],[SV]`; the first part of the string, `0xb6e64b`, is object_id of the sequence in hexadecimal format; the second part of the string is set to 0 (not applicable to SV resource types); and the third part of the string is SV, indicating the resource type. The following query shows the SV resource_name for the T1_SEQ sequence.

```
REM query gv$ges_resource to identify the resource name.
col res  new_value resource_name

SELECT DISTINCT '[0x'
    ||trim(TO_CHAR(object_id, 'xxxxxxxx'))
    ||'][0x'
    || trim(TO_CHAR(0,'xxxx'))
    || '],[SV]' res
FROM dba_objects WHERE object_name=upper('&objname')
     AND owner=upper('&owner') AND object_type LIKE 'SEQUENCE%'
/
Objname: T1_SEQ
Owner: RS
RES
---------------------------
[0xb6e64b][0x0],[SV]
```

In Listing 11-8, view gv$ges_resource is queried to identify the resources in GRD. Column value of `value_blk` is of special interest. Column `value_blk` of the resource holds the current value of the sequence; in this example, notice that the first few bytes of the `value_blk` are set to `0x82c10b`. The second part of that value is `c10b`, which is the internal representation of value 10. Essentially, value_blk column of SV resource holds current sequence value.

Listing 11-8. Sequence Value and SV Resource

```
REM Using the derived resource_name, identify all GES resources.

col master_node head 'Mast|node' format 99
col value_blk format a30
SQL> SELECT inst_id, resource_name, master_node, value_blk
     FROM gv$ges_resource WHERE resource_name LIKE '&resource_name%'
/
```

[5]In 12c, PDB container_id is also coded in to the resource name.

```
                                  Mast
Inst RESOURCE_NAME                node VALUE_BLK
---- ---------------------------- ---- ----------------------------------------
   2 [0xb6e64b][0x0],[SV][ext 0x0,0    2 0x82c10b00000000000000000000000000 .
```

Oracle Database uses special representation to store the numbers, apparently for cross-platform compatibility reasons. For example, a value of 10 is represented as a two-byte value. The following two SQL statements can be used to identify the internal representation:

```
SQL> select dump(10) from dual;
DUMP(10)
-------------------
Typ=2 Len=2: 193,11
```

Converting the values 193 and 11, we get c10b:

```
select to_char(193,'xx')||to_char(11,'xx') from dual;

TO_CHA
------
 c1  b
```

RAC background processes coordinate the value_blk column value and return the next value to the foreground process. As expected, locks are also acquired by the background processes on the SV resource. In the following output, PID 24313 has KJUSERNL mode lock, and PID 24313 is LCK0 process.

```
REM Identify all GES locks for that resource.

col state format a10
col inst_id format 99 head Inst
col owner_node format 99 head 'Owner|Node'

SQL> SELECT inst_id, resource_name1, pid, state, owner_node , grant_level, request_level
     FROM gv$ges_enqueue WHERE resource_name1 LIKE '&resource_name%'
/
                                                      Owner
Inst RESOURCE_NAME1                     PID STATE       Node GRANT_LEV REQUEST_L
---- ---------------------------- ---------- ---------- ----- --------- ---------
   2 [0xb6e64b][0x0],[SV][ext 0x0,0      24313 GRANTED        1 KJUSERNL  KJUSERNL
   3 [0xb6e64b][0x0],[SV][ext 0x0,0       4416 GRANTED        2 KJUSERNL  KJUSERNL
   3 [0xb6e64b][0x0],[SV][ext 0x0,0          0 GRANTED        1 KJUSEREX  KJUSEREX
```

Let us understand how to identify the object_id associated with DFS lock handle wait event and SV resource type. The first line shows the DFS lock handle wait event. Column id1 in that trace file line shows the object_id of the sequence involved in DFS lock handle contention. Note that the obj# field value from the trace line is not referring to the sequence (as of release 11.2).

```
nam='DFS lock handle' ela= 827 type|mode=1398145029 id1=11986507 id2=0 obj#=-1

col object_name format a30
```

```
SQL> SELECT object_name, object_id FROM dba_objects WHERE object_id=11986507
/
OBJECT_NAME                    OBJECT_ID
------------------------------ ----------
T1_SEQ                          11986507
```

In summary, if the derived lock type from `DFS lock handle` wait event is SV, then identify the sequence_name and alter the sequence attributes matching with workload. Here are guidelines that will help you to decide the attributes of sequences:

1. If the sequence is heavily accessed, alter the sequence cache to upwards of 1000 with no ORDER attribute.
2. If the sequence is lightly accessed and if the requirement is to retrieve strictly sequential values, then use `ORDER CACHE` attribute.
3. If there is no requirement for ordered values, then use `CACHE` without any `ORDER` attribute.

CI Resources

Enqueues on CI resources are acquired to invoke instance-level action(s) in background processes. For example, truncating a table T1 would require buffers to be flushed from other nodes. So, DBW processes in all nodes must be posted to invalidate buffers in the buffer cache of that table T1. CKPT process coordinates the instance-level actions, and DBW process will do the actual scanning and writes.

ASM instances running in clustered mode use DFS lock handle to coordinate the activity too. For example, the addition of a data file or disk needs to modify extent map in all instances, and that coordination is performed through DFS lock handle mechanism.[6,7]

I will explain CI enqueues with a small example, by truncating a table with 10046 event trace on. Reviewing the trace file, we can identify the trace line marking the wait for DFS lock handle wait event. A few wait events from the trace file are printed below.

```
alter session set events '10046 trace name context forever, level 12';
truncate table t1;
alter session set events '10046 trace name context forever, level off';

REM from the trace file
...
nam='DFS lock handle' ela= 844 type|mode=1128857605 id1=13 id2=1 obj#=11984883
nam='DFS lock handle' ela= 914 type|mode=1128857605 id1=13 id2=3 obj#=11984883
nam='DFS lock handle' ela= 8159 type|mode=1128857605 id1=13 id2=2 obj#=11984883
...
```

Using the SQL statement shown in Listing 11-9, we can identify lock type and lock mode. In this case, lock type is CI and mode is 5.

[6]Sandesh Rao points out that any disk group operation such as add disk, drop disks, add files, etc., would trigger DFS lock handle–based concurrency control.

[7]This concurrency control is improved in version 12c as concurrent disk group operations can be performed in the same instance.

Listing 11-9. Identify Lock Type from p1.REM Identify Lock Type

```
SELECT chr(bitand(&&p1,-16777216)/16777215) || chr(bitand(&&p1,16711680)/65535) type,
     mod(&&p1, 16) md
     FROM dual
/
TY         MD
-- ----------
CI          5
```

There are numerous reasons for DFS lock handle waits with enqueue type of CI. Fortunately, note 34631.1 provides mapping between id1, id2 values and the semantics of the combination, but the list is not complete. You might need to engage Oracle support for further analysis, as performance issues due to CI resource type could be a software bug too.

A quick approach to gain more knowledge about specific id1, id2 value combinations of CI lock type is to identify the background process associated with that resource. Using id1=13 and id2=2, we can coin a resource name as `[0xd][0x2],[CI]` and search in gv$ges_enqueue view. In Listing 11-10, PID 23557 in node 2 is a DBW process holding locks on this resource. You can make an educated guess about the problem with SQL statement affected and the background process involved.

Listing 11-10. Resources with CI Enqueue

```
SELECT resource_name1, inst_id, pid, blocker, blocked
FROM gv$ges_enqueue
WHERE resource_name1 LIKE '%[0xd][0x2],[CI]%'
/
RESOURCE_NAME1                   INST_ID        PID    BLOCKER    BLOCKED
------------------------------ ---------- ---------- ---------- ----------
[0xd][0x2],[CI][ext 0x0,0x0]            2      23557          0          0
[0xd][0x2],[CI][ext 0x0,0x0]            3       4213          0          0
[0xd][0x2],[CI][ext 0x0,0x0]            1      10753          0          0
```

A few ASM-related CI events are printed in Table 11-4. For example, "ASM map resize message" event will use the resource [0x2a][0x0],[CI], where 0x2a is the hexadecimal representation of 42. These resources will be visible in ASM instance(s).

Table 11-4. *ASM-Specific CI – id1*

CI – id1	Meaning in ASM instance
42	ASM map resize message
43	ASM map lock message
44	ASM map unlock message (phase 1)

Events for ASM-related operations are usually short-lived. However, when you are add a new datafile to a tablespace or extend a datafile, in a database with higher activity, then ASM-related wait events will be prominent.

Note that, usually, DFS lock handle contention with CI as resource type is just a symptom of another root cause lurking underneath. For example, excessive truncates of tables in huge buffer caches induce DFS lock handle contention with CI resource type. The root cause in that example is excessive truncation, and use of global temporary

table is a recommended approach. So, to resolve DFS lock handle contention, you must understand the root cause before resolving the contention. Also, it is important to realize that every instance will have some DFS lock handle contention and only when the time spent waiting for DFS lock handle contention is a major contributor of wait time should you worry about the contention.

DFS Lock Handle Summary

Table 11-5 lists few common enqueue types associated with DFS lock handle mechanism. It is out of scope to discuss every DFS lock handle enqueue type.

Table 11-5. *DFS Lock Request—Enqueue Types*

Enqueue	Comment	Possible reason
SS	Sort Segment	Drop of sort segment or temporary tablespace configuration issues.
IV	Invalidation	Object invalidation due to a DDL statement.
CI	Cross Invocation	Many possible reasons. Check note 34631.1.

Library Cache Locks/Pins

Library cache locks protect library cache objects from concurrent modifications, and library cache pins protect a library cache object while that library object is busy in execution. In a single-instance database, parse locks on a SQL statement (cursors) implicitly acquire library cache locks on dependent objects. For example, when a table is altered then the library cache object for that table will be invalidated. That invalidation would recursively invalidate dependent cursors, requiring cursors to be reparsed on subsequent execution.

In a RAC database, dependency between the cursors and objects must be maintained globally. Local library cache locks are not sufficient to maintain the global dependency. For example, if you alter a table in instance 1, then all packages and cursors dependent upon the table must be invalidated in all instances. This invalidation is needed to trigger an automatic recompilation of library cache objects in the subsequent execution of cursors.

Global dependency for library cache object is maintained through GRD resources and locks. Library cache locks are globalized as resource types in the range of LA through LP, and library cache pins are globalized as NA through NZ types of resources. I will explain the global locks with an example. In Listing 11-11, a PL/SQL block is creating a dynamic SQL statement joining table `T_LIBTEST` 1024 times, and then parses the statement calling dbms_sql package. This example is artificial and serves no real-world purpose other than to show the use of GRD resources and locks.[8]

Listing 11-11. Library Cache Lock and Pin

```
REM create a table
CREATE TABLE t_libtest (n1 NUMBER );

REM dynamically populate a CLOB variable and try to parse it.
DECLARE
  v_sqltext CLOB;
  c1 NUMBER ;
```

[8]MOS note 122793.1 is also a good document to read about library cache lock contention.

```
BEGIN
  v_sqltext:= ' select a0.* from t_libtest a0';
  c1       := dbms_sql.open_cursor;
  FOR i    IN 1 .. 1024
  LOOP
    v_sqltext := v_sqltext ||' , t_libtest a'||i;
  END LOOP;
  dbms_output.put_line(v_sqltext);
  dbms_sql.parse( c1, v_sqltext, dbms_sql.native);
END;
/
```

GRD uses first 16 bytes of name hash value of the library cache object to construct a resource_name. The following SQL statement uses that algorithm to construct resource_name for library cache objects. I will use the resource_name constructed as follows to query gv$ views.

```
col res format a30
col owner format a30
col objname format a30
SELECT '[0x' || SUBSTR(kglnahsv, 1,8) || '][0x'|| SUBSTR(kglnahsv, 9, 8) || ']' res,
 kglnaown owner, kglnaobj objname
FROM x$kglob WHERE kglnaobj LIKE upper('&objname');

Enter Objname: T_LIBTEST
RES                            OWNER                          OBJNAME
------------------------------ ------------------------------ ------------------------------
[0xd9bcbc52][0xb0a18d29]       APPS                           T_LIBTEST
```

In Listing 11-12, `cursor c1` derives resource_name using the `kglnahsv` column from `x$kglob` table (Fixed table `x$kglob` keeps all library cache objects, and `kglnashv` column shows the name hash value). Using the derived resource_name, `cursor c2` searches gv$ges_resource to identify the resources, and `cursor c3` searches gv$ges_enqueue to identify locks associated with the resource_name.

Listing 11-12. Library Cache Lock Resources

```
SET serveroutput ON size 1000000
DECLARE
BEGIN
  FOR c1 IN
  (SELECT '[0x'|| SUBSTR(kglnahsv, 1,8) || '][0x'|| SUBSTR(kglnahsv, 9, 8)|| ']' res,
          kglnaown owner, kglnaobj objname
   FROM x$kglob WHERE kglnaobj LIKE upper('&objname')
  )
  LOOP
    dbms_output.put_line ('--------------------');
    dbms_output.put_line ('Object Details...' || c1.owner||'.'|| c1.objname);
    dbms_output.put_line ('--------------------');
    dbms_output.put_line ('--------------------');
    dbms_output.put_line ('Resource details...');
    dbms_output.put_line ('--------------------');
    FOR c2 IN
```

```
    (SELECT resource_name, master_node FROM v$ges_resource
     WHERE resource_name LIKE '%'||c1.res||'%'
    )
    LOOP
      dbms_output.put_line ('Resource name '|| c2.resource_name ||
                                ', Master '||c2.master_node );
    END LOOP;
    dbms_output.put_line ('-------------------');
    dbms_output.put_line ('Lock details...');
    dbms_output.put_line ('-------------------');
    FOR c3 IN
    (SELECT resource_name1, transaction_id0, pid,
      state, owner_node, grant_level
    FROM v$ges_enqueue WHERE resource_name1 LIKE '%'||c1.res ||'%'
     )
    LOOP
      dbms_output.put_line ('Res name '|| c3.resource_name1 || ', owner '||c3.owner_node );
      dbms_output.put_line ('...Transaction_id0 '|| c3.transaction_id0 ||
                        ',Level '||c3.grant_level|| ' State ' || c3.state );
    END LOOP;
  END LOOP;
END;
/
```

Refer to the output of Listing 11-12 printed in the following section: There are two types of resources—LB and NB types—for the library cache object of T1_LIBTEST table. Output from gv$ges_enqueue shows that locks are acquired by instance 1 in KJUSERPR mode while parsing the statement. LB resource type represents the global resource for library cache lock on table T1_LIBTEST, and NB resource type represents the global resource for library cache pin on table T1_LIBTEST.

```
Searches gv$ges_enqueue to identify locks associated with the resource_name.
-------------------
Object Details...APPS.T_LIBTEST
-------------------
Resource details...
-------------------
Resource name [0xd9bcbc52][0xb0a18d29],[LB][, Master 2
Resource name [0xd9bcbc52][0xb0a18d29],[NB][, Master 2
-------------------
Lock details...
-------------------
Res name [0xd9bcbc52][0xb0a18d29],[LB][, owner 1
...Transaction_id0 131479,Level KJUSERPR ,State GRANTED
Res name [0xd9bcbc52][0xb0a18d29],[NB][, owner 1
...Transaction_id0 131479,Level KJUSERPR ,State GRANTED
```

Troubleshooting Library Cache Lock Contention

It is crucial to understand how library cache locks and resources are globalized in a RAC database, and the preceding section provided a foundation. In this section, I will create a library cache lock and library cache pin contention to show a method to troubleshoot the contention.

I will use the script in Listing 11-11 that joins a table 1,024 times to create a library cache lock contention. From session 1, I will execute a script. While session 1 is executing the script, I will try to alter table t_libtest to add a column from session 2. A column cannot be added since the first session has not released the global lock yet, and session 2 will wait for the first session to complete.[9]

```
REM from session #1
@listing_11_10.sql

REM from session #2
Alter table t_libtest add (n3 number);
<.. this session wait..>
```

Let us inspect global locks and wait events. Session 8075 is waiting for library cache lock wait event. This wait event is a bit misleading, since the actual wait is for global lock.

```
SELECT sid, SUBSTR(event, 1, 28) event, state, seconds_in_wait wis
FROM v$session
WHERE state='WAITING'
AND event LIKE 'library cache%';

   SID EVENT                        STATE             WIS
------ ---------------------------- --------------- -----
  8075 library cache lock           WAITING           165
```

Output of ges_blocking_locks.sql script is printed in the following section. From the output, session 11296 is holding a lock on resource [0xad07b0f5][0x339db0c8],[LB] in KJUSERPR mode, while session with SID 8075 is waiting to lock the same resource in KJUSEREX mode. Session 8075 is requesting the lock in *exclusive* mode since the DDL statement on a table must invalidate dependent library cache locks.

```
SQL>@ges_blocking_locks.sql

INST_ID                    : 2
RESOURCE_NAME1             : [0xad07b0f5][0x339db0c8],[LB][
GRANT_LEVEL                : KJUSERPR
REQUEST_LEVEL              : KJUSERPR
STATE                      : GRANTED
SID                        : 11296
EVENT                      : Disk file operations I/O
SEC                        : 197
-------------------------------------------------------------
INST_ID                    : 2
RESOURCE_NAME1             : [0xad07b0f5][0x339db0c8],[LB][
GRANT_LEVEL                : KJUSERNL
REQUEST_LEVEL              : KJUSEREX
STATE                      : OPENING
SID                        : 8075
EVENT                      : library cache lock
SEC                        : 186
```

[9]Sandesh Rao points out that locking will happen in a single-instance database also, but the difference is that library cache locks and pins have global resources representing them, whereas in a single-instance database, this problem will be visible in x$kgllk and x$kglpn memory views only.

In RAC, it is hard to debug the library cache lock and library cache pin contention using traditional methods. It is easier to debug the contention by reviewing global locks. Script ges_blocking_locks.sql comes in handy to understand holders and waiters of GRD locks.

Enqueue Statistics

There are a few enqueue statistics maintained by Oracle database related to global lock activity. AWR report also calculates the average enqueue time using these statistics. A few lines from AWR report are given in the following output, showing that the average global enqueue time is 0.1 ms.

```
Global Cache and Enqueue Services—Workload Characteristics
------------------------------------------------------------
                  Avg global enqueue get time (ms):      0.1
```

Three statistics are used in the calculation of this average: "`global enqueue get time`," "`global enqueue get async`," and "`global enqueue gets sync`." The `global enqueue get time statistic` maintains the total time spent waiting for global locks; `global enqueue gets async` maintains the total number of async gets (mostly by background processes); and `global enqueue gets sync` maintains the total number of sync gets (mostly by foreground processes).

Sum of the values of `global enqueue get async` and `global enqueue get sync` is the total number of enqueue gets. Average global enqueue time is calculated by dividing `global enqueue get time` by total number of enqueue gets.

It is very important to realize that `Avg global enqueue get time is` an average at an instance level. So, you need to identify specific locks involved in the locking contention to troubleshoot elevated global enqueue get time. The value of `Avg global enqueue get time` is merely an indicator about locking efficiency. Higher values for the statistics imply that there was a locking contention during that AWR sample period.

Here is an example analysis from a production problem. As you can see, Avg global enqueue get time is very high.

```
Global Cache and Enqueue Services—Workload Characteristics
------------------------------------------------------------
                  Avg global enqueue get time (ms):    419.7
```

Reviewing the foreground wait events section, you will realize that there is no locking contention that can justify elevated global enqueue get time. But, in RAC, library cache locks are globalized as global locks, and so waits for the GRD locks are counted toward global enqueue get time statistics.

```
                                                    Avg
                                        %Time Total Wait    wait    Waits   % DB
Event                             Waits -outs   Time (s)    (ms)     /txn   time
-------------------------- ------------ ----- ---------- ------- -------- ------
library cache lock               5,446     0    456,860   83889      0.0  111.3
library cache: mutex X          11,668     0      5,483     470      0.1    1.3
latch: ges resource hash l       5,915     0      2,462     416      0.0     .6
...
```

v$wait_chains

View v$wait_chains is another useful view that can be used to identify locking contention quickly. This view retrieves rows from all instances, and so, there is no global counterpart for this view.

Essentially, v$wait_chains stores wait-for-graph as a chain and then identifies if there is any cycle in the chain. The usefulness of this view stems from the fact that locking contention is usually very complex in a high-end production environment. You are looking for just one blocker to terminate so that the application can continue processing normally. The following SQL statement queries v$wait_chains and prints the waits in a tree format to identify the blockers quickly.

```
col ses format a15
set lines 180
WITH blocked AS
      (SELECT * FROM
        (SELECT instance, sid, sess_serial#, blocker_instance,
          blocker_sid, blocker_sess_serial#, level lv,
          num_waiters, blocker_chain_id
         FROM v$wait_chains
          CONNECT BY PRIOR sid        = blocker_sid
          AND PRIOR sess_serial#      = blocker_sess_serial#
          AND PRIOR INSTANCE          = blocker_instance
          START WITH blocker_is_valid = 'FALSE'
        )
      WHERE num_waiters >0 OR blocker_sid   IS NOT NULL
      )
    SELECT instance,
      LPAD(' ', 2*(lv-1)) ||b.sid ses, b.sess_serial#,
      b.blocker_instance, b.blocker_sid, b.blocker_sess_serial#
    FROM blocked b
/
                                       Blocker Blocker    Blocker
  INSTANCE SES             Serial#   Instance     sid    serial#
---------- --------------- ------- ---------- ------- ----------
         1 10127             43405
         1   6374            27733          1   10127      43405
         1     1421          24393          1    6374      27733
```

The preceding output shows a wait-for-graph of 1421 → 6374 → 10127. So, we may have to terminate session 10127 to clear the locking contention.

Hanganalyze

Hanganalyze can also be used to identify chains. The hanganalyze command essentially goes through chain of waiters to identify if there is a process blocking all other sessions. In RAC, you need to take concurrent hanganalyze dumps using –g all option, as shown in the following.

```
SQL> oradebug setmypid
SQL> oradebug  -g all hanganalyze 10
Hang Analysis in /u01/app/product/rdbms/diag/cdb12/cdb12_diag_11019.trc
```

The DIAG process will generate trace file wait cycle information. The following trace file shows a sample of a DIAG trace file.

```
-------------------------------------------------------------------------------
Chain 1:
-------------------------------------------------------------------------------
    Oracle session identified by:
    {
                instance: 1 (prod.prod1)
                   os id: 26698
              process id: 7322, oracle@racdb1
              session id: 2066
    session serial #: 26535
```

Hanganalyze trace file is self-explanatory, and by following the chain of waits, you can identify the process blocking all other processes.

Deadlocks

A deadlock occurs if two or more transactions are waiting for resources locked by each other. In a single-instance database, sessions waiting for a lock will execute a small piece of code known as deadlock detection algorithm, to check if there is a deadlock. The deadlock detection algorithm checks if a session is repeated in a waiter-holder tree. If a session is seen again in the locking hierarchy, then it would indicate that there is a deadlock and current session aborts the statement to resolve the deadlock.

In a single-instance database, waiters and holders are visible to every session connected to the database. So, a foreground process can perform deadlock detection. In RAC, waiters and holders may be in different instances, and all necessary data for deadlock detection is not available to the foreground processes. So, LMD background process performs deadlock detection in a RAC database.

I will explain the deadlock detection with a simple test case. From two sessions connected to two different instances, I will update two rows in a table.

```
REM From session 1 connected to instance 1
UPDATE t1 SET n1=n1 WHERE n1=100;
1 row updated.
REM From session 2 connected to instance 2
UPDATE t1 SET n1=n1 WHERE n1=200;
1 row updated.
```

Next, I will update the row where n1=200 from session 1 and update the row with n1=100 from session 2. After the update, since these two sessions are waiting for each other, we induce conditions for deadlock.

```
REM From session 1
UPDATE t1 SET n1=n1 WHERE n1=200;
<..session waiting..>

REM From session 2
UPDATE t1 SET n1=n1 WHERE n1=100;
<.. session waiting..>
```

At this time, both sessions are waiting for each other, a classic behavior of a deadlock. Figure 11-2 shows the deadlock issue.

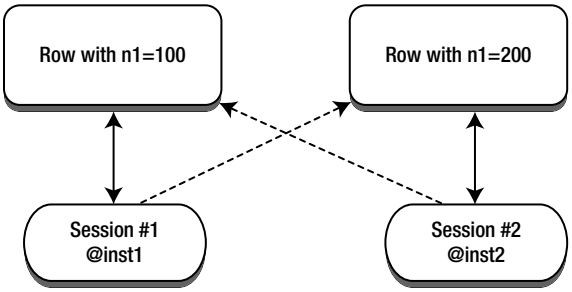


Figure 11-2. *Deadlock between two sessions*

The deadlock detection algorithm executed by LMD process will detect deadlocks in nearly 10 seconds. After 10 seconds of lock wait, a waiting session is chosen and the statement is rolled back.

```
update t1 set n1=n1 where n1=200
*
ERROR at line 1:
ORA-00060: deadlock detected while waiting for resource

Elapsed: 00:00:09.59
```

■ **Note** Deadlock detection is performed in ~10 seconds from Oracle Database version 11g onward. In version 10g, deadlock detection can take 60 seconds. Parameter _lm_dd_interval controls this behavior. Of course, there is no reason to adjust this parameter though. In single-instance database, deadlock detection routine is executed every 3 seconds, and therefore deadlock will be detected in 3 seconds.

In a single-instance database, a deadlock trace file is written by the foreground process, but in a RAC database as deadlocks are detected by LMD processes, you must review LMD trace file to understand the root cause of a deadlock.

LMD Trace File Analysis

Information written by the LMD processes to trace files follows a specific format. LMD process detecting deadlock writes resource and lock details in its own trace file. In addition, LMD processes running in other nodes are requested to dump details about holders and waiters of resources and locks involved in the deadlocks from their respective nodes. This coordinated dump by LMD processes is written to their own trace files, and so it is important to review LMD trace files from all instances at the same time to gain full insight into the deadlock issue.

LMD trace file dumps for deadlocks follow a general structure outlined in Figure 11-3. The first section prints resource details, queue details, and a summary of lock structures in the queue. Then, details about each of these lock structures are printed. If the lock structure is in local instance, then process such as program, user name, etc., are printed along with the lock details.

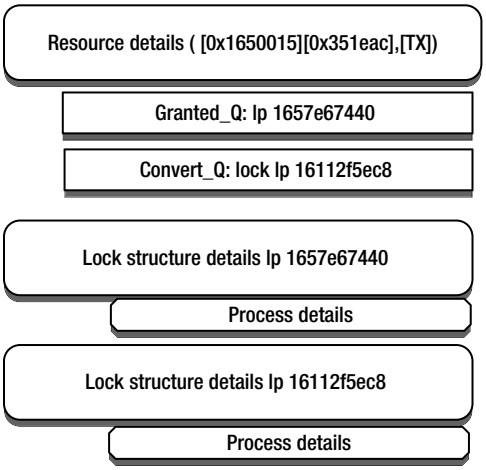


Figure 11-3. *LMD trace file structure*

Now, let me explain the details about each of these sessions. The following lines show the starting point for this resource dump in node 2 after a deadlock detection event. LMD process detecting the deadlock is starting to write deadlock information:

```
*** 2012-10-06 15:37:08.013

Global blockers dump start:---------------------------------
DUMP LOCAL BLOCKER/HOLDER: block level 5 res [0x1650015][0x351eac],[TX][ext 0x0,0x0]
```

Nearly at the same time, the following lines are written to LMD trace files in node 1, indicating that a dump request was received by the LMD process in node 1 to dump details about resource and lock structures.

```
*** 2012-10-06 15:37:08.018
ENQUEUE DUMP REQUEST: from 2 spnum 12 on [0x1650015][0x351eac],[TX][ext 0x0,0x0] for reason 3
DUMP LOCAL BLOCKER/HOLDER: block level 5 res [0x1650015][0x351eac],[TX][ext 0x0,0x0]
```

The first section of the dump file prints the resource details. In this case, the resource is a TX resource; as we discussed earlier in this chapter, row-level locks are implemented through TX resources. The next three lines show details about the local instance, directory instance, and master instance. The master instance is the resource master instance.

```
----------resource 17cfa16ca0----------------------
resname       : [0x1650015][0x351eac],[TX][ext 0x0,0x0]
hash mask     : x3
Local inst    : 1
dir_inst      : 1
master_inst   : 1
...
```

The following few lines show existing grants on that resource. `Count` line indicates number of locks in that mode. In this example, there is a NULL mode and an exclusive mode lock on this resource. Value block from the resource is also printed. Generally, value block is not useful for debugging deadlocks though.

```
grant_bits    : KJUSERNL KJUSEREX
grant mode    : KJUSERNL  KJUSERCR  KJUSERCW  KJUSERPR  KJUSERPW  KJUSEREX
count         : 1         0         0         0         0         1
val_state     : KJUSERVS_NOVALUE
valblk        : 0x000000010c279928000000000000000 .'(
...
```

Continuing the discussion, one lock has been granted and one lock is waiting as `Cvting_locks` is set to 1 and Granted_locks is also set to 1.

```
Granted_locks : 1
Cvting_locks  : 1
value_block:  00 00 00 01 0c 27 99 28 00 00 00 00 00 00 00 00
```

The following few lines show processes in the queue. The Granted_Q section prints the details about locks currently held on the resource. Possible PID 13787 is holding an exclusive mode lock on this resource. Convert_Q section prints the details about process waiting in the converting queue. Notice that there is no PID or XID information printed in this lock structure, as that information is not available in the local node. You will need to refer to the remote node LMD trace file to identify that information.

```
GRANTED_Q :
lp 1657e67440 gl KJUSEREX rp 17cfa16ca0 [0x1650015][0x351eac],[TX][ext 0x0,0x0]
master 1 gl owner 1753f447f8 possible pid 13787 xid 1007-0078-00001FF1 bast 0 ..
open opt KJUSERDEADLOCK

CONVERT_Q:
lp 16112f5ec8 gl KJUSERNL rl KJUSEREX rp 17cfa16ca0 [0x1650015][0x351eac],[TX]..
   master 1 owner 2  bast 1 rseq 43 mseq 0x40001 history 0xd497adaa
   convert opt KJUSERGETVALUE
```

LMD Trace File: Locks

The next section in the LMD trace file prints information about lock structures. Detail printed in this section is more complete if the lock is held by a local session. Again, by assembling data from all nodes of LMD trace files, you can get a complete picture about the deadlock issue.

The following section prints the data from enqueue structures. You can map lock pointers in the convert_q or grant_q in the resource details section to the lock details section. In this example, lock `1657e67440` is holding the lock and listed in grant_q section. This lock is also granted in exclusive mode as the grant_level is set to `KJUSEREX.` Process with PID 13787 is holding the lock in exclusive mode.

```
----------enqueue 1657e67440------------------------
lock version     : 185
Owner inst       : 1
grant_level      : KJUSEREX
req_level        : KJUSEREX
bast_level       : KJUSERNL
notify_func      : 0
...
possible pid     : 13787
xid              : 1007-0078-00001FF1
...
```

```
lock_state       : GRANTED
ast_flag         : 0x0
Open Options     : KJUSERDEADLOCK
...
```

Session-level details are also printed. Details such as program, username, application name, etc., will be useful in understanding the deadlock issues. Finally, SQL statement involved in the deadlock is also printed to debug the deadlock further.

```
user session for deadlock lock 0x1657e67440
  sid: 11303 ser: 58495 audsid: 4294967295 user: 0/SYS
    flags: (0x41) USR/- flags_idl: (0x1) BSY/-/-/-/-/-
    flags2: (0x40009) -/-/INC
  pid: 120 O/S info: user: oracle, term: UNKNOWN, ospid: 13787
    image: oracle@rac1 (TNS V1-V3)
client details:
O/S info: user: oraperf, term: pts/3, ospid: 13786
machine: wsqfinc1a program: sqlplus@wsqfinc1a (TNS V1-V3)
application name: sqlplus@wsqfinc1a (TNS V1-V3), hash value=321046474
current SQL:
  update t1 set n1=n1 where n1=200
```

In a nutshell, deadlock details are printed in LMD trace files. To better understand the root cause of deadlocks, assemble the sections from LMD trace files from all nodes at the time of deadlock. By reviewing details, you can identify the root cause of a deadlock.

Initially, LMD trace file details are overwhelming, since a huge amount of information is dumped to the trace file. You can use this chapter to understand the trace file contents and identify SQL statements and the processes suffering from deadlock. This approach should lead to a faster resolution of deadlocks.

Summary

Resources and locks are maintained in GRD by the GES layer of RAC code. This globalization of resources allows RAC to coordinate changes to the resources. By reviewing GES layer views, you can understand the root cause of the locking contention and resolve performance issues. In addition, the LMD process prints essential details in a trace file, and the analysis of LMD trace files should lead to quicker root cause analysis.

CHAPTER 12

Parallel Query in RAC

by Riyaj Shamsudeen

Parallel Execution (PX) is a powerful feature that makes it possible to utilize numerous processes to process millions, if not billions, of rows. In RAC, multiple nodes can be employed to execute a single SQL statement if the data set processed by a single SQL statement is huge.

In a single-instance database, PX server processes are allocated in the local instance. In a RAC database, PX server processes *can* be allocated in any or all active instances depending upon the configuration. For example, a PX query initiated in instance 1 can employ PX server processes from instances 1, 2, and 3 (in a three-node RAC cluster) to execute a SQL statement. The session initiating a PX acts as a query co-coordinator (QC), breaks the total work into individual units, and employs PX server processes to complete SQL statement execution. At the end of the execution, the QC process returns the final result set or the result of SQL execution to the calling program.

Overview

In most cases, parallel statement execution employs PX servers in a classic Producer/Consumer model for SQL execution. Producers collect and filter row pieces and distribute row pieces to the consumers, and consumers receive the row pieces and perform further processing, such as JOIN operation. Typically, the QC process waits for the PX servers to complete the work. It is also possible for QC to perform data processing depending upon the execution plan chosen by the optimizer.

The Producer/Consumer model is implemented as two PX server sets. One set of PX server processes acts as producers and the other set as consumers. For example, if you have specified a parallelism of 8, then it is probable that your query might get 16 PX servers for that SQL statement. Eight of them will be assigned to act as a producer and the other eight as a consumer. Essentially, a parallelism of 8 can employ 16 PX servers to execute a SQL statement.

Note There are special cases of PX execution that use just one set of PX server processes. For example, the SQL statement `select count (*) from table` will use just one set of PX server processes. Another example is partition-wise joins (discussed later in this chapter).

The Producer/Consumer concept can be explained with a JOIN operation. Listing 12-1 prints a PX plan. The tables PRODUCTS and SALES are joined using the HASH join technique and aggregated with a `group by` operation. Column TQ in the execution plan specifies the PX server set.

1. In the following execution plan, the `PRODUCTS` table is scanned by a set of PX server processes marked as Q1,00 in the TQ column. This PX server set acts as a producer and distributes row pieces to the second PX server set, Q1,01.

2. The SALES table is scanned by a set of PX server processes marked as Q1,01 in the TQ column. In addition to the scanning SALES table, this PX server set receives the row pieces of the PRODUCTS table sent by PX server set Q1,00. In this HASH join step, PX server set Q1,01 acts as a consumer.
3. After completing the HASH join operation, the Q1,01 PX server set processes redistribute the row pieces to the next set of PX server processes, marked as Q1,02 in the TQ column. PX server set Q1,02 performs a HASH GROUP BY operation. For the HASH GROUP BY operation, PX server set Q1,01 acts as a producer and the Q1,02 PX server set operates as a consumer.
4. PX server set Q1,02 completes grouping operations and sends the processed data to QC.

Listing 12-1. Parallel SQL Execution Plan

```
explain plan FOR
SELECT /*+ parallel (8) full (p) full (s) */
prod_name, SUM (QUANTITY_SOLD) FROM sales s , products p
WHERE s.prod_id = p.prod_id
group by prod_name HAVING SUM(quantity_sold)>100
/
Select * from table(dbms_xplan.display('','','BASIC +parallel'));

-------------------------------------------------------------------------------
| Id  | Operation                     | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |        |      |            |
|   1 |  PX COORDINATOR               |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)         | :TQ10002 |  Q1,02 | P->S | QC (RAND)  |
|   3 |    FILTER                     |          |  Q1,02 | PCWC |            |
|   4 |     HASH GROUP BY             |          |  Q1,02 | PCWP |            |
|   5 |      PX RECEIVE               |          |  Q1,02 | PCWP |            |
|   6 |       PX SEND HASH            | :TQ10001 |  Q1,01 | P->P | HASH       |
|   7 |        HASH GROUP BY          |          |  Q1,01 | PCWP |            |
|   8 |         HASH JOIN             |          |  Q1,01 | PCWP |            |
|   9 |          PX RECEIVE           |          |  Q1,01 | PCWP |            |
|  10 |           PX SEND BROADCAST   | :TQ10000 |  Q1,00 | P->P | BROADCAST  |
|  11 |            PX BLOCK ITERATOR  |          |  Q1,00 | PCWC |            |
|  12 |             TABLE ACCESS FULL| PRODUCTS |  Q1,00 | PCWP |            |
|  13 |          PX BLOCK ITERATOR    |          |  Q1,01 | PCWC |            |
|  14 |           TABLE ACCESS FULL   | SALES    |  Q1,01 | PCWP |            |
-------------------------------------------------------------------------------
```

The parallel query operation discussed earlier is shown in Figure 12-1. In the figure, PX servers Q1,00, Q1,01, and Q1,02 are drawn as nodes of a PX tree. The distribution between PX servers is represented as a connection between the nodes of the PX tree. The PX Distribution column prints the method of row distribution between PX server sets. In Listing 12-1, at step 10, PX distribution is set to BROADCAST, indicating that all rows of the PRODUCTS table are distributed from the Q1,00 PX server set to all PX server processes of the Q1,01 PX server set. (Broadcast mechanism can be chosen if the row source is smaller.) PX server processes of Q1,01 PX server set receive the PRODUCTS table and then join it to the SALES table to complete join processing. Notice that in step 6, the Distribution mechanism is HASH—a hashing algorithm applied to grouping columns—and rows are distributed to PX server processes of the Q1,02 PX server set.

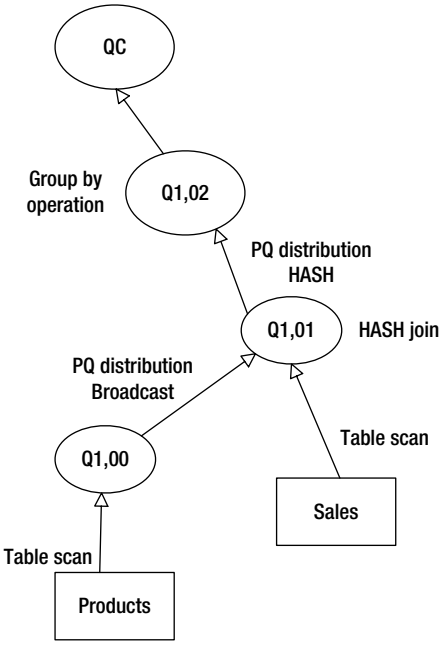


Figure 12-1. *PX operational details*

With BROADCAST, all rows are distributed from one PX server set to another PX server set. With the HASH distribution mechanism, a subset of rows is distributed from one PX server set to another PX server set, that is, each PX server process receives a subset of row pieces. In the preceding example, each PX server process of Q1,02 will receive 1/8th of the rows produced in step 6 of the execution plan.

You must understand that there can be at most two sets of PX processes active at any time during a SQL execution, even though the execution plan shows more than two PX server sets. PX server processes can be reused to perform another operation in a different part of the execution plan. In this example, PX processes associated with PX server set Q1,00 are reassigned as PX server set Q1,02 in step 3.

Listing 12-2 shows the PX execution in play for the query in Listing 12-1. Parallel statement execution with a degree of parallelism (DOP) of 8 employed 16 PX server processes. Notice that the `server_set` column is showing the PX server set. Essentially, there are two sets of PX server processes executing this statement.

Listing 12-2. PX Server Processes in Single Instance: Script pxslaves.sql

```
col username for a12
col "QC SID" for A6
col SID for A6
col "QC/Slave" for A10
col "Requested DOP" for 9999
col "Actual DOP" for 9999
col server_set for  A10
set pages 100
select
  decode(px.qcinst_id,NULL,username,
```

```
        ' - '||lower(substr(s.program,length(s.program)-4,4) ) ) "Username",
  decode(px.qcinst_id,NULL, 'QC', '(Slave)') "QC/Slave" ,
  to_char( px.server_set) server_set,
  to_char(s.sid) "SID",
  decode(px.qcinst_id, NULL ,to_char(s.sid) ,px.qcsid) "QC SID",
  px.req_degree "Requested DOP",
  px.degree "Actual DOP"
from
  v$px_session px,
  v$session s
where
  px.sid=s.sid (+)
 and
  px.serial#=s.serial#
order by 5 , 1 desc
/
Username     QC/Slave   ServerSet  SID    QC SID Requested DOP Actual DOP
------------ ---------- ---------- ------ ------ ------------- ----------
RS           QC                    12376  12376
 - p015      (Slave)    2          10756  12376             8          8
 - p014      (Slave)    2          11295  12376             8          8
 - p013      (Slave)    2          10211  12376             8          8
 - p012      (Slave)    2          9699   12376             8          8
 - p011      (Slave)    2          9147   12376             8          8
 - p010      (Slave)    2          7535   12376             8          8
 - p009      (Slave)    2          8609   12376             8          8
 - p008      (Slave)    2          8071   12376             8          8
 - p007      (Slave)    1          6474   12376             8          8
 - p006      (Slave)    1          7012   12376             8          8
 - p005      (Slave)    1          5403   12376             8          8
 - p004      (Slave)    1          5928   12376             8          8
 - p003      (Slave)    1          4847   12376             8          8
 - p002      (Slave)    1          4325   12376             8          8
 - p001      (Slave)    1          3787   12376             8          8
 - p000      (Slave)    1          2716   12376             8          8
```

In Figure 12-2, a SQL execution with a requested DOP set to 4 is shown. There are four processes acting as producers and four acting as consumers, and a QC receives the row pieces from consumers. This data flow operation shows that row pieces are distributed from one PX server set to another PX server set.

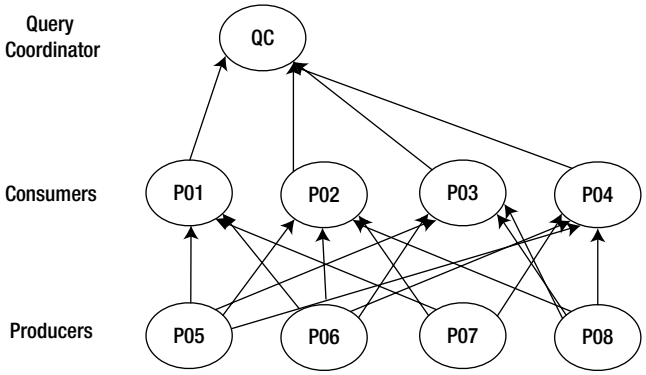


Figure 12-2. *PX execution in single instance*

Also, in a single-instance database, the PX server processes communicate by exchanging PX buffers.

PX Execution in RAC

PX execution in RAC follows the same principle as a single-instance PX execution, except that PX server processes can be allocated from any and all active instances.[1] Allocation of PX server processes from multiple instances to execute a SQL statement is known as inter-instance parallelism, and these PX server processes exchange message buffers between them through private networks. Allocation of PX server processes within the local instance is termed as intra-instance parallelism.

As we saw in Listing 12-1, row pieces (message buffers in network terminology) are transmitted between PX server sets using the PX Distribution method of HASH and BROADCAST (there are few other Distribution methods, namely, *Partition*, *none,* etc.). In RAC, row pieces are distributed through the private network. The optimal placement of PX server processes is of paramount importance to reduce private network traffic generated by PX workload.

RAC databases with higher PX workloads generally have higher bandwidth requirements in the private interconnect. Oracle Database versions 11g/12c are optimized to minimize private network traffic with optimal localization of PX server processes.

Figure 12-3 shows a PX server allocation scheme in RAC. In this figure, PX server processes are allocated in both nodes 1 and 2 to perform parallel execution. Two PX servers on each node act as consumers, and two PX servers on each node act as producers. Producer PX servers distribute row pieces to consumer PX servers through the private interconnect. In this example, to minimize interconnect traffic, both producers and their consumers should be allocated from the same node.

[1]I will discuss methods to control PX server allocation to an instance or a subset of instance later in this chapter.

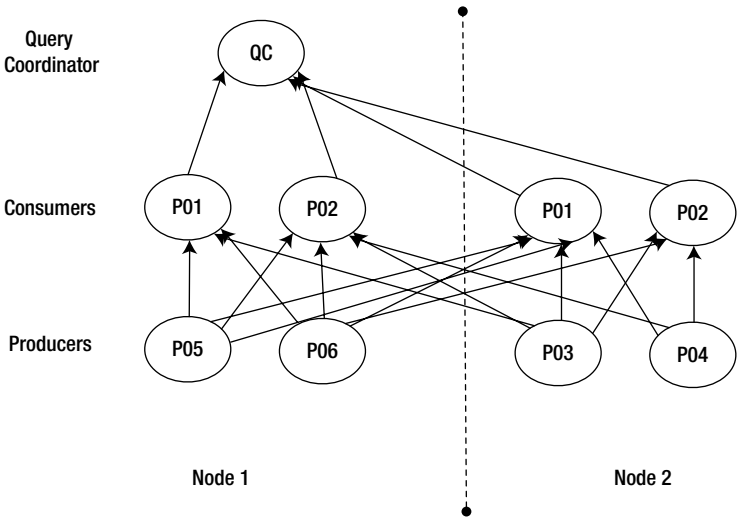


Figure 12-3. *PX execution in RAC*

In Figure 12-3, distribution of row pieces between the PX server processes is shown as the connection between the PX processes. PQ distribution in an inter-instance parallel operation flows through the private network (cluster interconnect).

Placement of PX Servers

Placement of PX servers can be controlled using a few techniques in RAC. Traditionally, in earlier versions of Oracle RAC, a combination of instance_group and parallel_instance_group parameters was employed to control PX server placement, but from Oracle Database 11g onward, it is preferable to use services to control the placement of PX servers.

Instance_group and Parallel_instance_group

The parameter `instance_group` specifies the membership of an instance to an instance group. PX servers are allocated from an instance group specified by the `parallel_instance_group` parameter. Adjustment to these two parameters can be utilized to control the placement of PX servers.

The following discussion uses a three-node cluster configuration with instances named `orcl1`, `orcl2`, and `orcl3`. In the following parameter setup, the `instance_groups` parameter is specified at the instance level using the syntax <instance>.<parameter>. The value of the parameter `orcl1.instance_groups` is set to `'po','scm'` and specifies that the orcl1 instance is a member of instance groups `po` and `scm`. Similarly, the `orcl2` instance is a member of the `po` and `scm` instance groups, and the `orcl3` instance is a member of the `finance` and `scm` instance groups. Hence, all three instances are members of the scm instance group, instances 1 and 2 are members of the po instance group, and instance 3 is the sole member of the finance instance group.

```
orcl1.instance_groups='po','scm'
orcl2.instance_groups='po','scm'
orcl3.instance_groups='finance','scm'
```

With the parameter setup just discussed, we can alter the parallel_instance_group parameter to control PX server allocation. In the following example, the value of parameter parallel_instance_group is set to `scm` in `orcl1` instance.

PX execution initiated from orcl1 *can* allocate PX servers in all instances, since all three instances are members of the scm instance group.

```
orcl1.parallel_instance_group= 'scm'
```

If we set the parallel_instance_group parameter to po in orcl1 instance, then PX executions initiated from the orcl1 instance can allocate PX servers from both orcl1 and orcl2 instances.

```
orcl1.parallel_instance_group= 'po'
```

If the parameter parallel_instance_group is set to finance in orcl3, then the PX execution can allocate PX servers only from the orcl3 instance, as only the orcl3 instance is a member of the finance instance group.

```
Orcl3.parallel_instance_group= 'finance'
```

If you set the parameter parallel_instance_group to finance in the orcl2 instance, then the PX execution initiated from the orcl2 instance will spawn PX servers in the orcl3 instance, as orcl3 is the sole member of the finance instance group.

```
Orcl2.parallel_instance_group= 'finance'
```

If the specified parallel_instance_group parameter value has no members, then the parallel execution will be disabled completely. For example, a parameter value of ar will disable parallel exectuion, since there are no instances in the ar instance group.

```
*.parallel_instance_group= 'ar'
```

You can alter the value of the parallel_instance_group parameter dynamically at either the session level or the system level. For example, the following alter session statement enables PX server allocation in orcl1 and orcl2 instances in the session. You can override instance- or database-level setup with session-level changes.

```
Alter session set parallel_instance_group= 'po';
```

The parameter instance_group cannot be altered dynamically (since release 11g), and instance restarts would be required to alter the instance_group parameter. There is no explicit method to create an instance group.

As of version 11g, the optimizer considers the parallelism setup before choosing a PX plan. If parallelism is disabled, in the session or globally, the optimizer will not choose a PX plan. Prior to version 11g, the optimizer could choose a PX plan even if the parallelism was disabled. At run time, the execution engine would execute the chosen PX plan serially, leading to worse performance. From Database version 11.2.0.2 onward, the cost-based optimizer is also aware of the parallelism setup. If the parallelism is disabled in the session, through either session-level parameter setup or global-level parameter setup, then the optimizer does not choose a parallel plan at all. For example, I modified the parallel_instance_group to dummy and executed the following statement. Cost-based optimizer 10053 event trace indicates that optimizer used a parallel degree of 1, even though the hint specifies a DOP of 8. Also notice that the cost of parallel plan (resp) is same as the cost of serial plan (Resc), indicating that the optimizer will not choose a PX plan.

```
alter session set events '10053 trace name context forever, level 1';
REM parallel_instance_group value is non-existent.
alter session set parallel_instance_group='dummy';
```

```
select /*+ use_hash (h l ) parallel (h 8) parallel (l 8) full(h) full(l) */
count(*)  from
oe_order_headers_all h, oe_order_lines_all l
where h.header_id = l.header_id and
h.open_flag='N'

/
Trace file due to 10053 event:

Best:: AccessPath: TableScan
      Cost: 197226.11  Degree: 1  Resp: 197226.11  Card: 10992402.90  Bytes: 0
```

After altering parallel_instance_group to the correct value, reparsing the same statement shows that the optimizer used a parallel degree of 8. Also, notice that the parallel cost is lower than the earlier serial cost of 197226.11.

```
Best:: AccessPath: TableScan
         Cost: 61202.86  Degree: 8  Resp: 61202.86  Card: 10992402.90  Bytes: 0
```

Even though the parameters instance_group and parallel_instance_group can be used to control PX server placement, use of these parameter combinations is deprecated from Oracle Database version 11.2. Now, it is recommended to use services to control placement of PX servers. Also, in 11.2, the incorrect value for parallel_instance_group parameter can lead to an excessive number of child cursor problems due to bugs such as 7352775. There are also a few unpublished bugs where the use of these two parameters leads to unshareable child cursors. So, it is preferable to use services to control PX server placement.

Services

Services are the preferred approach to control PX server placement. By default, PX servers are allocated in instances where the current service is active ("current service" refers to the service that the session initiating PX execution connected). For example, if you connect to po service and if po service is available in ORCL1 and ORCL2 instances, then PX servers *can* be allocated from both ORCL1 and ORCL2 instances.

The following output shows the service configuration in a database named ORCL. In this configuration, service po is active in ORCL1, ORCL2 instances, service scm is active in all three instances, and finance is active only in the ORCL3 instance. If you connect to the po service and initiate a parallel statement execution, then PX servers can be allocated from the ORCL1 and ORCL2 instances. Similarly, if you connect to finance service and initiate a parallel statement execution, then PX servers can be allocated from the ORCL3 instance only.

```
$ srvctl status service -d ORCL
Service po is running on instance(s) ORCL1,ORCL2
Service scm is running on instance(s) ORCL1,ORCL2,ORCL3
Service finance is running on instance(s) ORCL3
```

You can also use a combination of both services and the Parallel_instance_group parameter to control PX server placement. If you set the parallel_instance_group parameter to a value of service, then PX servers can be allocated from the instances that service is running. For example, after connecting to the finance service (the connection will go to the ORCL3 instance), you can alter the parallel_instance_group parameter at the session level to po service. Now, PX executions from the current session can allocate PX servers in the ORCL1 and ORCL2 instances, since po service is active in the ORCL1 and ORCL2 instances, even though PX is triggered from the ORCL3 instance. While this strategy provides the ability to control your PX server allocation, it is not a recommended approach.

```
$ sqlplus rs/temp123@//orcl-scan:1521/finance
...
Alter session set parallel_instance_group=po;
```

PX server placement utilizing the service configuration is elegant, since after a node/instance failure, services automatically fail over to the surviving nodes. You should consider instance failure scenarios while setting up services and make active/passive configuration for services too. Further, service configuration can be modified dynamically. In contrast, if you use instance_group and parallel_instance_group parameters, then during instance failure, you may have to modify the instance_group parameter, which might require another surviving instance restart.

Parallel_force_local

The parallel_force_local parameter can be used to localize the PX server allocation. If the parallel_force_local parameter is set to TRUE, then PX servers are allocated in the same instance that the PX query was initiated. For example, if you connect to the ORCL1 instance and initiate a PX execution, then all PX servers will be allocated from the ORCL1 instance. This parameter can be modified at the session level or at the global level the using alter session or alter system commands.

The problem with controlling the PX server placement with this parameter is that if all PX servers in the local instance are exhausted, then the query may run serially or with reduced parallelism. Thus, it is a preferable approach to use services to control the allocation. Also, this parameter doesn't work well with the parallel_statement_queuing parameter, as discussed later in this chapter.[2]

Measuring PX Traffic

Since Oracle Database version 11g, private network traffic statistics have been captured at the workload level and externalized in the x$ksxpclient fixed table.

In Listing 12-3, query shows that the data from the x$ksxpclient fixed table is connected to the PROD1 instance. From the output, we can calculate bytes sent and received for each workload. The PROD1 instance sent 32TB and received 49TB for the buffer cache (cache fusion) workload. Similarly, the PROD1 instance sent 8.7TB and received 5.2TB for the DLM (Distributed Lock Manager) workload. Instance PROD1 also sent 1.8TB and received 2.1TB for the PX workload. In the following output, PX workload is indicated by the line with name as ipq.

Listing 12-3. x$ksxpclient

```
set pages 100
SYS@PROD1> SELECT name,
           TRUNC (bytes_sent/1024/1024,2) bytes_sent_MB,
           TRUNC(bytes_rcv  /1024/1024,2) bytes_recv_MB
FROM x$ksxpclient;

NAME      BYTES_SENT BYTES_RECV
--------- ---------- ----------
cache       32165225 49870288.1
dlm       8754651.32 5217822.01
ipq        1791954.4 2121334.66
ksxr         5069.85    4572.55
```

[2]Arup Nanda points out that, in the Exadata platform, by default, the parallel_force_local parameter is set to TRUE. It is interesting to realize that even with the highly efficient Infiniband fabric-based interconnect, this parameter is set to TRUE. However, choosing a value for this parameter might require knowledge about your application workload and database server architecture.

```
cgs           294.73     161.94
osmcache           0          0
streams          .03       4.02
diag         28846.1  103344.72
ksv                0          0
ping         5050.59    5050.58
internal           0          0
```

Of course, the output in Listing 12-3 shows the traffic breakdown from the start of the PROD1 instance. So, output from x$ksxpclient can be misleading if we need to measure interconnect network traffic for a constrained time window. AWR captures the data from x$ksxpclient into AWR tables and externalizes in dba_hist_ic_client_stats view. Listing 12-4 queries dba_hist_ic_client_stats and prints the network traffic for the PX workload. By adding value from all three instances for bytes_received (or bytes_sent), we can calculate total bytes transmitted for the PX workload. In this example, on 11/11/12 between 4:00 and 4:30 p.m., this database generated 82MB of network traffic for the PX workload.

Listing 12-4. dba_hist_ic_client_stats

```
SELECT TO_CHAR(snaps.begin_interval_time, 'DD-MON-YYYY HH24:MI') begin_time,
  snaps.instance_number,
  hist.name,
  TRUNC((hist.bytes_received - lag (hist.bytes_received) OVER (
        PARTITION BY hist.name, startup_time, hist.instance_number
        ORDER BY begin_interval_time, startup_time,
          hist.instance_number, hist.name))/1048576,2) Mbyte_rcvd,
  TRUNC((hist.bytes_sent     - lag (hist.bytes_sent) OVER (
        PARTITION BY hist.name, startup_time, hist.instance_number
        ORDER BY begin_interval_time, startup_time,
          hist.instance_number, hist.name))/1048576,2) Mbyte_sent
FROM dba_hist_ic_client_stats hist,
  DBA_HIST_SNAPSHOT snaps
WHERE snaps.snap_id           = hist.snap_id
AND snaps.instance_number     = hist.instance_number
AND snaps.begin_interval_time > sysdate-1
AND hist.name                 ='ipq'
ORDER BY snaps.snap_id, instance_number;

BEGIN_TIME               Inst NAME      MBYTE_RCVD MBYTE_SENT
------------------------ ---- --------- ---------- ----------
...
11-NOV-2012 16:00           1 ipq            31.23       33.7
                            2 ipq            28.31      35.01
                            3 ipq             42.9      33.72
11-NOV-2012 16:30           1 ipq            82.19      44.41
                            2 ipq            32.61      66.86
                            3 ipq            63.57      67.06
11-NOV-2012 17:00           1 ipq            32.89      32.33
                            2 ipq            14.28      36.99
                            3 ipq            55.45      33.32
```

This metric is also printed in an AWR report in the `interconnect throughput by client` section. You can see that interconnect traffic for PX workload is very minimal in this database.

```
Interconnect Throughput by Client      DB/Inst: PROD/PROD1  Snaps: 52265-52313
-> Throughput of interconnect usage by major consumers
-> All throughput numbers are megabytes per second
                         Send     Receive
Used By           Mbytes/sec  Mbytes/sec
---------------- ----------- -----------
Global Cache           15.19       22.93
Parallel Query           .05         .07
DB Locks                4.11        2.46
DB Streams               .00         .00
Other                    .00         .00
```

PX and Cache Fusion

PX execution generally reads the block directly into the Program Global Area (PGA) of the PX server process, bypassing buffer cache. Since the buffer cache is not used, cache fusion locks are not required for the blocks read using the direct path read method.

■ **Note** Not all PX executions will bypass buffer cache. For example, a PX plan can use nested loops join (each PX server performing its own nested loops join for a subset of rows), and so PX execution using nested loops join might use the buffer cache. Also, the new feature **in memory parallelism** in Oracle Database release 11.2 also enables the use of the buffer cache for PX executions.

The following lines are printed from a SQL trace file after initiating a parallel query execution. You can see that there are no global cache event waits for the direct reads. The direct path method reads the blocks directly into the PGA of PX server processes. Notice that 128 blocks are read from the disk to PGA directly without incurring a single global cache–related wait event. For massive segment scans, direct path reads are very efficient and reduce the global cache overhead associated with a traditional buffered read. If 128 blocks had to be buffered in the buffer cache, then there would have been numerous waits for global cache events.

```
select /*+ full(a) parallel (4) */ count(*) from apps.mtl_material_transactions a
...
nam='direct path read' ela= 1676 file number=845 first dba=401920 block cnt=128 obj#=37871
nam='direct path read' ela= 37423 file number=845 first dba=402048 block cnt=128 obj#=37871
nam='direct path read' ela= 26573 file number=845 first dba=402304 block cnt=128 obj#=37871
```

However, PX execution using the direct path reads method triggers an object-level checkpoint in all instances at the start of PX execution. Excess PX execution can trigger numerous object-level checkpoints, causing elevated write activity. Further, excessive object-level checkpoints with a combination of huge buffer cache and numerous instances (eight-plus instances) can lead to continuous checkpoints in all instances, leading in turn to slowdown for the query itself. So, you should consider the effect of the checkpoint while designing SQL statements to scan smaller tables using parallel execution.

```
nam='KJC: Wait for msg sends to complete' ela= 1166 msg=61225879752 ...
nam='enq: KO - fast object checkpoint' ela= 1330 name|mode=1263468550 ...
nam='enq: KO - fast object checkpoint' ela= 722 name|mode=1263468545 ...
```

Note that it is still possible for parallel execution to suffer from cache fusion waits for undo, undo header, and segment header blocks. For example, if a block has an uncommitted transaction, then PX server processes will access undo blocks/undo header blocks to construct a Consistent Read (CR) version of the block, which would require global cache locks.

PEMS

In an inter-instance PX execution, PX servers transmit buffers through the private interconnect and the size of transmission buffer is controlled by the parameter `parallel_execution_message_size` (PEMS). Prior to 11g, the default value of this parameter was about 2KB. With smaller PEMS, data transmission between the PX server processes is inefficient due to chattier transmission.

From Oracle Database version 11.2 onward, if the compatibility parameter is set to 11.2.0 or higher, then PEMS defaults to a value of 16K. If your application uses the PX execution feature heavily, then you must verify that PEMS is set to a value of at least 16K.

In PX-intensive environments, it is also advisable to use Jumbo Frames so that chatty network traffic can be reduced. In my experience, the combination of Jumbo Frames and PEMS set to 16K increased throughput of PX workload in a production database dramatically.

Parallelism Features and RAC

Oracle Database version 11.2 introduced three important parallelism features, and version 12c enhances these features. The feature **In-memory parallelism** introduces buffered reads for PX workload. The **AutoDOP** (automatic DOP) feature introduces automatic PX plan, and with the **PX Statement Queuing** feature, PX is delayed until sufficient PX server processes are available.

All three features are controlled by a master switch parameter, namely, `parallel_degree_policy`.

1. If you set the parallel_degree_policy parameter to a value of AUTO, then all three features are enabled.
2. If you set the parallel_degree_policy parameter to a value of LIMITED, then that parameter value disables PX statement queuing and in-memory parallelism. However, for statements accessing objects decorated with a parallel degree of DEFAULT, the AutoDOP feature is enabled.
3. If you set the parallel_degree_policy parameter to a value of MANUAL, then these three new features are disabled.
4. Version 12c introduces AUTO_FEEDBACK, a new value which is a combination of AUTO and FEEDBACK. All three parallel features discussed earlier are enabled with the AUTO value. In addition, a FEEDBACK mechanism to the cursor in memory is used to mark a statement to be reparsed with a different DOP. Two types of feedback mechanisms are possible: statement-level time feedback and operator-level time feedback. After a parallel statement execution, if the calculated DOP is significantly[3] different from actual DOP, then the statement is signaled to be reparsed with actual DOP during the execution. This feature is more granular than just the statement-level feedback and tracks the DOP at the operation level. If the actual operator-level DOP is different from estimated DOP, then the operator-level execution DOP is used as a feedback mechanism while reparsing the SQL statements during subsequent execution of the cursor.

I will introduce these features and discuss how they can be effectively utilized in a RAC environment.

[3]Significantly different means that actual DOP is lower than 0.5 * estimated DOP or higher than 2 * estimated DOP.

In-Memory Parallelism

Traditionally, PX servers read blocks directly from the disk to PGA, bypassing buffer cache. In a few cases, it may be better to buffer the blocks in the database buffer cache so that subsequent PX server processes can reuse the buffers. This feature is pertinent in machines with huge swaths of memory allocated for SGA. Of course, buffering in SGA requires global cache locks, which might induce global cache event waits for the PX servers reading the block from disk.

From Oracle Database version 11.2 onward, if the parallel_degree_policy parameter is set to auto, then the in-memory parallelism feature is enabled. Objects are chosen to be cached in the buffer cache if the object is not too small or too big. If the segment is too small, then that segment can be cached in PGA itself; if the segment is too big, then buffering will overrun the buffer cache. Thus, this feature cleverly chooses segments of the right size.
The advantage of this feature is that subsequent processes accessing the same table do not need to repeat the work of reading the blocks from disk.

The algorithm uses the following directives to determine if the objects can be buffered or not:

1. Size of the object (must fit in the buffer cache of one instance; the parameter _parallel_cluster_cache_pct limits the maximum percentage of buffer cache that can be used for affinity and defaults to 80%).
2. Object access frequency.
3. Object change frequency.

In RAC, object fragments are affinitized among active instances. For example, if there are three instances in a PX execution, then approximately 1/3rd of the object is affinitized to each instance. Affinitization is performed on a per-extent basis for a non-partitioned table; 1/3rd of extents are affinitized to each node in a three-node cluster. If the table is hash partitioned, then the object is affinitized per partition basis. PX server processes are also affinitized such a way that PX servers access extents/partitions in the local buffer cache. For example, if partition P1 is affinitized to the ORCL2 instance, then PX processes are allocated in the ORCL2 instance if the execution plan requires access to partition P1. Affinitization is designed to reduce cache fusion network traffic.

This feature can be enabled at the session level also:

```
alter session set "_parallel_cluster_cache_policy"=cached;
```

This feature can be disabled in session level using the following statement. Of course, it is always better to check with Oracle Support before using underscore parameters in production code. Also, instead of hard-coding these SQL statements in production code, wrap this code in a PL/SQL procedure and call the procedure from application code. This enables you to change the code in the database without altering the application.

```
alter session set "_parallel_cluster_cache_policy"=adaptive;
```

The following SQL statistics show a comparison between parallel execution of a statement with caching enabled and disabled. The comparison shown here is for the second execution of the statement for each mode in a production database. If the caching is enabled, subsequent executions of the statement perform almost no disk I/O and reuse cached buffers. If the caching is disabled, then each execution performs direct I/O to read blocks from the disk. In the following example, SQL statement performed 232K reads with an increased elapsed time of 81 seconds, compared to 18 seconds when caching was enabled.

```
SQL_ID         EXEC CPU_TIME ELA_TIME BGETS    DRDS     ROWS
-------------  ---- -------- -------- -------- -------- -------
gj1p0kqyxvqas     1    17.37    18.66   455257        0     809 <- with caching enabled
gj1p0kqyxvqas     1    17.09    81.42   237443   232779     809 <- without caching
```

You need to understand that the effect of this feature depends upon the activity in the database buffer cache. If the buffers are not accessed frequently, then the buffers will be flushed out of the buffer cache, and so PX servers may be doing more work with no future benefit.[4] On the Exadata platform, this feature is probably not useful, as the smart scan feature requires direct path reads.

As the tables are buffered, there is a possibility that this feature can increase interconnect network traffic inadvertently. So, if you are planning to enable in-memory parallelism, verify that interconnect hardware is properly sized to support a possible increase in private network traffic.

AutoDOP

AutoDOP is a new feature available from Oracle Database version 11.2. Prior to version 11.2, the optimizer chose PX only if the objects had a DOP of greater than 1 or if there were hints in the SQL statement enabling parallelism. From 11.2 onward, parallelism is considered if the estimated run time of a SQL statement exceeds a threshold. The following algorithm outlines the logic of the AutoDOP feature.

```
Estimate total run time of a SQL statement for serial execution.
If estimated_run_time > 10 seconds threshold then
Re-parse the statement with a default DOP and determine the cost.
If the estimated cost is lower than the estimated cost for serial execution, then
               Execute the statement with Default DOP.
End;
Else
Execute the statement serially.
End;
```

The AutoDOP feature can be enabled by setting parallel_degree_policy to `auto` or `limited` (11.2) or `auto_feedback` (12c). If the parallel_degree_policy is set to limited, then objects with parallelism must be accessed in the SQL statement for the AutoDOP to be enabled. The following few lines are printed from event 10053 trace file. These lines show that AutoDOP was disabled since the parallel_degree_policy parameter is set to a value of limited and none of the objects accessed have a parallel degree decorated.

```
**************************
Automatic degree of parallelism (ADOP)
**************************
Automatic degree of parallelism is disabled: limited mode but no parallel objects referenced.
```

If you set the parallel_degree_policy parameter to auto, then the AutoDOP feature is enabled, but I/O must be calibrated for AutoDOP to be enabled.

```
**************************
Automatic degree of parallelism (ADOP)
**************************
Automatic degree of parallelism is enabled for this statement in auto mode.
kkopqSetDopReason: Reason why we chose this DOP is: IO calibrate statistics are missing.
```

[4]More work here refers to additional tasks of maintaining cache fusion lock structures to read buffers into buffer cache.

Enabling I/O calibration using dbms_resource_manager package will enable the AutoDOP feature. Calibration results are visible in v$io_calibration_status view. Note that there is no documented method to remove the calibration,[5] and so this change must be tested in a non-production database before calibrating I/O in production.

To calibrate I/O, you can choose any instance of a cluster and trigger I/O calibration. The database engine uses just one instance to calibrate I/O, but since I/O resource is common to all nodes, performing calibration in one node is good enough.

```
SET SERVEROUTPUT ON
DECLARE
  l_lat  INTEGER;
  l_iops INTEGER;
  l_mbps INTEGER;
BEGIN
  -- Calibrate I/O and print statistics
  DBMS_RESOURCE_MANAGER.CALIBRATE_IO (20, 20, l_iops, l_mbps, l_lat);
  DBMS_OUTPUT.PUT_LINE ('max_iops = ' || l_iops);
  DBMS_OUTPUT.PUT_LINE ('latency = ' || l_lat);
  DBMS_OUTPUT.PUT_LINE ('max_mbps = ' || l_mbps);
END;
/ [6]
```

If the estimated run time exceeds the value of the parameter parallel_min_time_threshold, then the statement is reparsed with a higher DOP. The parameter value of parallel_min_time_threshold defaults to a value of AUTO, which indicates a threshold of 10 seconds. If you prefer more queries to be considered for parallelism, then you can decrease the value of the parallel_min_time_threshold parameter to less than 10 seconds.

After the completion of I/O calibration, regenerating event 10053 trace file for the same SQL statement shows that AutoDOP was enabled for this statement execution. The optimizer uses calibrated I/O statistics to convert the cost to time using I/O calibration statistics. In this example, the estimated time is 38 seconds, exceeding parallel_min_time_threshold parameter, and so optimizer considers AutoDOP.

```
kkeCostToTime: using io calibrate stats
 maxmbps=100(MB/s) maxpmbps=11(MB/s)
 block_size=8192 mb_io_count=4 mb_io_size=30247 (bytes)
 tot_io_size=422(MB) time=38387(ms)
AutoDOP: Table/Index(#76100) access (scan) cost=14638.09 estTime=38386.60 unit=10000.00
   dop=3 -> maxdop=3
Best:: AccessPath: TableScan
Cost: 14638.09 Degree: 1 Resp: 14638.09 Card: 159987.00 Bytes: 0
```

Following lines from the trace file shows that optimizer is triggering a reparse of the SQL statement with a statement-level parallelism of 3.

```
AUTODOP PLANS EVALUATION
****************************************
Compilation completed with DOP: 1.
   Cost_io: 59742.00 Cost_cpu: 1227068251.35
   Card: 318995.00 Bytes: 1279807940.00
```

[5]It is possible to manually delete rows from an underlying table. Please contact Oracle Support to remove I/O calibration if needed.
[6]Optimizer estimated cost is roughly a number normalized to single block reads.

```
   Cost: 59810.86 Est_time: 156847ms
Serial plan is expensive enough to be a candidate for parallelism (59811)
Signal reparse with DOP 3.
****************************
Number of Compilations tried: 1
```

Let us review a few important parameters controlling the AutoDOP feature. Parameter `parallel_degree_limit` governs the maximum DOP that can be chosen for a SQL statement. By default, this parameter is set to CPU, implying default DOP as the maximum DOP for AutoDOP execution. Default DOP is calculated by multiplying the total number of CPUs in the cluster (Sum of cpu_count from all active instances) by the parallel_threads_per_cpu (Default=2) parameter. So, if there are 32 cores each in a three-node cluster, then the value of parallel_degree_limit is calculated as 32 * 3 * 2 (parallel_threads_per_cpu=2), equaling a value of 192. So, 192 PX servers *can* be used for a SQL statement execution.

■ **Note** The parameter parallel_degree_limit can be set to I/O or an integer. If you set the parameter value to I/O, then I/O bandwidth derived from dbms_resource_manager.calibrate_IO procedure call is used to derive upper bounds for default DOP. You can also set the parallel_degree_limit parameter to a number specifying the maximum default parallelism to be used for AutoDOP calculations.

The parameter `parallel_max_servers` is a hard upper bound determining the maximum number of PX servers available in any instance, and the parallel_max_servers parameter overrides the parallel_degree_limit parameter if the parallel_max_servers parameter is set to a value lower than the default DOP.

While parallel_degree_limit controls the upper bound for parallelism, the actual parallelism used in a query (calculated DOP or actual DOP) is less than the default DOP. The exact algorithm is not documented for the actual DOP calculation; however, from event 10053 trace files, we can infer that the optimizer is recosting the statement with different DOPs to find the optimal DOP for a SQL execution.

There are a few dangers with AutoDOP feature in a production database.

1. If many concurrent queries are parallelized inadvertently, then those queries can deplete I/O and CPU resources in the database server, causing performance issues with the online application.
2. In addition, inadvertent parallel queries can consume all parallel servers, starving critical parallel queries of parallel servers.
3. Also, in RAC, if many queries are parallelized, then there is a possibility that numerous queries might perform inter-instance PX, leading to worse interconnect latency for online application(s).

So, in the production database, it is generally a good idea to control parallelism so that you don't accidentally run out of resources. Further, you need to understand that AutoDOP uses *estimated* run time to decide if AutoDOP should be enabled or not. Hence, it is critical to have optimal statistics in a RAC environment so that excessive PX executions are not accidentally triggered due to a statistical issue.

AutoDOP feature is more suitable for data warehousing environments. You might need to tune the `parallel_min_time_threshold` parameter to match the workload in your RAC cluster. By default, the parallel_min_time_threshold parameter is set to a value of 10 seconds. In my experience, a lower default value for the parallel_min_time_threshold parameter will trigger a tsunami of parallel executions. So, if you are planning to enable the AutoDOP feature, increase the parallel_min_time_threshold parameter to a higher value, at least 600 seconds.

This feature is very useful in ad hoc reporting and data warehousing environments, where queries initiated by the users may not specify selective predicates and parallelism may be the only option to tune those queries. It may not be feasible to tune those ad hoc queries individually either. In these scenarios, it may be prudent to use the AutoDOP feature and let the optimizer choose optimal parallelism, but use Database Resource Manager or proper parallel_max_servers to control query parallelism.

Version 12c introduces a new parameter, `parallel_degree_level`, and this parameter controls the aggressiveness of default DOP calculations. It also controls the scaling factor for default DOP calculations. If you set this parameter value to 50, then calculated default DOP will be multiplied by 0.50, reducing DOP to one-half of the default DOP.

Parallel Statement Queuing

It is all too common for SQL statements to acquire fewer PX servers than the requested DOP in a production database. Consider a SQL statement requesting a DOP of 32, but other SQL statements currently executing parallel statements are using all PX servers except for four; then, this SQL statement will execute with a reduced parallelism of 4. Even if more PX servers are available minutes after the start of parallel exectuion, the SQL statement will continue to execute with a DOP of 4. A process with a normal run time of 1 hour might run for 8 hours (assuming a linear extrapolation) and cause performance issues. Had that SQL statement delayed the execution for a few seconds, it could have acquired all requested PX servers and might have completed the task in normal run time.

■ **Note** A short story about parallel execution. We had a client performance issue: a job intermittently ran longer than normal. Our task was to identify the root cause, as the data was about the same every day. Reviewing ASH data, we were able to identify that a critical SQL statement in the program did not get enough PX servers on slower days. Even though the parallel_max_servers parameter was set to a much higher value, the SQL statement did not get enough PX servers. Finally, we found the root cause to be that another process was scheduled at exactly the same second, consuming nearly all available PX servers and starving the critical process. Whichever job allocated PX servers first got the PX servers, and the other job could not get all of the required PX servers. The solution we used was to delay the non-critical job by a minute to allow the critical process to acquire all PX servers. However, parallel statement queuing would have been an even better solution for this problem.

Prior to Database version 11.2, the parameter parallel_min_percent could be used (with little coding) to wait for requested PX servers to be available. Since version 11.2, a waiting scheme for PX server availability can be achieved more elegantly using the Parallel Statement Queuing feature. Essentially, if the requested number of PX servers are not available, then the parallel execution is delayed and the process waits in a statement queue until the requested PX servers are available.

If the parallel_degree_policy is set to AUTO, then the Parallel Statement Queuing feature is enabled. This feature also can be enabled in a session by altering the `_parallel_statement_queuing` parameter to true.

```
alter session set "_parallel_statement_queuing"=true;
```

The `parallel_servers_target` parameter controls when the Parallel Statement Queuing feature becomes active. If the number of PX servers currently executing in a cluster is greater than parallel_servers_target, then the feature becomes active. Consider an example: if parallel_servers_target is set to 96 and the number of PX servers currently active is 100, then this feature becomes active. If a SQL statement requests a parallel degree of 32, then the session will

be queued until at least 32 PX servers are available in the cluster.[7] As soon as 32 PX servers are available, the session will allocate PX servers and continue processing. In essence, this feature is protecting a SQL statement from executing with reduced number of PX servers. Still, the maximum number of PX servers that can execute at any time in an instance is dictated by the parallel_max_servers parameter.

Consider the following setup to explain this feature with clarity. Assuming that the server has 32 cores (cpu_count=32) and three active instances in a cluster, then the default values for the parameters parallel_max_servers and parallel_servers_target are calculated as follows:

```
Parallel_max_servers = 32* parallel_threads_per_cpu * (2 if pga_aggregate_target >0) * 5
                = 32 * 2 * 2 * 5 = 640

Total parallel_max_servers     =  parallel_max_servers * active_instance_count
                = 640 * 3 = 1,920

Parallel_servers_target     = 4 * 32 * parallel_threads_per_cpu * active_instance_count
                = 4 * 32 * 2 * 3 = 768 PX servers
```

If the total number of active PX servers exceeds 768, then only Parallel Statement Queuing will become active. However, the feature would not limit PX allocation yet. Suppose that 800 PX servers are active; if another process requests 200 more PX servers, then the process will continue executing in parallel with 200 PX servers without waiting in the statement queue, as the total number of active PX servers is lower than the parallel_max_servers parameter.

Suppose that three sessions consumed 1,500 PX servers in total, leaving just 420 PX servers available. If another process requests 500 PX servers, then the process will wait until all 500 PX servers are available. So, the parallel_servers_target parameter determines the time at which this feature will become active, and the feature will limit PX server allocation only when requested PX servers are not available.

Default values for parallel_max_servers and parallel_servers_target parameters are very high and overly optimistic about I/O resources available in a database machine. It is not uncommon to keep the parallel_max_servers parameter to a lower value than the default. If you adjust the parallel_max_servers parameter, then you should adjust the parallel_servers_target parameter also. You should consider the type of workload, I/O resources available in the database machine, and PX activity to set these parameters.

```
A session enqueued waiting for PX servers will wait for resmgr: pq queued event.
```

It is possible to enable the Parallel Statement Queuing feature for a SQL statement using the hint STATEMENT_QUEUING even if the Parallel Statement Queuing feature is disabled in the database. Additionally, the hint NO_STATEMENT_QUEUING can be used to disable the Parallel Statement Queuing feature if the feature is enabled.

However, the Parallel Statement Queuing feature doesn't work well with the parallel_force_local parameter. If the parameter parallel_force_local is TRUE, then parallel_statement_queuing keeps the session in the queue, even if PX servers are available in other nodes.[8] It is a better idea to use services instead of the parallel_force_local parameter if you are planning to use the Parallel Statement Queuing feature.

[7]Note that parallel_max_servers parameter controls the hard maximum number of PX servers that can be active in an instance. So, even after exceeding 96 PX servers as specified by parallel_servers_target, SQL statement may be able to acquire PX servers if parallel_max_servers is set to a value greater than 128.

[8]It is also possible that I encountered an undocumented bug in 11gR2 and 12c versions. In future versions, this behavior needs to be tested.

Prioritizing Parallel Statement Queues (12c)

The Parallel Statement Queuing feature works in a strict First In- First Out (FIFO) model in Database versions up to 11.2. In version 12c, Database Resource Manager can be utilized to create multiple parallel statement queues and prioritize the next parallel statement execution. Parallel statements from a higher-priority parallel statement queue are dequeued for the next execution. The parallel statement queue feature is enhanced to have multiple statement queues.

It is easier to understand priority among parallel statement queues using an example. Consider that you want to design a queuing scheme with the following objectives:

1. SYS_GROUP should have highest priority and any statements in this consumer group must be dequeued first.
2. If no session is waiting from the SYS_GROUP queue, then the SQL statement should be dequeued in the order of PX_CRITICAL, PX_HIGH, PX_MEDIUM, and PX_LOW queues.

The following code will implement the designed priority objective. In this example, four new consumer groups are created and priority is assigned among those consumer groups. First, a pending area is created to make changes to resource consumer groups. The next step creates a resource plan named PQ_STMT_PLAN.

```
begin
 dbms_resource_manager.create_pending_area();
  end;
  /
--------------Create Resource Manager Plan -----------------------------
begin
 dbms_resource_manager.create_plan(
  plan => 'PQ_STMT_PLAN',
  comment => 'Resource plan for PQ statement Queuing 12c');
end;
/
```

The following step creates four Resource Manager consumer groups. These consumer groups will be associated with a plan directive in the next step.

```
-------------- Create Resource Manager Consumer Groups -------------------
begin
 dbms_resource_manager.create_consumer_group(
  consumer_group => 'PX_CRITICAL',
  comment => 'Resource consumer group/method for online users / report sessions');
 dbms_resource_manager.create_consumer_group(
  consumer_group => 'PX_BATCH_HIGH',
  comment => 'Resource consumer group/method for batch high users');
 dbms_resource_manager.create_consumer_group(
  consumer_group => 'PX_BATCH_MED',
  comment => 'Resource consumer group/method for batch med users');
 dbms_resource_manager.create_consumer_group(
  consumer_group => 'PX_BATCH_LOW',
  comment => 'Resource consumer group/method for batch low users');
end;
/
```

The next step creates the priorities among parallel statement queues of the resource consumer groups. Directive mgmt_p1 decides the priority of a specific queue in level 1. In the following example, SYS_GROUP has a 100% priority at level 1. A parallel statement with SYS_GROUP as the consumer group will be chosen next to be executed. If there is no statement belonging to SYS_GROUP that is waiting in the parallel statement queue, then a parallel query from the next-lowest-level PX statement queues will be considered for execution.

Directive mgmt_p2 decides the priority of the parallel statement queue at level 2, and the value of mgmt_p2 is set to 50 for the consumer group PX_CRITICAL. So, there is a 50% probability that the next query will be scheduled for the sessions with a resource consumer group associated with the PX_CRITICAL group. Similarly, PX_BATCH_HIGH has a 30% probability that a PX session for the PX_BATCH_HIGH consumer group will be chosen to be executed next. Also, PX_BATCH_LOW and PX_BATCH_MED have a probability of 10%.

The priority of OTHER_GROUPS is set to 100 at level 3 by setting the mgmt_p3 attribute. A statement belonging to OTHER_GROUPS will be executed only if there is no other statement waiting for higher-level queues.

```
------------- Create Resource Manager Plan Directives --------------------
begin
 dbms_resource_manager.create_plan_directive(
  plan => 'PQ_STMT_PLAN',
  group_or_subplan => 'SYS_GROUP',
  comment => 'SYS sessions at level 1',
  mgmt_p1 => 100 );
 dbms_resource_manager.create_plan_directive(
  plan => 'PQ_STMT_PLAN',
  group_or_subplan => 'PX_CRITICAL',
  comment => 'Online day users sessions at level 1',
  mgmt_p2 => 50);
 dbms_resource_manager.create_plan_directive(
  plan => 'PQ_STMT_PLAN',
  group_or_subplan => 'PX_BATCH_HIGH',
  comment => 'Batch users level 2 -HIGH- Fast running ',
  mgmt_p2 => 30);
 dbms_resource_manager.create_plan_directive(
  plan => 'PQ_STMT_PLAN',
  group_or_subplan => 'PX_BATCH_MED',
  comment => 'Batch users sessions at level 2 - MED - Medium running',
  mgmt_p2 => 10);
 dbms_resource_manager.create_plan_directive(
  plan => 'PQ_STMT_PLAN',
  group_or_subplan => 'PX_BATCH_LOW',
  comment => 'Batch users sessions at level 2 - LOW - long running',
  mgmt_p2 => 10);
dbms_resource_manager.create_plan_directive(
  plan => 'PQ_STMT_PLAN',
  group_or_subplan => 'OTHER_GROUPS',
  comment => 'Batch users sessions for Other groups',
  mgmt_p3 => 100);
end;
/
begin
 dbms_resource_manager.validate_pending_area();
end;
/
```

```
begin
 dbms_resource_manager.submit_pending_area();
end;
/
```

In a nutshell, version 12c enhances the strict FIFO policy employed in earlier versions for PX statement execution. A complex scheme such as the one discussed in this section can be used to prioritize the workload so that higher-priority sessions do not have to wait for lower-priority parallel statement execution.

This enhancement is useful in mixed-workload environments such as highly critical short-running parallel executions and low-priority long-running parallel executions mixed in the same database. If critical SQL statement must wait for low-priority parallel statement execution, then critical parallel execution may be delayed. Worse, low-priority execution can be a long-running statement with a higher number of PX servers, and so the wait time for critical statements can be longer. By enabling prioritization within the workload, you can reduce the wait time for critical SQL execution in parallel statement queues.

Critical Parallel Statement Queues (12c)

In the preceding section, you learned how to prioritize parallel statement execution among consumer groups. You may recall that the Parallel Statement Queuing feature is enabled only after the active PX server count exceeds the parallel_servers_target parameter. In some cases, you might prefer for ultra-critical jobs to acquire PX servers completely bypassing the PX statement queue. For example, an important index must be rebuilt quickly: waiting in the parallel statement queue is not acceptable, and you prefer to bypass it completely. As the parallel_servers_target parameter is configured to be less than the parallel_max_servers parameter, it may be beneficial to bypass the queue for ultra-critical workloads.

Version 12c introduces the attribute `parallel_stmt_critical` for a resource consumer group. If you set this attribute to BYPASS_QUEUE, then sessions attached to this resource consumer group will bypass the PX statement queue and start execution immediately. As the parallel_max_servers parameter is a hard threshold, it is possible for this critical statement to execute with fewer PX servers than requested.

```
begin
      dbms_resource_manager.create_pending_area();
   end;
   /
   begin
dbms_resource_manager.create_consumer_group(
  consumer_group => 'PX_ULTRA_CRIT',
  comment => 'Resource consumer group/method for critical users');
dbms_resource_manager.create_plan_directive(
  plan => 'PQ_STMT_PLAN',
  group_or_subplan => 'PX_ULTRA_CRIT',
  comment => 'Very Critical stuff',
  parallel_stmt_critical => 'BYPASS_QUEUE');
    end;
/
begin
 dbms_resource_manager.validate_pending_area();
end;
/
begin
 dbms_resource_manager.submit_pending_area();
end;
/
```

Think of the bypass mechanism as an override mechanism. You should design a consumer group with this attribute only if it is absolutely required, as there is a danger of critical statements executing with a lower number of PX servers than requested.

Parallel Statement Queue Timeout (12c)

By default, PX statements will wait in the statement queue indefinitely, but that may not be preferred in a production environment. Version 12c introduces a timeout, namely, `parallel_queue_timeout`, for a resource consumer group. If a session waits in the parallel statement queue longer than the parallel_queue_timeout parameter value, then the statement is terminated with an ORA-7454 error and removed from the parallel statement queue. For example, sessions associated with PX_BATCH_LOW will wait in the parallel statement queue for an hour at most.

```
dbms_resource_manager.create_plan_directive(
  plan => 'PQ_STMT_PLAN',
  group_or_subplan => 'PX_BATCH_LOW',
  comment => 'Batch users sessions at level 2 - LOW - long running',
  mgmt_p2 => 10, parallel_queue_timeout =>3600);
```

You should also note that it is the responsibility of the application code to handle this error condition and react properly if ORA-7454 is received by the application.

Grouping Parallel Statement Queue Execution (12c)

A problem with the Parallel Statement Queuing feature is that a transaction may contain multiple parallel statements, and every parallel statement can wait in the statement queue, leading to a prolonged transaction. For example, a transaction may consist of two statements, with the first inserting 500 million rows into a table using parallelism and the second summarizing those 500 million rows using parallelism. Since the Parallel Statement Queuing feature works at the statement level, the session might wait twice for the whole transaction, inducing performance issues. Further, multiple waits in the parallel statement queue have the effect of prolonging a transaction, which could further lead to unnecessary locking issues or UNDO management issues. The following group of SQL statements shows the example scenario.

```
-- First parallel statement
Insert /*+ parallel(10) */ into huge_table
  Select  /*+ parallel (10) */ col1, col2 .. from oltp_transactions..;
update  emp..     ;              -- Serial statement

-- Second parallel statement
insert /*+ parallel (10) */ into summary_table
select /*+ parallel (10) */ ..from huge_table;

commit;
```

It may be acceptable for the first parallel SQL statement to wait in the parallel statement queue, but the subsequent, nearly immediate, parallel statement should not wait in the parallel statement queue. In 11gR2, every statement will wait in the PX statement queue if the requested PX servers are not available.

Version 12c introduces a new feature to group multiple parallel statements to reduce the number of waits in the parallel statement queue. Procedure call BEGIN_SQL_BLOCK and END_SQL_BLOCK can be used to group parallel statements. The following code groups prior parallel statements and enables reuse of PX servers acquired in the earlier parallel statement execution.

```
exec dbms_resource_manager.begin_sql_block;
Insert /*+ parallel(10) */ into huge_table
  Select  /*+ parallel (10) */ col1, col2 .. from oltp_transactions..;
update  emp..     ;              -- Serial statement
--second parallel statement
insert /*+ parallel (10) */ into summary_table
select /*+ parallel (10) */ ..from huge_table;
commit;
exec dbms_resource_manager.end_sql_block;
```

With this strategy, multiple parallel statements in the block will be considered as one group, and the waits in parallel statement queue will be minimal. The downside of this feature is that if there is a delay between two parallel statement executions in the same parallel statement block, then PX servers can be unnecessarily held up by this process.

Debugging PX Execution

Debugging PX execution problems in a RAC cluster requires reviewing PX execution environment setup. The PX tracing event is useful to understand why a query was not parallelized even though the optimizer chose a PX plan.[9]

In the following example, I will review a problem of a SQL statement executing serially, even though the optimizer execution plan indicates a PX plan. I will trace PX execution setting _px_trace event to high, all.[10]

```
alter session set "_px_trace"=high,all;

select /*+ parallel (8) full(h) full(l) */ count(*)  from
oe_order_headers_all h, oe_order_lines_all l
where h.header_id = l.header_id and
h.open_flag='N';
```

Reviewing the trace file generated by _px_trace event, we can walk through the PX server allocation and parallel execution. This part of the trace file shows the default DOP calculation. In this database, the calculated default DOP is 168.

```
kxfrSysInfo                                                  [      12/     0]
        DOP trace -- compute default DOP from system info
        # instance alive  = 3 (kxfrsnins)
kxfrDefaultDOP                                               [      12/     0]
        DOP Trace -- compute default DOP
            # CPU       = 28
            Threads/CPU = 2 ("parallel_threads_per_cpu")
            default DOP = 56 (# CPU * Threads/CPU)
            default DOP = 168 (DOP * # instance)
            Default DOP = 168
```

[9]Note that discussion here is not “why is SQL statement not parallelized?”, but rather “why didn’t the PX plan use the PX servers?”.
[10]Note that _px_trace=high, all creates a huge amount of diagnostics details in PX trace files. You might need to contact Oracle Support to understand further details printed in a PX trace file.

Subsequent lines show the reasoning behind serial execution. In this database, the Database Resource Manager disabled parallelism for the current user by granting zero PX servers. That's the reason why the SQL statement did not get any PX servers.

```
PX_Messaging:kxfp.c@17808:kxfpclinfo():     2   (78  :0   :100:35  :16    :0     :0   ...)
    Ressource Manager reduced num server to maxdop = 0
    serial -- Requested 8 threads, Granted 0 Aborting!
       Max Servers=32  Adaptive=0 RM Max-DOP=0
```

After switching to the correct resource consumer group and retracing the execution of the statement, we can see that the query allocated eight threads each in instances 1 and 2. As you can see, two sets of eight threads were requested by this parallel query, and all of them were granted by the Resource Manager. In the following output, P000 is allocated from instance 2 and the complete trace file lists all 16 PX server allocations.

```
kxfpgsg                                                    [     586/     0]
         getting 2 sets of 8 threads, client parallel query execution flg=0x234

         Height=8, Affinity List Size=0, inst_total=1, coord=2
         Insts     2
         Threads   8
kxfpg1sg                                                   [     587/     1]
         q:e8dfd9668 req_threads:8 nthreads:8 unit:1 #inst:1 normal
         jStart:0  jEnd:32  jIncr:1 isGV:0  i:0 instno:2 kxfpilthno:8
kxfpg1srv                                                  [     587/     0]
         trying to get slave P000 on instance 2 for q:e8dfd9668
         slave P000 is local
         found slave P000 dp:e11cdc888 flg:18
         Got It. 1 so far.
```

You can debug PX execution problems by tracing the statements with _px_trace event and reviewing the generated trace file. Refer to Oracle support document ID 400886.1 for further information about _px_trace command.

Index Creation in RAC

Parallel index creation is a much-discussed-topic in a RAC database. Should you allocate PX servers from all instances to create a big index? Or should you allocate PX servers from just one instance? The strategy depends upon the environment, and the following resource constraints should be considered before deciding the strategy:

1. Does one node have enough memory, CPU, and I/O capacity to complete index creation?
2. How much interconnect bandwidth is available for PQ workload? Can the bandwidth sustain the burst of interconnect traffic generated by index creation statements?
3. Is there sufficient I/O bandwidth such that the LGWR process in one node can write all index redo entries to the log files without incurring any performance delay? Index creation in two nodes uses the LGWR processes in both nodes and might perform better if LGWR is a bottleneck.

For example, a 6GB index creation using two nodes induced an interconnect traffic of 2.6GB. The following shows the interconnect traffic bytes from x$ksxpclient.

```
INSTANCE_NUMBER         NAME       DIFF_SENT   DIFF_RECV
-----------------       ---------  ----------  ----------
               1        ipq        1440328342  1297116631
               2        ipq        1299190976  1438331216
               3        ipq            733181      575309
               4        ipq            482387      350462
```

The same index creation with PX servers allocated in all four nodes generated 4.1GB of interconnect traffic for PQ workload.[11] So, index creation from many nodes can increase interconnect traffic.

```
INSTANCE_NUMBER         NAME      DIFF_SENT  DIFF_RECV
---------------         --------- ---------- ----------
              1         ipq       1101141155  989508581
              2         ipq       1015271038  992655090
              3         ipq       1150033881  981687750
              4         ipq        906133897 1204963877
```

If your database server has sufficient resources, you should probably set parallel_force_local=TRUE in your session and create the index in that session. This method would allocate all PX servers in the same node, reducing interconnect traffic.

Also, you should know that index creation on partitioned tables can be slower if not executed properly. By default, a partition is the granularity of PX allocation while creating a partitioned index. So, *one* PX server is assigned to process *one* table partition while creating a partitioned index. This strategy can cause performance issues, especially if the partitioned table is huge, which is typically the case. Worse, if the table has non-uniform size partition, then a few PX servers can be stuck processing huge partitions while other PX servers are idle. Ideally, at least half of allocated PX servers should be active during index creation. Hence, it is an optimal approach to create index structures with `unusable` state and then rebuild index partitions individually. Further, you can design your strategy in such a way that each index partition is rebuilt by an instance.

■ **Note** The dbms_pclxutil package is immensely useful to rebuild partitioned indexes with ease. The procedure call dbms_pclxutil.build_part_indexes uses a parallel-enabled rebuild strategy to rebuild index partitions effectively, but the package doesn't allow you to distribute jobs among instances. If you are planning to use an instance to rebuild indexes, then consider using the dbms_plcxutil package.

Parallel DML in RAC

DML statements can be executed in parallel, utilizing PX servers from multiple instances. There are similarities in processing between parallel DML and index creation. Please refer to the section "Index Creation in RAC" earlier in this chapter for discussions about choosing intra-instance or inter-instance parallelism.

Parallel DML must be enabled in the session using the `alter session enable parallel dml` command. Two types of workload are possible with parallel DML, and the type chosen depends upon the execution plan: (1) parallel scan only, or (2) parallel scan and change. In the first case, PX servers will scan the objects and the DML operation will be performed by QC. In the second case, PX servers will scan the objects and perform DML operations too. In the following execution plan output, in step 3, the table is updated and then a message is sent to QC. Essentially,

[11]Calculated by summing up either the DIFF_SENT column or the DIFF_RECV column, but not both.

both scan and change are done by the PX server processes themselves. The keyword UPDATE is under the PX SEND QC operation, indicating that PX servers perform both scan and update operations.

```
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name           | Starts | E-Rows | A-Rows |   A-Time   |...
-------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT      |                |      1 |        |      4 |00:00:24.91 |
|   1 |  PX COORDINATOR       |                |      1 |        |      4 |00:00:24.91 |
|   2 |   PX SEND QC (RANDOM) | :TQ10000       |      0 |  32186 |      0 |00:00:00.01 |     |
|   3 |    UPDATE             | HUGETABLE_HASH |      0 |        |      0 |00:00:00.01 |
|   4 |     PX BLOCK ITERATOR |                |      0 |  32186 |      0 |00:00:00.01 |
|*  5 |      TABLE ACCESS FULL| HUGETABLE_HASH |      0 |  32186 |      0 |00:00:00.01 |
-------------------------------------------------------------------------------------------
```

In the following execution plan, PX servers are scanning and identifying the rows to update. The QC process performs the change. Notice that the UPDATE keyword precedes the PX SEND QC step, indicating that changes are performed by the QC process.

```
-------------------------------------------------------------------------------------------
| Id  | Operation                         | Name           | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT                  |                |     1 |  1028 |  2600   (1)|
|   1 |  UPDATE                           | HUGETABLE_HASH |       |       |            |
|   2 |   PX COORDINATOR                  |                |       |       |            |
|   3 |    PX SEND QC (RANDOM)            | :TQ10003       |     1 |  1028 |  2600   (1)|
|   4 |     NESTED LOOPS                  |                |     1 |  1028 |  2600   (1)|
|   5 |      BUFFER SORT                  |                |       |       |            |
|   6 |       PX RECEIVE                  |                |       |       |            |
|   7 |        PX SEND BROADCAST          | :TQ10002       |       |       |            |
|   8 |         VIEW                      | VW_NSO_1       | 32186 |   408K|  2445   (1)|
|   9 |          HASH UNIQUE              |                |     1 |   817K|            |
|  10 |           PX RECEIVE              |                |     1 |   817K|            |
|  11 |            PX SEND HASH           | :TQ10001       |     1 |   817K|            |
|  12 |             HASH UNIQUE           |                |     1 |   817K|            |
|* 13 |              HASH JOIN            |                | 32186 |   817K|  2445   (1)|
|  14 |               PX BLOCK ITERATOR   |                | 32186 |   408K|  1222   (0)|
|  15 |                TABLE ACCESS FULL  | HUGETABLE_HASH | 32186 |   408K|  1222   (0)| ...
...
```

In RAC, you should write parallel DML so that all PX server processes perform the changes. With this strategy, true parallelism is achieved, as multiple PX servers are performing the DML operation. If only QC is carrying out the changes and if the magnitude of change is much higher (for example, numerous rows to be updated), then performance will suffer, as just one process is performing the change while PX servers are idle. Further, with inter-instance parallelism, multiple LGWR processes can work on writing the redo buffer to the log file, concurrently improving redo write throughput.

Concurrent Union Processing (12c)

Prior to 12c, union branches were executed one at a time. For example, if there were three union branches in a SQL statement, then each of these branches would have been executed sequentially. Note that each individual union branch can execute utilizing PX servers, but only one union branch will be active during SQL execution.

This sequential execution is not efficient for a few types of SQL statements. For example, in the following SQL statement, most of the time is spent accessing the remote database over the database link; that is, time is spent in the network transfer. Sequential execution of each branch adds up to a huge elapsed time interval for the SQL statement.

```
select /*+ PQ_CONCURRENt_UNION */ col1, col2, col3 from orders@retail_oltp
union all
select col1, col2, col3 from warehouse_orders@shipping
union all
select col1, col2, col3 from orders;
```

Version 12c introduces a concurrent union processing feature, and multiple union branches can be executed in parallel. So, in this example, since most of the time is spent in the network, executing the union branches concurrently will reduce the elapsed time of the SQL statement.

In RAC, this new feature has interesting uses. A complex parallel statement execution, with multiple union all branches, can be designed to execute the utilization of inter-instance parallelism in multiple nodes.[12] Essentially, each node will work on its own union all branch.

Partition-Wise Join

Traditional PX execution uses the Producer/Consumer model, and partition-wise join is an exception to that model. In partition-wise join, each PX server will read a partition from joined tables and perform join processing. Finally, joined result set is sent to the next process in the PX tree or QC. Since this technique does not use the Producer/Consumer model, PX messaging between the processes is reduced. In RAC, a partition-wise join operation in an inter-instance parallel execution does not induce excessive interconnect traffic.

An execution plan for a partition-wise join follows. In this plan, you can see that PX PARTITION HASH ALL in step 5 precedes the HASH JOIN step. This execution sequence indicates that the partition-wise join is in play. Essentially, every PX server process is performing its operation underneath the PX PARTITION step. Each PX server performs HASH join between two partitions of the table, aggregates data, and then sends the aggregated results to QC. As both aggregation and join operations are performed by a PX server process, interconnect traffic is minimal.

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name           | Starts | E-Rows | A-Rows |   A-Time   |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                |      1 |        |      1 |00:00:37.42 |
|   1 |  SORT AGGREGATE          |                |      1 |      1 |      1 |00:00:37.42 |
|   2 |   PX COORDINATOR         |                |      1 |        |      8 |00:00:37.42 |
|   3 |    PX SEND QC (RANDOM)   | :TQ10000       |      0 |      1 |      0 |00:00:00.01 |
|   4 |     SORT AGGREGATE       |                |      0 |      1 |      0 |00:00:00.01 |
|   5 |      PX PARTITION HASH ALL|                |      0 |  32186 |      0 |00:00:00.01 |
|*  6 |       HASH JOIN          |                |      0 |  32186 |      0 |00:00:00.01 |
|   7 |        TABLE ACCESS FULL | HUGETABLE_HASH |      0 |  32186 |      0 |00:00:00.01 |
|   8 |        TABLE ACCESS FULL | HUGETABLE_HASH |      0 |  32186 |      0 |00:00:00.01 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   6 - access("T1"."N1"="T2"."N1")
```

[12]In 12c, you may not be able to force this sort of inter-instance parallelism where instances are operating on disjoint branches of union all execution. But, hopefully, future release might provide a mechanism to force a parallel concurrent execution.

In Figure 12-4, two instances are participating in a partition-wise join between SALES and ORDERS tables. PX servers P001 and P002 participate in this join operation in both instances. P001 reads partition P1 of the SALES table and partition P1 of the ORDERS table and joins these two partitions. Similarly, process P002 reads partition P2 of the SALES table and partition P2 of ORDERS table and joins the rows from these two partitions. PX messages are kept minimal between PX server processes, since this type of execution does not use the Consumer/Producer model. Finally, a result set is sent to the QC, reducing interconnect traffic for an inter-instance parallel operation.

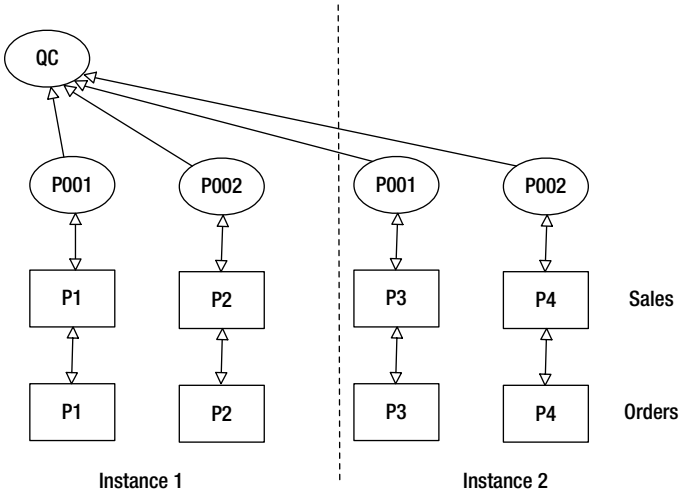


Figure 12-4. *Partition-wise join*

However, specific conditions must be met for partition-wise join to be chosen. Some requirements for partition-wise joining include matching of parallelism chosen for joining tables and matching of the partition boundaries of the partitioned tables.

Summary

Parallel execution is an effective feature in a RAC environment. Although Oracle Database is optimized for effective placement of PX servers to reduce interconnect traffic, you should design interconnect bandwidth to match your application workload. If your application workload is mostly data warehouse queries, then you could expect higher interconnect traffic due to parallel execution. The following are the essential key elements of optimal practices in setting up parallelism in RAC:

1. Use services to control placement of PX servers. Placement using the instance_groups parameter is deprecated, but placement using a combination of services and parallel_instance_group is allowed.
2. Verify that the PEMS parameter is set to 16K. In PX-intensive environments, use Jumbo Frames.
3. If parallel queries do not get enough PX servers, use the Parallel Statement Queuing feature to provide consistent elapsed times.
4. For DDL statements on huge tables, if one node can handle the PX execution, then set up parallel parameters so that PX servers can be allocated from one node or use a parallel_force_local parameter to force PX server allocation to a node.
5. Use raccheck and exacheck tools to verify parallelism setup in a RAC environment.

CHAPTER 13

Clusterware and Database Upgrades

by Syed Jaffar Hussain

Whenever Oracle announces a new version, the million-dollar question facing every organization and every DBA is whether to upgrade their current (and mostly stable) environment. There is little doubt that the prospect of an upgrade gives you, as a DBA, butterflies in your stomach and the feeling that it will take a lot of hard work to make the process successful. It's hardly surprising that many organizations are afraid to accept the risks of an upgrade.

The objective of this chapter is to explain the key elements that drive an organization to consider upgrading. It also offers the best practices, tips, and tricks to upgrade your current Oracle Grid Infrastructure (GI) and databases to the new Oracle 12c.

Typically, people reject the risk of an upgrade for the following main reasons:

- The organization's legacy application doesn't support the new Oracle release.
- The organization doesn't want to face new challenges that might materialize after an upgrade.
- There are no experienced or skilled DBAs to perform such an important action.
- Resources, such as a nonproduction environment to simulate the change before taking them into production, are lacking.

Here are a few key factors that drive an organization or DBA to consider an upgrade of their existing environment nevertheless:

- To stay up to date with the latest technology as dictated by standards or security policies
- To address or fix some of the critical issues encountered in the current environment
- To leverage the enhancements of the new version to improve database performance or flexibility
- To continue to use Oracle Support; Oracle will stop supporting the older version in terms of providing bug fixes, patches, etc., after a grace period or a version's extended support policy

Configuration

This chapter used the following setup configuration to demonstrate an Oracle Clusterware 11gR2 (11.2.0.3) and Database upgrade to Oracle 12c (12.1.0.0.2):

- Oracle 11gR2: 11.2.0.3
- RedHat Enterprise Linux 5: OS

- Oracle ASM: for database storage
- Two-node RAC: rac1, rac2
- RAC database with two instances: PRDDB (prddb1,prddb2)
- Oracle 12c: 12.1.0.0.2
- OCR (Oracle Cluster Registry) and voting disk: on ASM diskgroup
- /u00 filesystem: 100GB space
- Following RPMs:
  - `binutils-2.17.50.0.6`
  - `compat-libstdc++-33-3.2.3`
  - `compat-libstdc++-33-3.2.3 (32 bit)`
  - `gcc-4.1.2`
  - `gcc-c++-4.1.2`
  - `glibc-2.5-58`
  - `glibc-2.5-58 (32 bit)`
  - `glibc-devel-2.5-58`
  - `glibc-devel-2.5-58 (32 bit)`
  - `ksh`
  - `libaio-0.3.106`
  - `libaio-0.3.106 (32 bit)`
  - `libaio-devel-0.3.106`
  - `libaio-devel-0.3.106 (32 bit)`
  - `libgcc-4.1.2`
  - `libgcc-4.1.2 (32 bit)`
  - `libstdc++-4.1.2`
  - `libstdc++-4.1.2 (32 bit)`
  - `libstdc++-devel 4.1.2`
  - `libXext-1.0.1`
  - `libXext-1.0.1 (32 bit)`
  - `libXtst-1.0.1`
  - `libXtst-1.0.1 (32 bit)`
  - `libX11-1.0.3`
  - `libX11-1.0.3 (32 bit)`

- libXau-1.0.1
- libXau-1.0.1 (32 bit)
- libXi-1.0.1
- libXi-1.0.1 (32 bit)
- make-3.81
- sysstat-7.0.2

Pre-Upgrade Checklist

Every successful deployment needs careful planning and must go through prerequisite verification before upgrading an Oracle environment. You are therefore advised to work through some of the important pre-upgrade tasks mentioned in the subsequent sections to ensure a smooth and successful upgrade.

Oracle Software Backups

Before any upgrade, it is essential to have an up-to-date backup copy of the existing Oracle software: GI and RDBMS, and also the Cluster's critical components (OCR and OLR [Oracle Local Registry]) in order to secure yourself from any sort of disaster situations that might arise before and after the upgrade. We recommend that you consider the following backup actions:

Back up your current GI and RDBMS homes, preferably the entire Oracle software filesystem.
Perform a manual physical OCR/OLR backup.
Perform a full Oracle Database backup of the source databases.

Validating Node Readiness

Validating node (server) readiness prior to the upgrade procedure is one of the key steps of a successful upgrade. It is mandatory to verify all the prerequisites, such as storage, OS kernel parameters, OS patches, etc., on the current nodes to ensure that they meet the minimal requirements to proceed with the upgrade.

Prerequisite validations can be performed in two ways: with the Cluster verification utility (`runcluvfy.sh`) or by letting the Oracle Universal Installer (OUI) perform the verification as part of the prerequisite checking during the upgrade process. The following command demonstrates an example how to use the cluvfy utility to perform the verification prior to installation of the Clusterware:

```
./runcluvfy.sh stage -pre crsinst -upgrade -n rac1,rac2 -rolling -fixup-src_crshome
/u01/app/11203/grid -dest_home /u01/app/12.1.0/grid -dest_version12.1.0 -verbose
```

■ **Note** The `runcluvfy.sh` must be executed from the 12c software staging location.

In the event of any validation failures produced by the preceding command, you should fix the issues and re-execute the command to ensure that all the prerequisite checks have been met.

Unsetting Oracle Variables

Ensure the Oracle variables for GRID and RDBMS homes on the local node are unset prior to cluster upgrade. The following example demonstrates how to unset Oracle variables on the Unix/Linux platforms:

```
$ unset ORACLE_HOME ORA_CRS_HOME TNS_ADMIN ORA_NLS10 ORACLE_HOME
```

The `$ORACLE_HOME/bin` path should not be part of the current PATH variable settings on the local node. If it is, remove the path from the current PATH settings.

Upgrading Flexibility and Restrictions

A few important points to keep in mind before proceeding with an upgrade process:

- Depending on your needs, you have the flexibility to choose a rolling cluster upgrade or a non-rolling cluster upgrade method. A rolling upgrade option lets you perform an upgrade on one node at a time while the other nodes in the cluster go on working, hence requiring less application downtime for the activity.
- For a non-rolling cluster upgrade, on the other hand, you will have to stop the cluster stack completely across all nodes in the cluster. This type of upgrade requires more downtime and a complete system stop.
- Clusterware and ASM instance can be upgraded together at once, or you may choose to perform Clusterware and ASM instance upgrades individually. In this approach, you will have to upgrade the Clusterware first, after a successful cluster upgrade to 12c, and subsequently upgrade the ASM instance using the ASM Configuration Assistant tool (`asmca`) from the 12c Grid Home. Clusterware and ASM instance upgrade together is highly recommended.
- Before upgrading to 12c, the OCR and voting disks must be migrated from RAW/Block devices to ASM diskgroup (refer to Chapter 2 to learn how to migrate OCR/voting disks to ASM diskgroups), as Oracle 12c no longer supports RAW/Block devices. If the OCR/voting disk files are already on ASM, then you can safely ignore this suggestion.
- You can't directly upgrade Oracle 10g, 11gR1, or 11gR2 Clusterware to Oracle 12c Flex Cluster. It can be upgraded only to standard cluster upgrade. To make use of Oracle 12c Flex Cluster benefits, all nodes in the cluster must be successfully upgraded to 12c before the Flex Cluster is configured.
- Only the owner of existing GI software can perform an upgrade to Oracle 12c. You can't use a different user to upgrade the existing GI home.
- Upgrades on shared Oracle homes are feasible.

Cluster and ASM Oracle 12c Upgrade Compatibility Matrix

Before upgrading the current environment, review the compatibility matrix to determine whether the current environment is capable of performing a direct upgrade to Oracle 12c or not. You need to be prepared for situations such as the current environment not supporting a direct upgrade; how then would you proceed?

Table 13-1 depicts a compatibility matrix for performing direct Oracle 12c upgrades from earlier Oracle versions.

Table 13-1. *Cluster Upgrade Compatibility Matrix*

Oracle Version	Compatibility
Oracle 10gR1 (10.1.0.5)	Direct upgrade possible
Oracle 10gR2 (10.2.0.3)	Direct upgrade possible
Oracle 11gR1 (11.1.0.6)	Direct upgrade possible
Oracle 11gR2 (11.2.0.2)	Direct upgrade possible: patch set 11.2.0.2.3 (PSU 3) or later must be applied

If your current Oracle environment doesn't match any of the Oracle versions listed in Table 13-1, you will have to perform an indirect upgrade, in which you need to first either apply a patch or upgrade it to the version specified in the table before you upgrade it to 12c.

Launching OUI

When all the prerequisites are met, and all upgrade best practices have been performed, you are good to go with the upgrade process. So, let's get into the real action. Navigate through the Oracle 12c GI software staging location and launch OUI by running the following command:

```
./runInstaller
```

Initiating an Oracle Clusterware Upgrade

The guidelines provided here, with the help of the screenshots, will demonstrate a two-node Oracle 11gR2 (11.2.0.3) GI upgrade to Oracle 12c (12.1.0.0.2) on Linux 5 OS.

1. If you don't want to receive updates about software updates from My Oracle Support, select the `Skip Software updates` option on the `Software Upgrade` screen.
2. Select the Upgrade Oracle GI or Automatic Storage Management option on the Installation Option screen, as demonstrated in Figure 13-1.

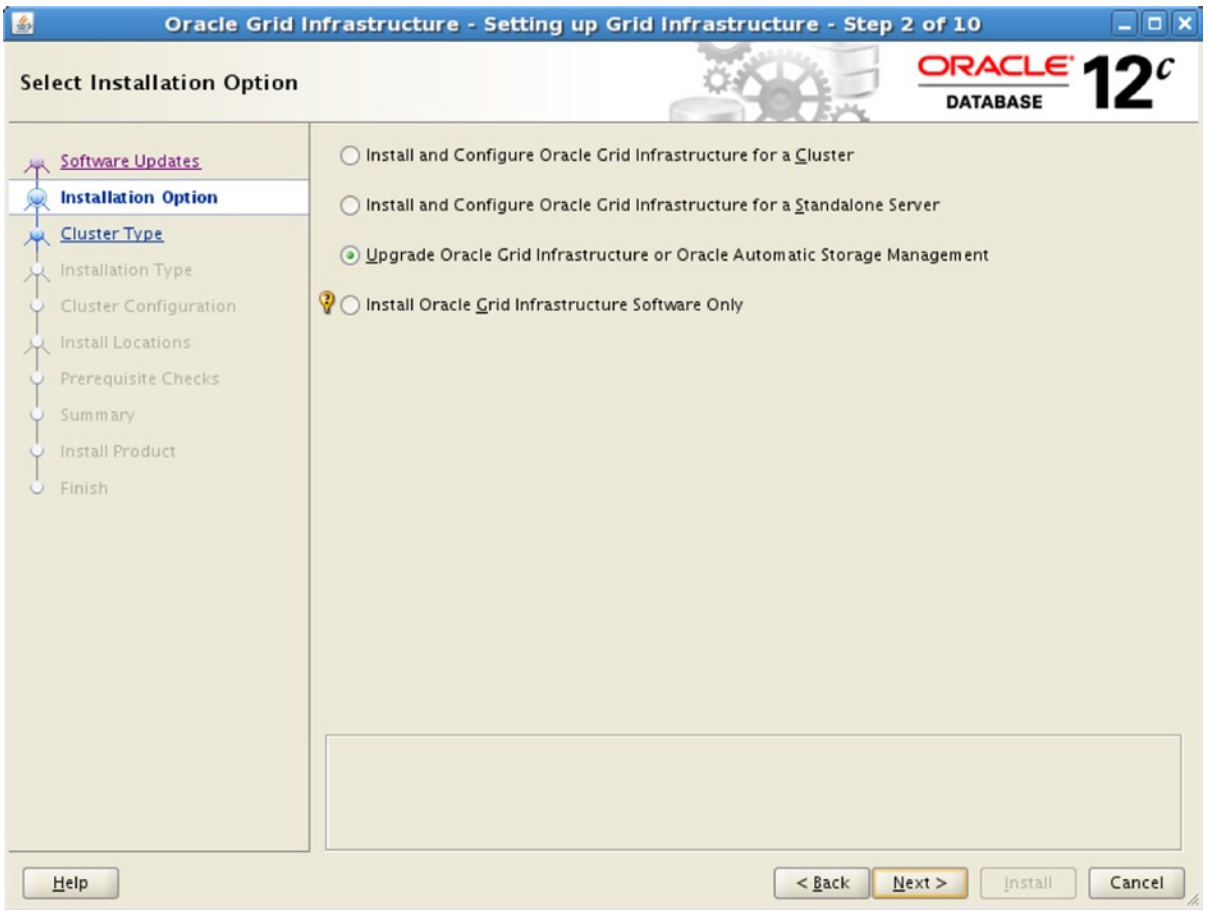


Figure 13-1. *Select Installation Option screenshot*

3. Select the preferred languages based on your requirements on the `Select Product Languages` screen.
4. Select all nodes from the given list as part of the cluster upgrade procedure on the `Node Selection` screen, as demonstrated in Figure 13-2.

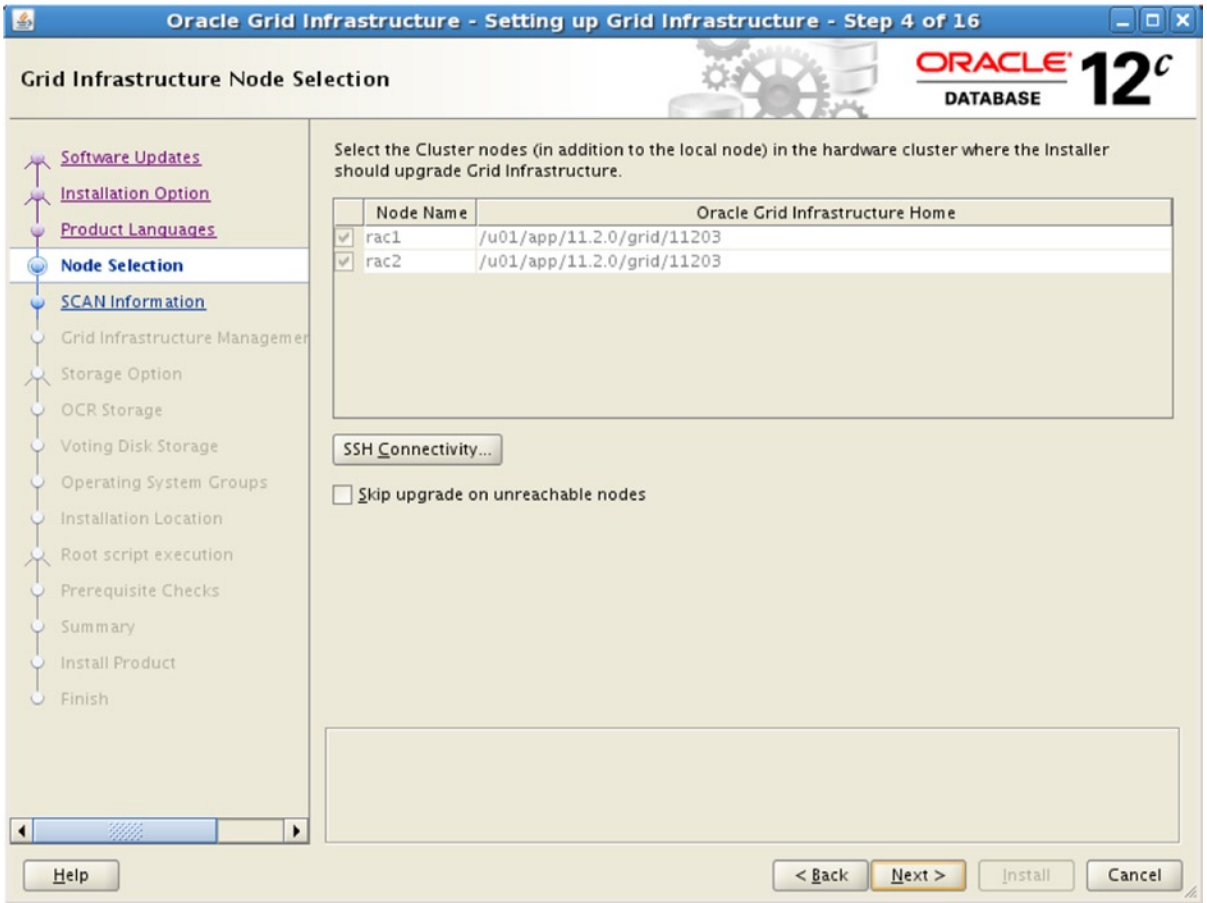


Figure 13-2. *GI Node Selection screenshot*

5. When the GI management repository is configured as part of an Oracle 12c new installation or upgrade, a new database in the context will be created to maintain Cluster Health Monitor (CHM) data and other Cluster futures. The historical data kept in the database can be used later on to analyze the cluster issue. At the time of writing this chapter, there is no other option available to configure the GI management repository if you miss it during the installation upgrade. If you select the configuration (Figure 13-3), the OCR and voting disks will be automatically created in the same ASM diskgroup used for the MGMT database.

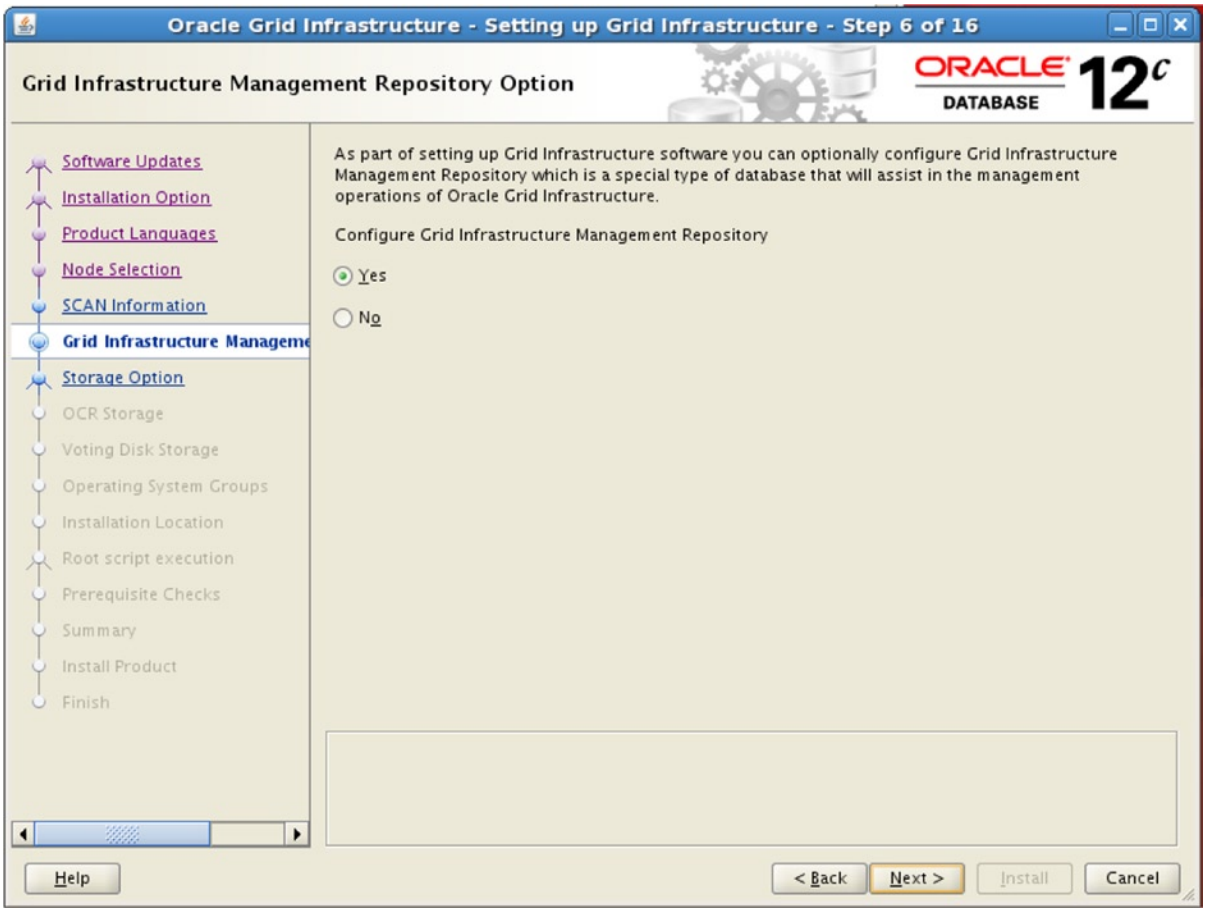


Figure 13-3. *GI Management Repository Option screenshot*

6. Set the appropriate groups on the `Operating System Groups` screen.
7. Input the Oracle 12 GI location details on the `Installation Location`, as shown in Figure 13-4.

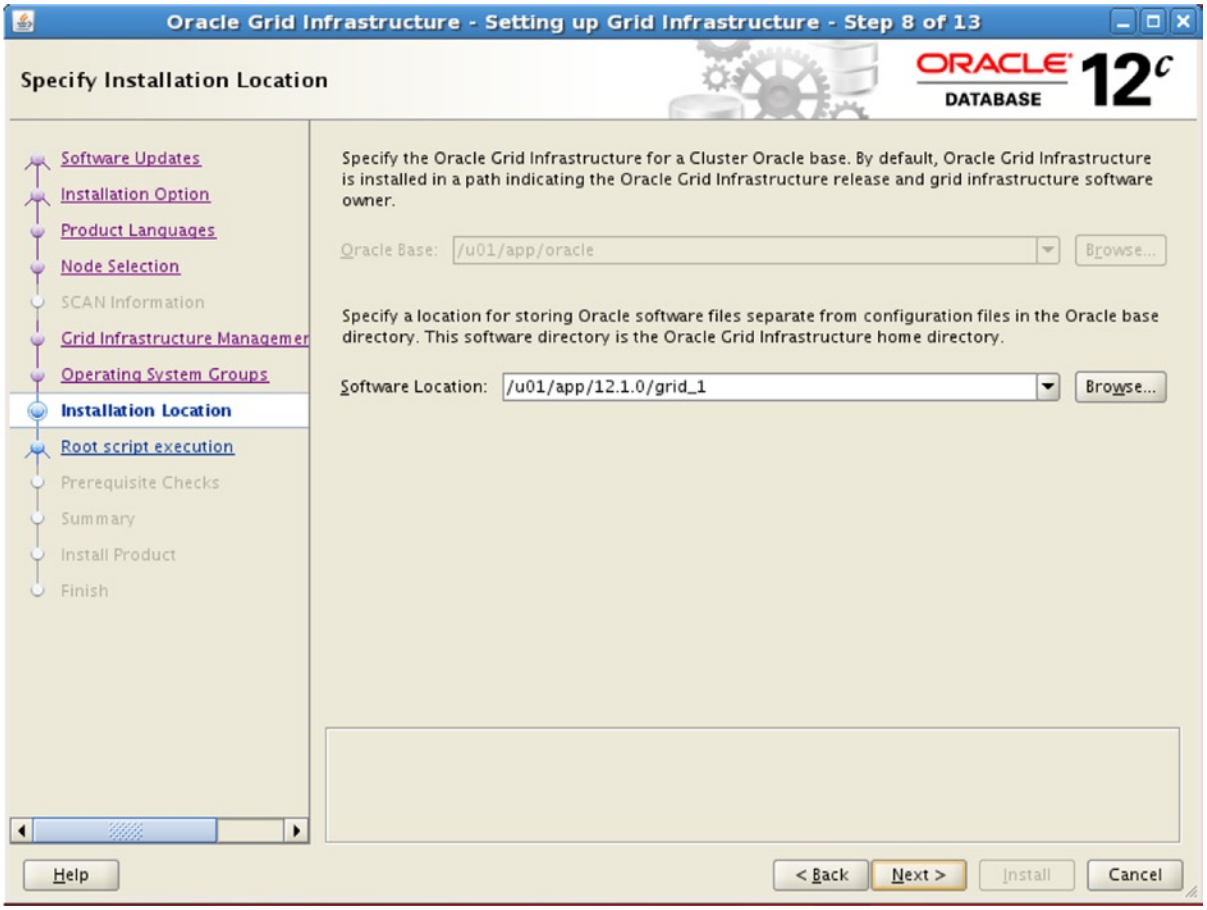


Figure 13-4. *Specify Installation Location screenshot*

8. With 11gR2, all upgrades are now performed as out-of-box upgrades, where you must input a different location for the new GI home in contrast to the existing GI home. Therefore, you must enter a different GI home location for 12c.

9. Starting in Oracle 12c, you have the flexibility to automate the `root.sh` or `rootupgrade.sh` execution across all nodes in the cluster during a new GI installation or upgrade deployment. You can prefer to automate the execution of the script either by providing the root user login credentials or by defining a `sudo` user to execute the script automatically. Automating the script execution eliminates the need for manual execution.

10. If you don't select the `Automatically run configuration scripts` option, you will have to run the script manually when prompted, as you used to do in earlier releases (Figure 13-5).

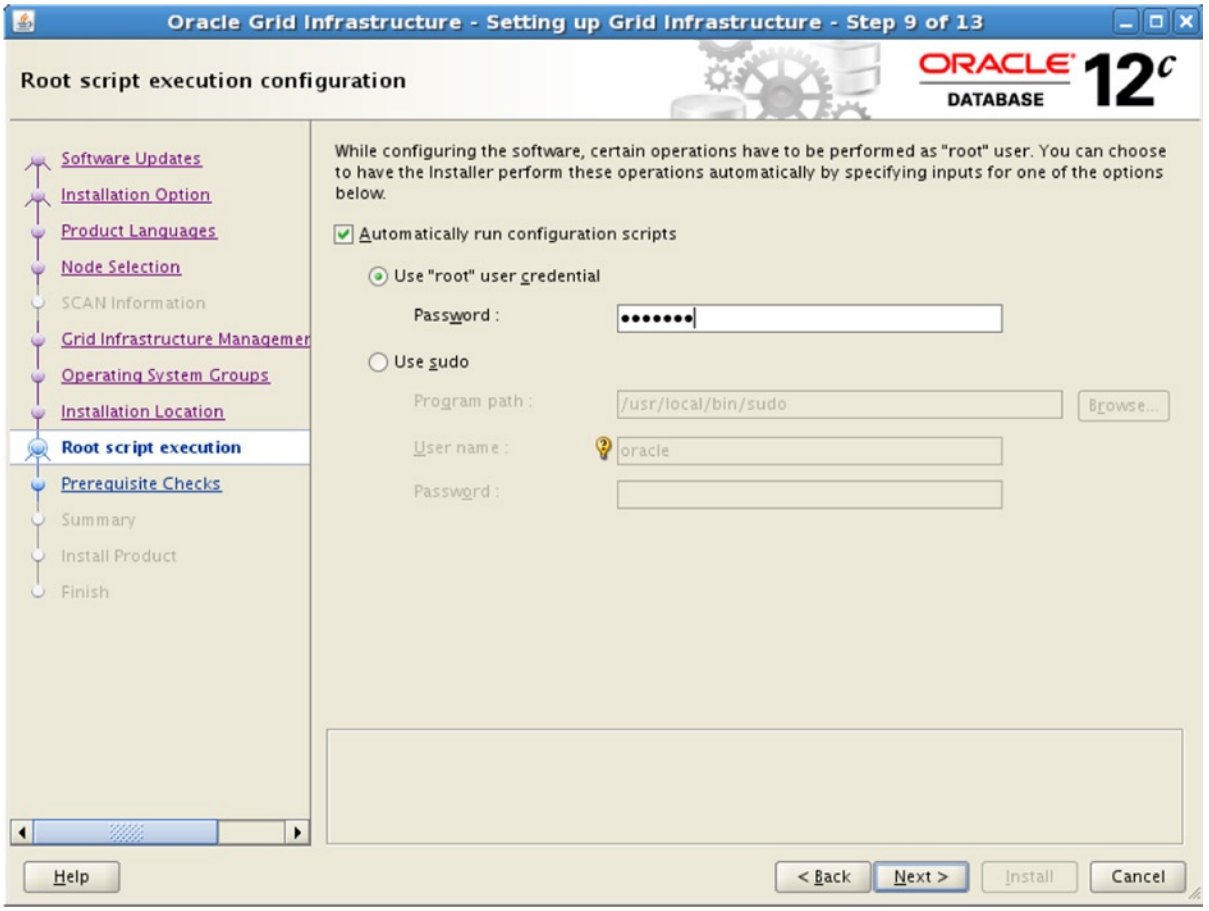


Figure 13-5. *Root script execution configuration screenshot*

11. We all know the possibilities of running the `root.sh` script in parallel to expedite the deployment when there are a huge number of nodes involved in the process. For example, in earlier Oracle releases, when we have, say, about six nodes as part of the deployment, we can initiate the script in parallel as follows:

    - First, run the script on the local node, that is, node 1.
    - Run the script in parallel on nodes 2 to 5.
    - Finally, execute the script on the last node, that is, node 6.

12. A similar behavior can be achieved in 12c now by defining the "pooling the nodes into batches" feature. You can scatter all your nodes into batches, up to a maximum of three batches. The Installer will then run the `root.sh/rootupgrade.sh` script on the nodes as mentioned in the batches. It will also ask for the confirmation before proceeding to the next batch. Figure 13-6 describes this.

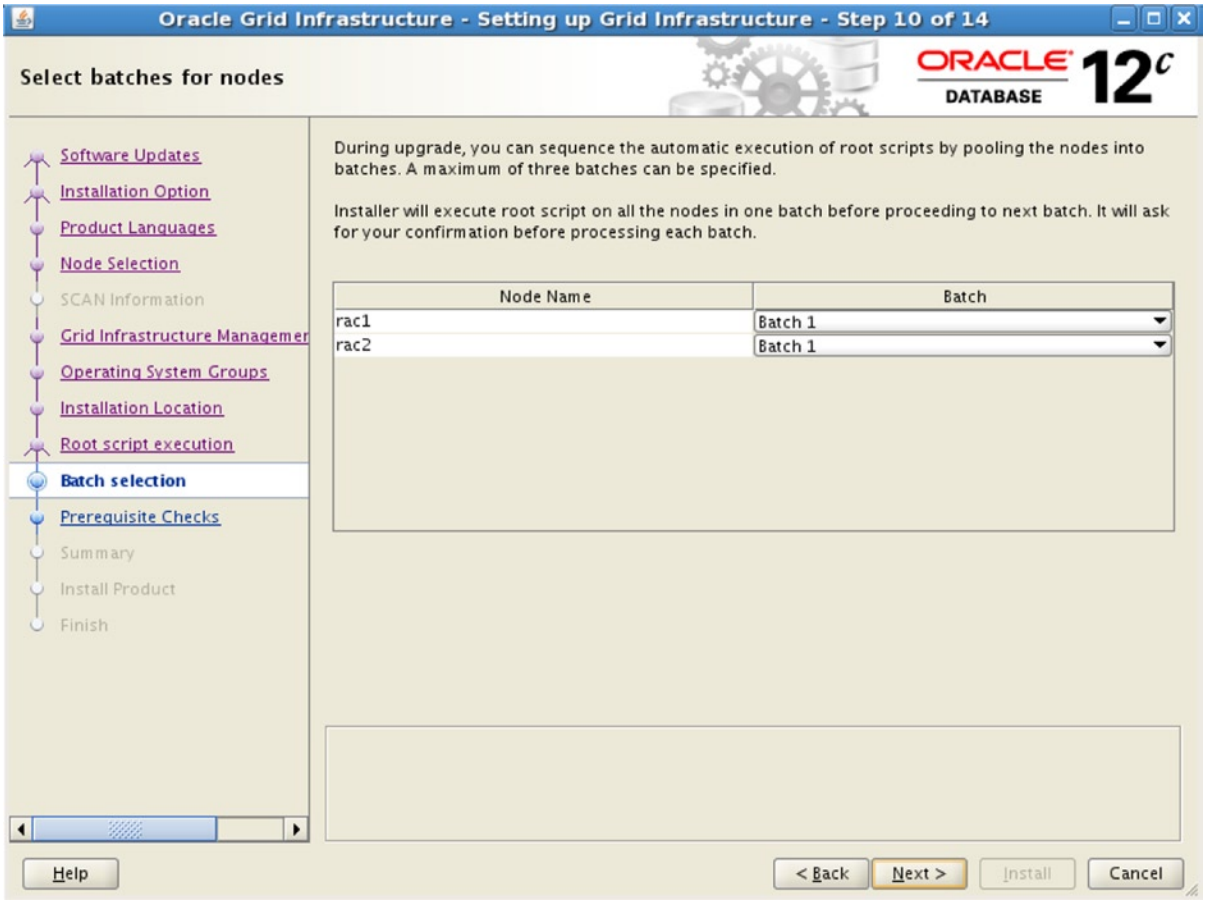


Figure 13-6. *Select batches for nodes screenshot*

13. The Installer then performs the prerequisite verification checks and displays the result on the `Prerequisite Checks` screen. Look for any verification failures and take appropriate action before moving forward.

14. Review the `Summary` report, and if you find anything colored red, take the appropriate steps and then proceed to the real action.

15. The Installer subsequently proceeds with the installation of Oracle 12c GI and other steps, as shown in Figure 13-7.

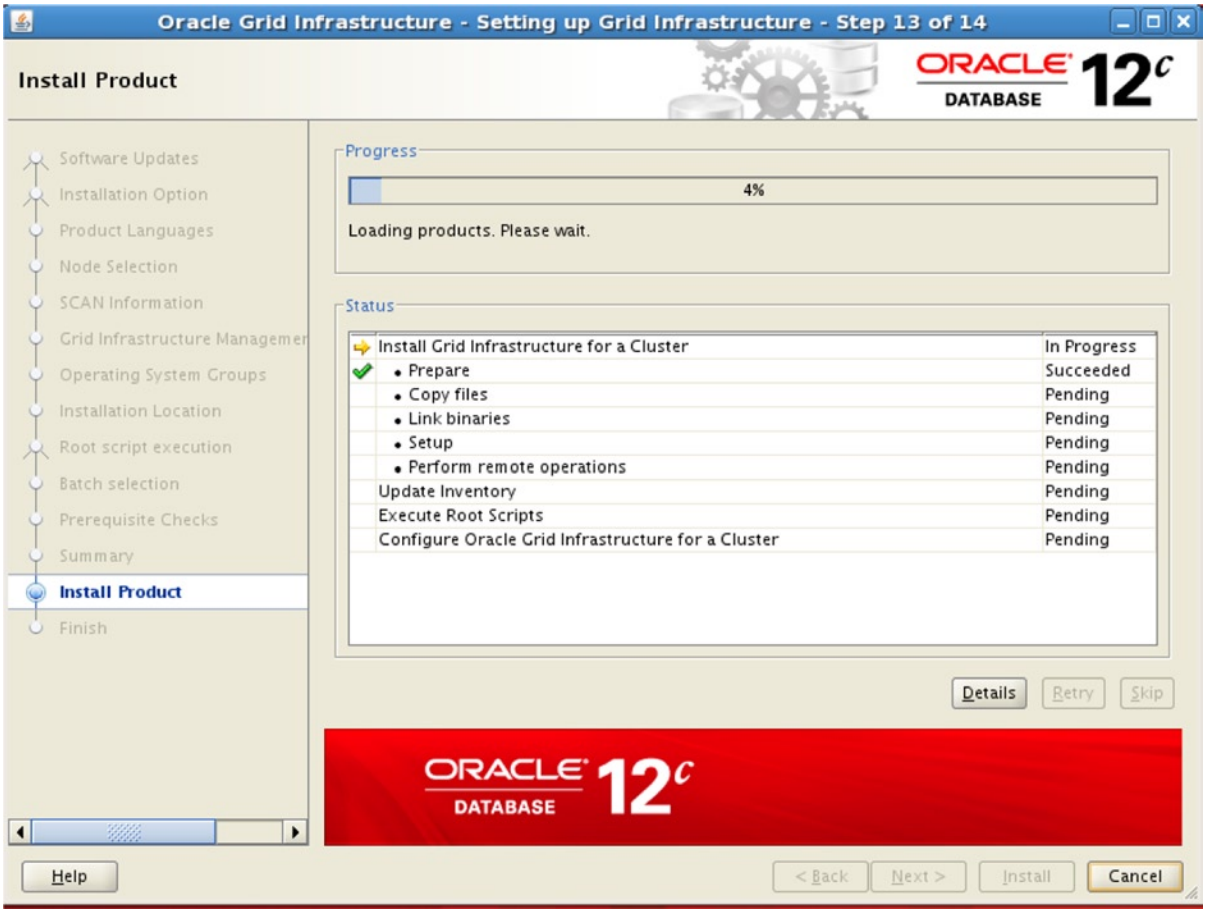


Figure 13-7. *Install Product screenshot*

16. Since we enabled automation of `root.sh` execution, you will have to respond with Yes to execute the root.sh automatically, as demonstrated in Figure 13-8.

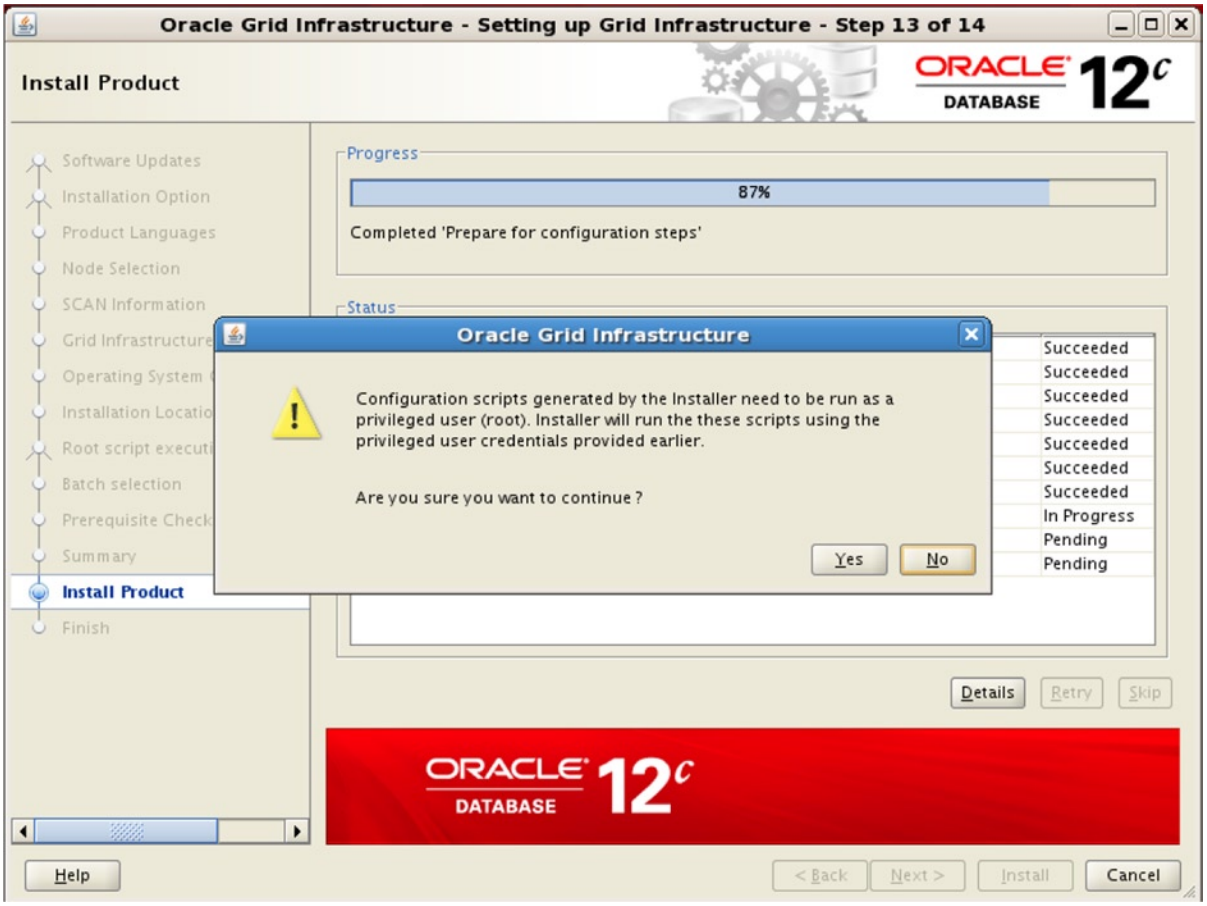


Figure 13-8. *Install Product root execution screenshot*

If you have opted to run the `root.sh/rootupgrade.sh` script manually, the screen shown in Figure 13-9 will appear.

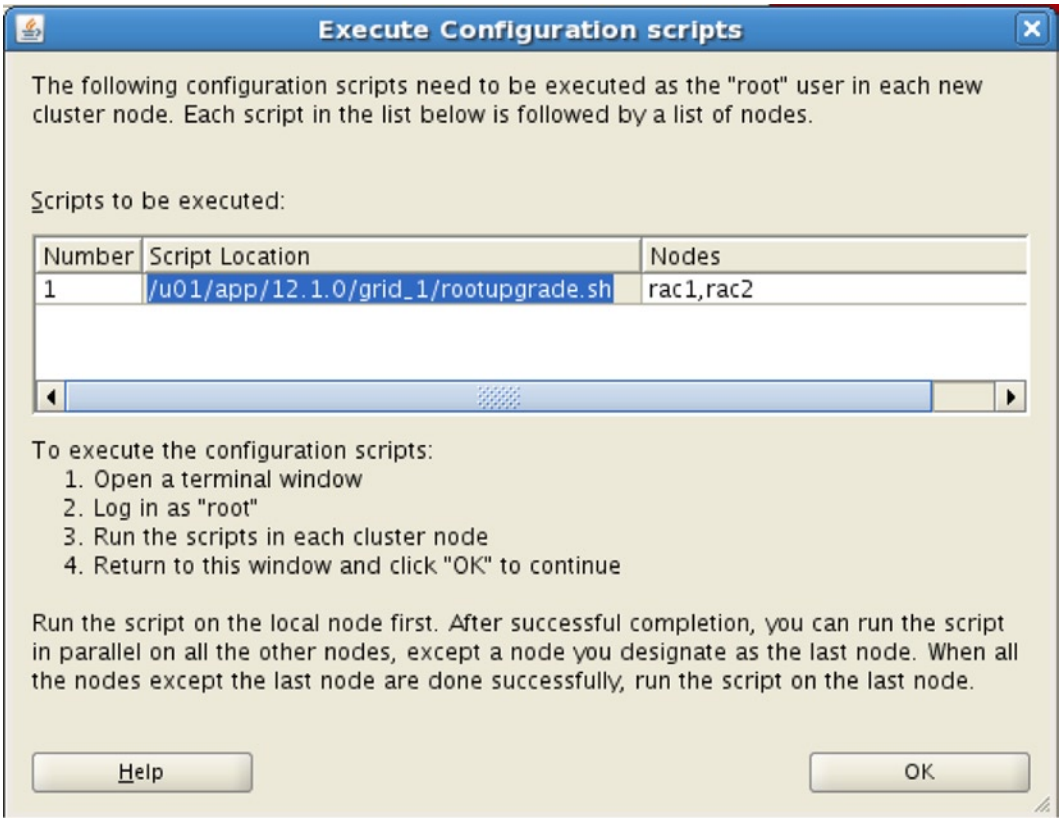


Figure 13-9. *rootupgrade script execution screenshot*

Based on your response or action, the Installer then proceeds with running the script on the nodes, as per the instructions (batch) specified previously. While the script is being executed on the respective nodes, you can see the feedback on the `Install Product` screen on which node the script is currently being executed.

After the completion of the script on the local node, from the other command prompt on the local node, you can query the CRS active/software versions. You will notice that the `crssoftwareversion` will be shown as the new version; however, the `crsactiveversion` will remain to previous release. This is the expected result: until the script doesn't complete on all nodes, the crsactiveversion will remain to previous release.

```
$ ./crsctl query crsactiveversion
Oracle Clusterware active version on the cluster is [11.2.0.3.0]

$ ./crsctl query crssoftwareversion
Oracle Clusterware version on node [rac1] is [12.1.0.0.2]
```

After successfully completing the execution of `rootupgrade.sh` on all nodes, the Installer will then proceed forward in configuring the GI management database, if selected, where a new database will be configured in the context. This can be observed in Figure 13-10.

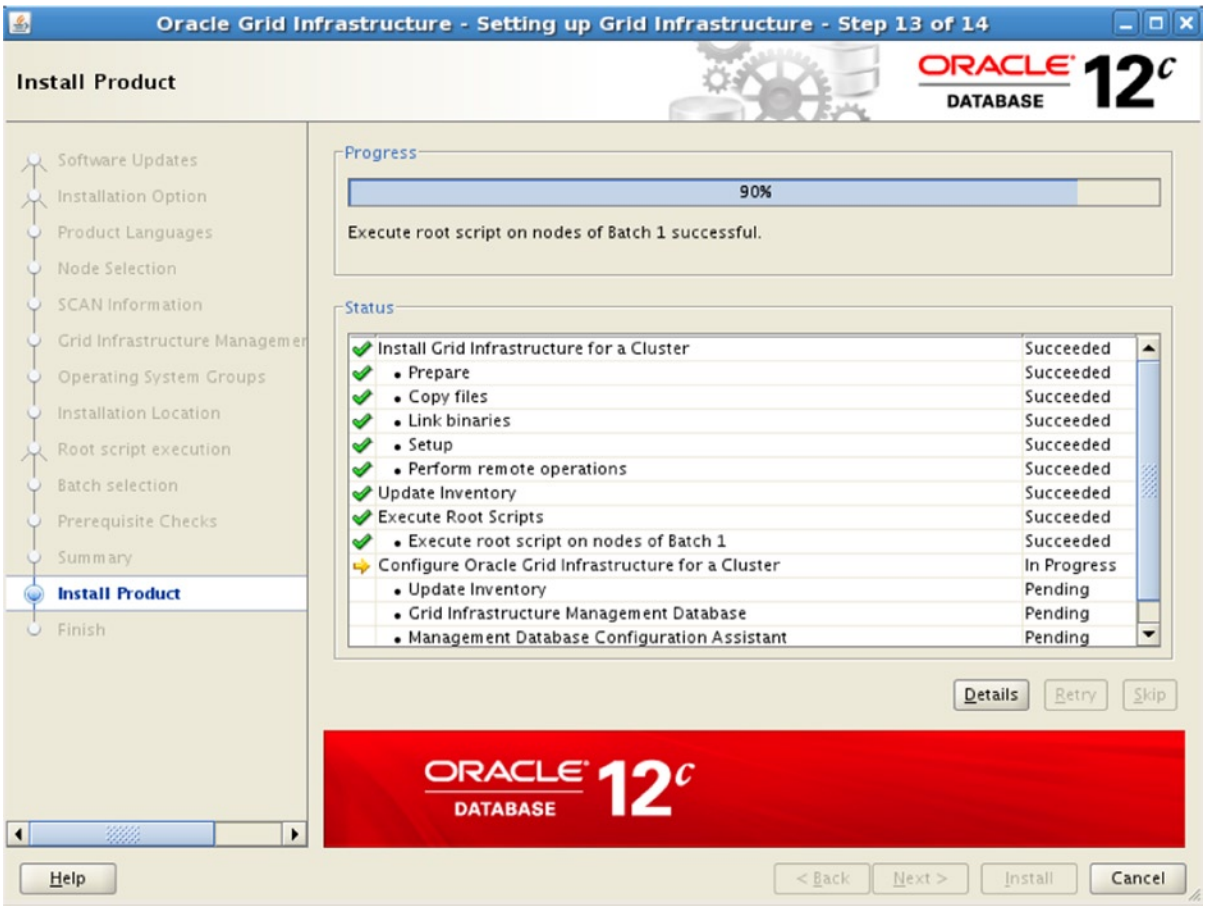


Figure 13-10. *Progress bar screenshot*

17. Complete the upgrade procedure when the status bar shows 100% completion. You can then verify the active and software version using the following examples. You must see both outputs mentioning the same cluster version, that is, 12.1.0.0.2 in our example.

```
$ ./crsctl query crs activeversion
Oracle Clusterware active version on the cluster is [12.1.0.0.2]

$ ./crsctl query crs softwareversion
Oracle Clusterware version on node [rac1] is [12.1.0.0.2]
```

The Importance of the Rootupgrade.sh Script

Only software binaries are deployed under the respective homes on the local and remote nodes until the `rootupgrade.sh` script is executed. The `rootupgrade.sh` actually performs the core Clusterware stack configuration and upgrade. In the context, the script does the following core duties:

On the local node (first node):

1. ASM upgrade will be performed.
2. OLR will be configured accordingly.
3. The cluster auto start entries in the /etc/inittab file on the OS will be modified.
4. The existing cluster stack will be shut down.
5. The Oracle 12c cluster stack on the local node will be started up.
6. The OCR will be configured with necessary information before displaying the success Cluster configuration message.

The following output is expected when the rootupgrade.sh script is executed on the local and other nodes of the cluster, except the last node in the cluster:

```
[root@rac1 oracle]# /u01/app/12.1.0/grid_1/rootupgrade.sh
Performing root user operation for Oracle 12c

The following environment variables are set as:
    ORACLE_OWNER= oracle
    ORACLE_HOME=  /u01/app/12.1.0/grid_1

Enter the full pathname of the local bin directory: [/usr/local/bin]:
The file "dbhome" already exists in /usr/local/bin.  Overwrite it? (y/n)
[n]:
The file "oraenv" already exists in /usr/local/bin.  Overwrite it? (y/n)
[n]:
The file "coraenv" already exists in /usr/local/bin.  Overwrite it? (y/n)
[n]:

Entries will be added to the /etc/oratab file as needed by
Database Configuration Assistant when a database is created
Finished running generic part of root script.
Now product-specific root actions will be performed.
Using configuration parameter file: /u01/app/12.1.0/grid_1/crs/install/crsconfig_params
2013/04/12 10:07:18 CLSRSC-363: User ignored prerequisites during installation

ASM upgrade has started on first node.

OLR initialization - successful
2013/04/12 10:12:29 CLSRSC-329: Replacing Clusterware entries in file '/etc/inittab'

CRS-4133: Oracle High Availability Services has been stopped.
CRS-4123: Oracle High Availability Services has been started.
CRS-2673: Attempting to stop 'ora.drivers.acfs' on 'rac1'
CRS-2677: Stop of 'ora.drivers.acfs' on 'rac1' succeeded
2013/04/12 10:17:24 CLSRSC-343: Successfully started Oracle clusterware stack
```

```
clscfg: EXISTING configuration version 5 detected.
clscfg: version 5 is 11g Release 2.
Successfully accumulated necessary OCR keys.
Creating OCR keys for user 'root', privgrp 'root'..
Operation successful.
2013/04/12 10:20:34 CLSRSC-325: Configure Oracle Grid Infrastructure for a Cluster ... succeeded
```

On the last node:

1. OLR will be configured accordingly.
2. Modify the cluster auto start entries in the /etc/inittab file on the OS.
3. Existing cluster stack will be shut down.
4. Start-up the Oracle 12c cluster stack on the local node.
5. Configure the OCR with necessary information.
6. Starts the actual OCR,CSS,ASM & CRS process.

The following output is expected when the `rootupgrade.sh` script is executed on the last node in the cluster:

```
[root@rac2 ~]# /u01/app/12.1.0/grid_1/rootupgrade.sh
Performing root user operation for Oracle 12c

The following environment variables are set as:
    ORACLE_OWNER= oracle
    ORACLE_HOME=  /u01/app/12.1.0/grid_1

Enter the full pathname of the local bin directory: [/usr/local/bin]:
The file "dbhome" already exists in /usr/local/bin.  Overwrite it? (y/n)
[n]:
The file "oraenv" already exists in /usr/local/bin.  Overwrite it? (y/n)
[n]:
The file "coraenv" already exists in /usr/local/bin.  Overwrite it? (y/n)
[n]:

Entries will be added to the /etc/oratab file as needed by
Database Configuration Assistant when a database is created
Finished running generic part of root script.
Now product-specific root actions will be performed.
Using configuration parameter file: /u01/app/12.1.0/grid_1/crs/install/crsconfig_params
2013/04/12 10:22:05 CLSRSC-363: User ignored prerequisites during installation

OLR initialization - successful
2013/04/12 10:25:18 CLSRSC-329: Replacing Clusterware entries in file '/etc/inittab'

CRS-4133: Oracle High Availability Services has been stopped.
CRS-4123: Oracle High Availability Services has been started.
CRS-2673: Attempting to stop 'ora.drivers.acfs' on 'rac2'
CRS-2677: Stop of 'ora.drivers.acfs' on 'rac2' succeeded
2013/04/12 10:29:28 CLSRSC-343: Successfully started Oracle clusterware stack
```

```
clscfg: EXISTING configuration version 5 detected.
clscfg: version 5 is 11g Release 2.
Successfully accumulated necessary OCR keys.
Creating OCR keys for user 'root', privgrp 'root'..
Operation successful.
Start upgrade invoked..
Started to upgrade the Oracle Clusterware. This operation may take a few minutes.
Started to upgrade the OCR.
Started to upgrade the CSS.
The CSS was successfully upgraded.
Started to upgrade Oracle ASM.
Started to upgrade the CRS.
The CRS was successfully upgraded.
Successfully upgraded the Oracle Clusterware.
Oracle Clusterware operating version was successfully set to 12.1.0.0.2
2013/04/12 10:35:20 CLSRSC-325: Configure Oracle Grid Infrastructure for a Cluster ... succeeded
```

Post-Upgrade Tasks

After a Clusterware upgrade, quickly go through the following post-upgrade checklist to confirm success:

1. Verify Cluster active/software versions.
2. Ensure that the Clusterware daemons are started from the new Oracle 12c home, as shown in Figure 13-11.

```
[oracle@rac1 ~]$ ps -ef |grep d.bin
root      6228     1  0 01:44 ?        00:00:20 /u01/app/12.1.0/grid_1/bin/ohasd.bin reboot
oracle    6376     1  0 01:45 ?        00:00:06 /u01/app/12.1.0/grid_1/bin/evmd.bin
oracle    6379     1  0 01:45 ?        00:00:00 /u01/app/12.1.0/grid_1/bin/mdnsd.bin
oracle    6394     1  0 01:45 ?        00:00:09 /u01/app/12.1.0/grid_1/bin/gpnpd.bin
oracle    6428     1  0 01:45 ?        00:00:41 /u01/app/12.1.0/grid_1/bin/gipcd.bin
oracle    6713     1  0 01:46 ?        00:01:44 /u01/app/12.1.0/grid_1/bin/ocssd.bin
root      6976     1  0 01:49 ?        00:00:05 /u01/app/12.1.0/grid_1/bin/octssd.bin
root      7433     1  0 01:52 ?        00:00:16 /u01/app/12.1.0/grid_1/bin/crsd.bin reboot
```

Figure 13-11. *12c cluster output screenshot*

3. Verify the status of all cluster-critical resources using the following example:

```
$ ./crsctl stat res -t -init
--------------------------------------------------------------------------------
NAME           TARGET  STATE        SERVER                   STATE_DETAILS
--------------------------------------------------------------------------------
Cluster Resources
--------------------------------------------------------------------------------
ora.asm
      1        ONLINE  ONLINE       rac1                     Started
ora.cluster_interconnect.haip
      1        ONLINE  ONLINE       rac1
```

```
ora.crsd
      1        ONLINE  ONLINE       rac1
ora.cssd
      1        ONLINE  ONLINE       rac1
ora.cssdmonitor
      1        ONLINE  ONLINE       rac1
ora.ctssd
      1        ONLINE  ONLINE       rac1                     ACTIVE:0
ora.diskmon
      1        OFFLINE OFFLINE
ora.drivers.acfs
      1        ONLINE  ONLINE       rac1
ora.evmd
      1        ONLINE  ONLINE       rac1
ora.gipcd
      1        ONLINE  ONLINE       rac1
ora.gpnpd
      1        ONLINE  ONLINE       rac1
ora.mdnsd
      1        ONLINE  ONLINE       rac1
ora.storage
      1        ONLINE  ONLINE       rac1
```

4. Modify all existing scripts to reflect the new Oracle 12c GI home settings; for example, in the `.bash_profile`, other manual scripts that you may maintain at your premises.
5. Perform the filesystem backup of the new GI home.
6. Perform the manual physical backup of the OCR.
7. Optionally deconfigure and clean the previous Cluster.

Clusterware Downgrade

Whenever you prepare to introduce a new change to the existing environment, it is always a recommended and safe approach to have a complete backout plan to recover from any disasters that may occur after the change. If you work for a reputable organization, the management naturally expects a back-out plan to accompany all changes to the current environment. The back-out plan is a procedure to roll back the changes that were applied if the change didn't go well, or the change creates new problems.

The objective of this section is to summarize all the necessary steps that are required to downgrade from a successfully upgraded or from a failed Clusterware/ASM to earlier versions.

Oracle's out-of-place upgrade option makes the downgrade procedure faster by keeping the old and new versions of Oracle Clusterware software in different homes/locations. This makes the downgrade procedure a very light and quick process, in contrast with pre-11gR2 versions.

Initiating Downgrade Procedure

In order to kick-start the downgrade procedure, perform the following steps:

From the new 12c Grid Home, run the following command on all nodes of the cluster sequentially, preferably starting from the first node to the last node:

```
$GRID_HOME/crs/install/rootcrs.pl -downgrade
```

Use the [-force] option in case if you encounter any issues with the preceding command due to the failed upgrade.

The command will stop all currently running resources on the server and shut down the 12c cluster stack subsequently.

After successfully executing the preceding command across all nodes in the cluster, execute the following command on the first node:

```
$GRID_HOME/crs/install/rootcrs.pl –downgrade -lastnode
```

This process will downgrade the OCR and set to the previous release.

As GI installation owner, run the following commands from the 12c $GRID_HOME/oui/bin location on every successfully upgraded node to complete the downgrade procedure:

```
./runInstaller -nowait -waitforcompletion -ignoreSysPrereqs -updateNodeList -silent CRS=false
ORACLE_HOME=/u01/app/12.1.0/grid

./runInstaller -nowait -waitforcompletion -ignoreSysPrereqs -updateNodeList -silent CRS=false
ORACLE_HOME=/u01/app/11.2.0/grid
```

■ **Note** You need to provide the 12c Grid Home location in the first command, and in the second command, you need to provide the previous release Grid Home location.

For Oracle version 11gR2 or later, after successfully executing the preceding steps, as the root user, bring up the previous release cluster stack using the following command on each node in the cluster:

```
$/u01/app/11.2.0/grid/bin/crsctl start crs
```

For pre–Oracle 11gR2 versions, as root user, execute the root.sh script manually from the earlier Cluster release on each node in sequence to bring up the previous cluster stack.

Oracle 12c GI supports downgrade option to Oracle 11gr2 and Oracle 11gR1.

■ **Note** Keep in mind that any pre-/post-upgrade configuration modifications performed will be removed and unrecoverable after the downgrade.

Forcing Upgrade—When Some Nodes Are Inaccessible

Pre–Oracle 12c, if some nodes are not accessible or become unreachable before or during the upgrade, then the Clusterware upgrade won't be completed due to missing/inactive nodes in the cluster. In contrast, with the introduction of the –force option with the rootupgrade.sh script, you can perform a forced upgrade despite missing/inactive nodes in the cluster. Therefore, when you query active crs version to confirm the upgraded, you will see that the crs active version is set to the new Cluster version in contrast to the pre–Oracle 12c releases.

Here is an example to run the rootupgrade.sh script with the –force option:

```
$ /u01/app/12.1.0/grid/rootupgrade –force
```

When the inactive nodes become accessible after a forced upgrade, you can join them to the cluster to either upgrade or delete them from the cluster. Execute the following examples on the first node of the cluster to join the node that was not accessible during the course of the upgrade:

```
$/u01/app/12.1.0/grid/crs/install/rootcrs.pl -join -existingNode rac1 upgrade_node rac2
```

In the preceding example, `existingNode` indicates the node on which the upgrade was successful, and `upgrade_node` is the node which was not reachable during the course of upgrade and wants to join the node in the cluster after a forced upgrade.

Installing Oracle 12c RDBMS Software

You will have to install the new Oracle 12c RDBMS binaries into a new RDBMS location right after your Clusterware upgrade to ally with the complete Clusterware environment upgrade procedure.

As stated earlier on, from 11gR2 onward, all upgrades are now out-of-place upgrades; you will have to create a new location in the context for the Oracle 12c binaries and install the software. This procedure will not have any impact on the active databases on the respective nodes unless you select the "Install database software only" option. In other words, we advise you not to select the "Upgrade an existing database option" on the Select Installation Option screen, as shown in Figure 13-12.

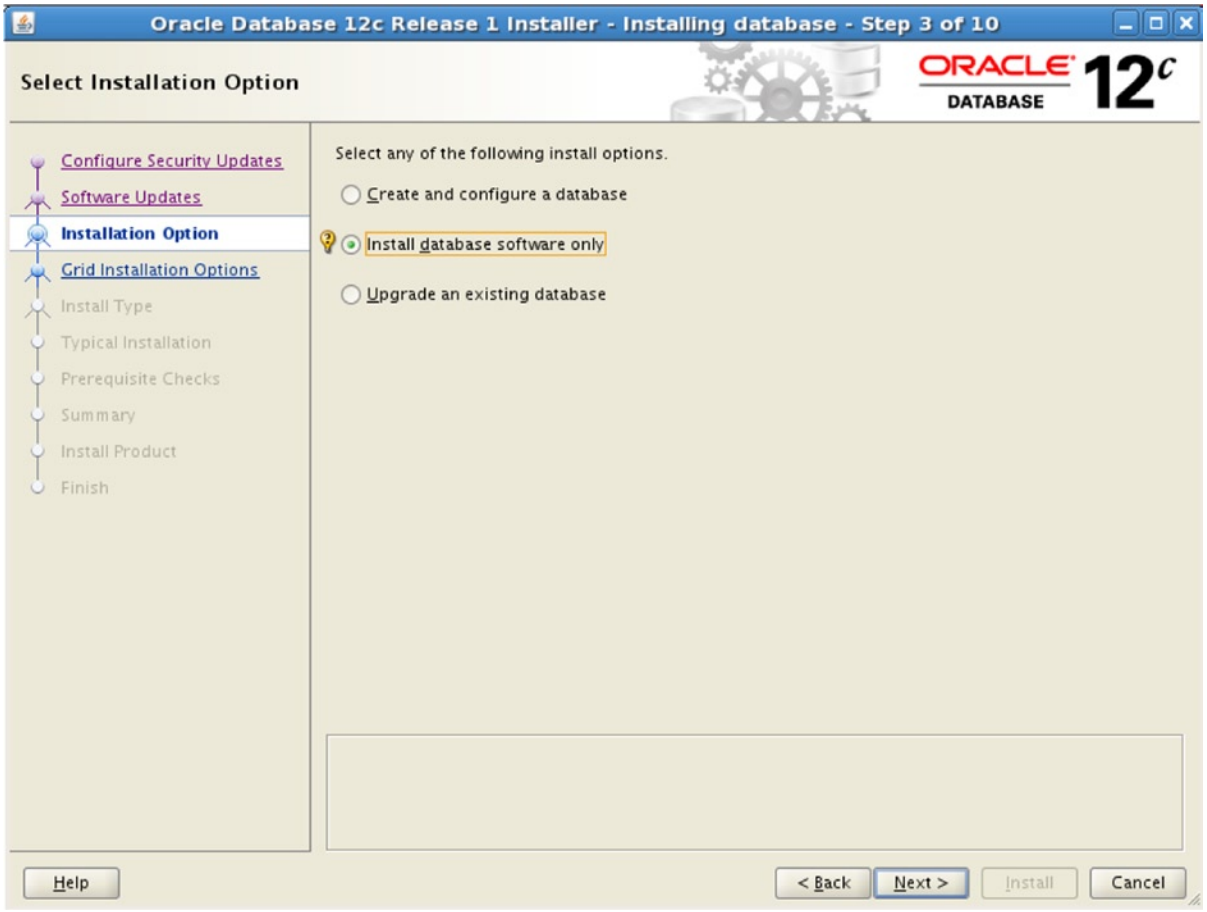


Figure 13-12. *Database Select Installation Option screenshot*

You will have to follow the typical RDBMS software installation in the RAC environment and complete the software installation considering the preceding recommendations discussed a little earlier.

Once you successfully complete the software installation, you may then proceed with the database upgrade process as per your schedule and convenience. Manual and Database Upgrade Assistant (DBUA) upgrade methods will be demonstrated in more detail in the subsequent section.

Database Upgrade

Although there are various methods, processes, and tools to perform an upgrade of an existing database to the new Oracle release, this section mainly focuses on the manual and DBUA upgrade methods. Just before the upgrade, it is important for you as a DBA to become familiar with those key enhancements brought into the upgrade procedure of the new release for a successful database upgrade. The new enhancements with regard to the database upgrade include the following:

Pre-upgrade tool: The new `preupgrd.sql` script in Oracle database 12c replaces the previous Oracle release pre-upgrade tool, `utlu112i.sql` script, which will provide recommendations to fix any pre– and post–database upgrade chaos that might emerge and cause disruption in the upgrade process. Therefore, pre–database upgrade, you must execute the script from the Oracle 12c home on the source database to gather the recommendations for pre– and post–database upgrade. The DBUA tool uses this script by default.

Parallel Upgrade Utility (catctl.pl): Unlike earlier releases of Oracle, with the introduction of the new Parallel Upgrade Utility in Oracle 12c, you can now make full use of the system's CPU capacity by initiating the database upgrade script, `catupgrd.sql`, in parallel. The new Parallel Upgrade Utility (`catctl.pl`) with Oracle 12c greatly improves the database upgrade runtime by executing the `catupgrd.sh` script in parallel. The DBUA tool takes full advantage of this utility by running it by default, as the utility is integrated with the DBUA tool. The following example summarizes the use of the utility when you do a manual database upgrade:

```
$ORACLE_HOME/per/bin/perl catctl.pl –n 2 –l /tmp catupgrd.sql
```

The `-n` parameter indicates the range of parallelism: the value must be 0-8.

`-l` is the location where the logs will be spooled.

Restart DBUA database upgrades: Unlike the earlier Oracle releases, the DBUA tool in Oracle 12c has the ability to resume the database upgrade process from the point where it failed without being started over again. The new RMAN feature in 12c, `Guaranteed Restore Point`, in this context will help in resuming the failed database upgrades initiated by the DBUA tool.

Database upgrade compatibility matrix: Before you start upgrading the existing database to the new release, it is essential to determine whether the new release will support a direct upgrade of an existing database version or not. Table 13-2 outlines the database upgrade compatibility matrix of various Oracle releases that support direct and indirect database upgrades to Oracle 12c.

Table 13-2. *Database Upgrade Compatibility Matrix*

Direct Database Upgrade	Indirect Database Upgrade
10.2.0.5	9.2.0.8 or earlier
11.1.0.7	10.1.0.5
11.2.0.2 or later	10.2.0.2,10.0.2.3,10.0.2.4
	11.1.0.6
	11.2.0.1

For example, if the existing Database version is 11.2.0.1, you need to upgrade it to 11.2.0.2 or higher first and then proceed with 12c upgrade.

Deploying Manual Database Upgrade

In this section, you will learn all the mandatory steps that are required to perform a manual database upgrade to Oracle 12c. The following demonstrates the procedure to upgrade an Oracle 11.2.0.3 database, named PRDDB, to Oracle 12c:

Back up the source database, if no recent backup exists.

Connect to the source database as/sysdba and execute the pre-upgrade script from the Oracle 12c home. This is a mandatory step when you perform a manual database upgrade; on the flip side, if the script doesn't run, the post–database upgrade step `ultrp.sql` script will be failed. Figure 13-13 shows how to run the pre-upgrade script on the source database:

```
export ORACLE_SID=PRDDB1
SQL> sqlplus / as sysdba
SQL> @/u01/app/oracle/product/12.0.1/rdbms/admin/preupgrd.sql
```

You are likely to see the following output on running the pre-upgrade tool:

```
SQL> @/u01/app/oracle/product/12.1.0/db_1/rdbms/admin/preupgrd.sql
Loading Pre-Upgrade Package...
No errors.
No errors.
old  23:   IF UPPER('&&1') = 'TERMINAL' THEN
new  23:   IF UPPER('') = 'TERMINAL' THEN
old  25:   ELSIF ( '&&1'IS NULL OR UPPER('&&1') = 'FILE') THEN
new  25:   ELSIF ( ''IS NULL OR UPPER('') = 'FILE') THEN
old  29:   IF UPPER('&&2') = 'XML' THEN
new  29:   IF UPPER('') = 'XML' THEN
old  31:   ELSIF ( '&&2'IS NULL OR UPPER('&&2') = 'TEXT') THEN
new  31:   ELSIF ( ''IS NULL OR UPPER('') = 'TEXT') THEN
Executing Pre-Upgrade Checks...
Pre Upgrade Log and fix up scripts located at: /u01/app/oracle/cfgtoollogs/PRDDB/preupgrade/
Pre-Upgrade Check Complete.
```

Figure 13-13. *Pre-upgrade screenshot*

Unlike earlier Oracle database releases, the pre-upgrade output is written to a log file. You will have to review the `preupgrd.log` file generated by the pre-upgrade tool for any warnings and recommendations. Similarly, review and apply the recommendations stated in the `preupgrade_fixups.sql` and`postupgrade_fixups.sql` scripts to fix any pre– and post–database upgrade issues recommended by the pre-upgrade tool on the source database, for example, collecting dictionary and fixed-objects statistics, database initialization parameter adjustments, invalid object details, tablespace size adjustments, etc. The log file and script files will typically be created under the `$ORACLE_BASE/cfgtoolslogs/PRDDB/preupgrade` location. If no $ORACLE_BASE variable is set on the server, the files in the context will be created under the `$ORACLE_HOME/cfgtoolslogs/PRDDB/preupgrade` location. The DBUA tool uses the pre-upgrade tool by default. The following shows the existence of the files under the $ORACLE_HOME location:

```
-rw-r--r-- 1 oracle dba 1444 Apr 13 09:34 postupgrade_fixups.sql
-rw-r--r-- 1 oracle dba 2491 Apr 13 09:34 preupgrade_fixups.sql
-rw-r--r-- 1 oracle dba 6397 Apr 13 09:34 preupgrade.log
```

Another important task while performing a manual database upgrade is to copy the source database password file, init/spfile, from the Oracle 11g home to the Oracle 12c Database home. Also, ensure that the local/remote listener parameter TNS configuration details are added in the new Oracle 12c tnsnames.ora file to avoid database startup failures when database is started from the Oracle 12c home. For a RAC database, you must do it for all instances. The following example demonstrates the copy command on Linux OS:

```
$cp /u01/app/oracle/product/11.2.0/db_1/dbs/orapwPRDDB1 /u01/app/oracle/product/12.1.0/db_1/dbs
```

If required, run the `preupgrade_fixups.sql` script and subsequently shut down and mount the database to disable the archivelog mode to avoid excessive archive generation during upgrade and to expedite the upgrade process.

```
srvctl stop database –d PRDDB

export ORACLE_SID=PRDDB1

sqplus / as sysdba

SQL> STARTUP MOUNT
SQL> ALTER DATABASE NOARCHIVELOG;
SQL> ALTER SYSTEM SET CLUSTER_DATABASE=FALSE SCOPE=SPFILE;
SQL> SHUTDOWN IMMEDIATE;
```

Set the `$ORACLE_HOME` to Oracle Database 12c version on the server.

```
export ORACLE_HOME=/u01/app/oracle/product/12.1.0/db_1
export PATH=$ORACLE_HOME/bin:$PATH
cd $ORACLE_HOME
```

On the SQL prompt, run the following database upgrade command:

```
SQL> CONNECT / AS SYSDBA
SQL> STARTUP UPGRADE
```

Exit from the SQL prompt and subsequently initiate the upgrade process in parallel mode, using the new Parallel Upgrade Utility, as explained here:

```
$/u01/app/oracle/product/12.0.1/db_1/perl/bin/perl catctl.pl–n 2 –l /tmp catupgrd.sql
```

If you prefer to perform an upgrade in a serial mode, like in previous releases, do the following at the SQL prompt:

```
SQL>@?/rdbms/admin/catupgrd.sql parallel=no
```

Database will be automatically shut down once the preceding command successfully completes. You therefore have to perform the following steps to enable the CLUSTER_DATABASE, set the database COMPATIBLE parameter to 12c, enable archive log, etc.:

```
SQL> STARTUP MOUNT
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER SYSTEM SET CLUSTER_DATABASE=TRUE SCOPE=SPFILE;
SQL> ALTER SYSTEM SET COMPATIBLE=12.1.0 scope=spfile;
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP
```

```
SQL> execute dbms_stats.gather_fixed_objects_stats;
SQL> @/u01/app/oracle/prdoduct/12.1.0/rdbms/admin/utlrp.sql
SQL> @/u01/app/oracle/product/12.1.0/rdbms/admin/utlobj.sql
SQL> @/u01/app/oracle/product/12.1.0/rdbms/admin/utlu121s.sql
SQL> SHUTDOWN IMMEDIATE;
SQL> EXIT

utlrp.sql             recomiples all invalid objects on the database
ultobj.sql             verfies the validity of all packages/classes on the database
utlu121s.sql             displays database upgrade summary
```

After successfully completing the database upgrade process, you will have to modify the cluster database configuration details in the OCR using the following command:

```
srvctl upgrade database –d PRODB –o /u01/app/oracle/product/12.1.0/db_1
```

At this point in time, your cluster database is successfully upgraded to 12c. Now, open the database using the following cluster command, which in turn will start all RAC database instances:

```
srvctl start database –d PRODB
```

■ **Note** Executing the `utlrp.sql` is mandatory to avoid performance degradation when accessing the data dictionary objects after the database upgrade completion. If the script doesn't run right after the database upgrade completion, whenever the data dictionary is being accessed first, it will have a significant performance impact. Therefore, it is advised to run the script after the upgrade.

Post-Database Upgrade Steps

Once the database is successfully upgraded to Oracle 12c, go through the following recommended post–database upgrade steps:

- Run the `postupgrade_fixups.sql` script as explained earlier
- Back up the database
- Also migrate the source database listener to Oracle 12 by dropping and re-creating the existing listener
- Adjust any RMAN backups, cron tab or other scripts, environment, profile variables, etc., to reflect the new Oracle Home settings
- Ensure that the database points to the new Oracle Home in the `oratab` file
- Adjust the `ATTRIBUTE 'compatible.asm'='12.1'` to the ASM diskgroups to make use of new ASM features
- Verify database configuration details with the following:

```
$ ./srvctl config database –d PRODB
```

Database Upgrade Using the DBUA

After learning a manual database upgrade procedure, it's time now to shift the focus to know how to perform a database upgrade using the DBUA tool in Oracle 12c. In addition to portraying the DBUA database upgrade procedure with a bunch of screenshots, this section also going to explain why DBUA is a preferred method over other methods to perform a RAC database procedure.

The step-by-step procedure that outlines database upgrade with DBUA tool is as follows:

Launch ./duba from the Oracle 12c home.

Select the `Upgrade Oracle database` option on the `Select Operation` page, as shown in Figure 13-14.

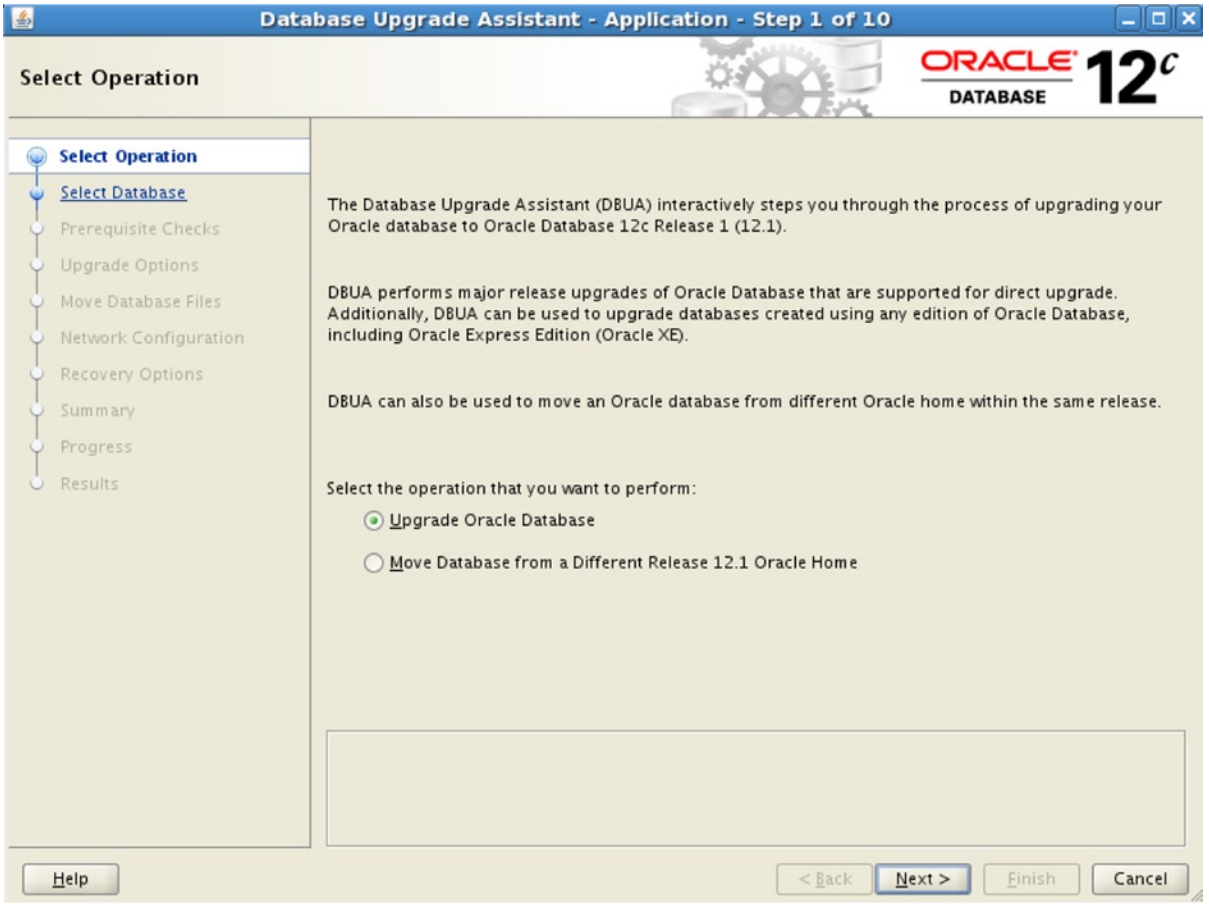


Figure 13-14. *Select Operation screenshot*

Select the database from the given database list on the `Select Database` page, as shown in Figure 13-15.

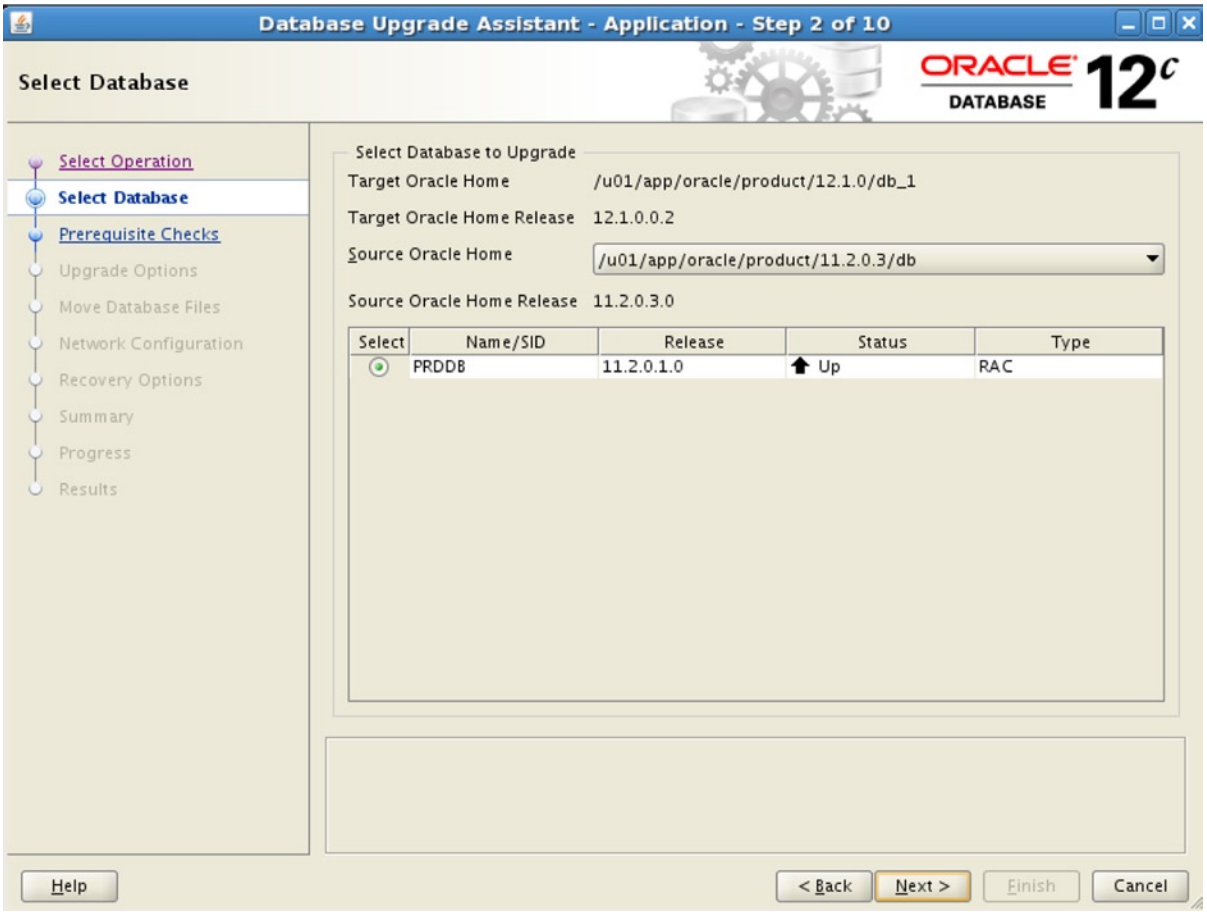


Figure 13-15. *Select Database screenshot*

Prerequisite checks are carried out and the result will be displayed. You will have an option to `Ignore, Fix or Revalidate` any prerequisite failures on this page. Also, you can run the prerequisite checks by clicking the Check Again button on the top right (Figure 13-16).

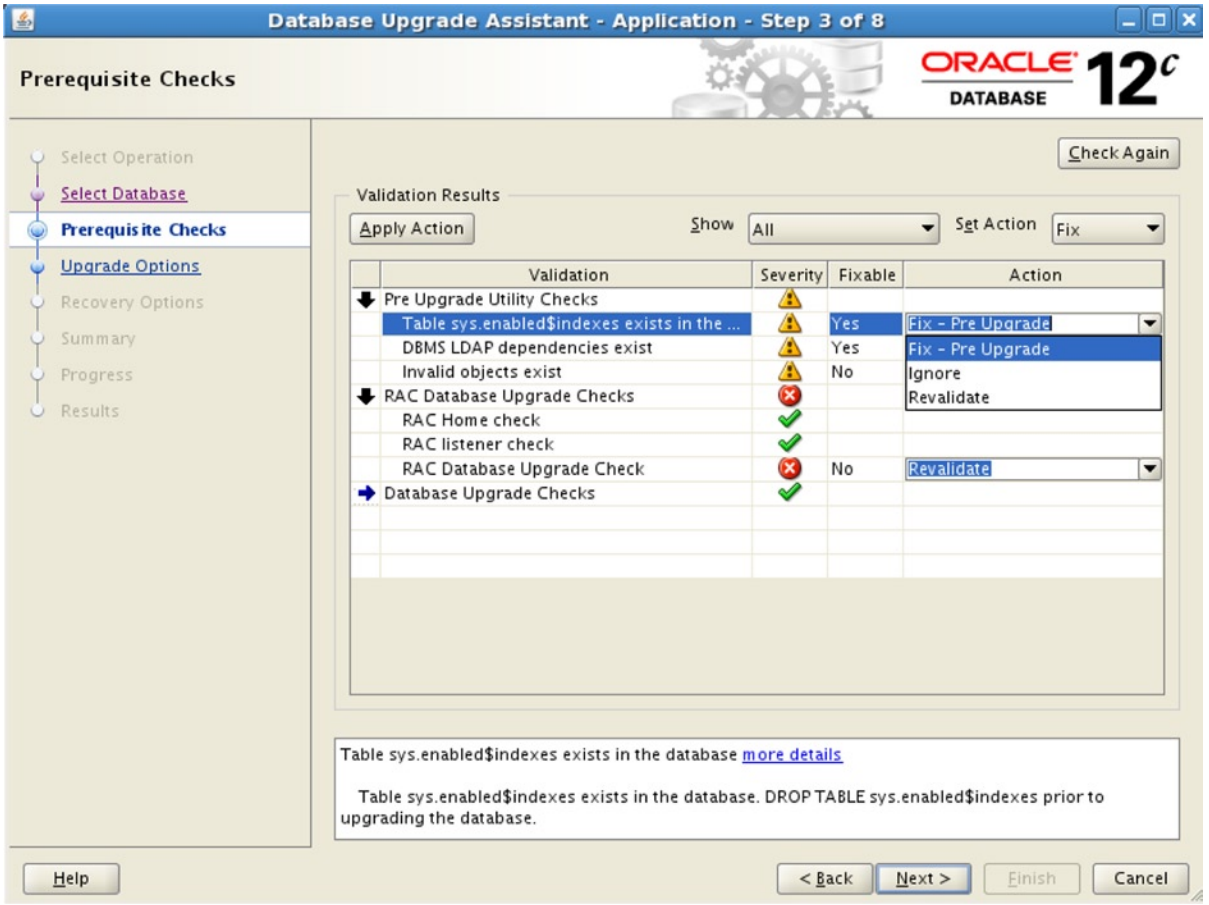


Figure 13-16. *Prerequisite Checks screenshot*

On the `Upgrade Options` screen, set the degree of parallelism to perform the upgrade and recompiling invalid objects in parallel. Optionally, you can also run any custom pre-/post-database upgrade Scripts.

Specify the configuration options for EM Cloud Control on the `Management Options` screen.

The new `Move Database Files` screen allows you to move the datafiles and FRA location between Filesystem, ASM, etc.

Also, the `Network Configuration` screen permits you to migrate an existing listener to the new Oracle release.

Optionally, the `Recovery Options` screen has the flexibility to define a recovery point to recover the database to the same point in case of any failures during the database upgrade process.

Once you input all the required information through the interactive screens, the Database Upgrade Information screen provides all details that you entered for this process.

Click on the `Finish` button to kick-start the upgrade action.

You will see the Upgrade Screen after the database upgrade is complete.

After going through the preceding steps, you will have a successful database upgrade for an Oracle 11gR2 (11.2.0.3) to Oracle 12c Database.

DBUA Advantages

This section will quickly take you through some of the following advantages of DBUA over other database upgrade options:

- When a RAC database is upgraded with DBUA, the tool automatically takes appropriate action, such as enabling, disabling CLUSTER_DATABAE parameter.
- There is the option to disable the archivelog mode during the course of upgrade, to avoid excessive archive log generation, which ultimately expedites the upgrade.
- DBUA makes use of the new tools by default: initiating the catupgrd process in parallel, running the pre-upgrade tool, etc.
- Recommendations in the forms of pre/postupgrade.slq also fix any recommendation failures.
- An option to recompile all objects post–database upgrade and gather pre-/post-database statistics is provided.
- The existing database listener is upgraded to the new Oracle home.
- Most importantly, the tool allows you to define a recovery point to roll back the upgrade in case of failure.
- You will also have an option to migrate the database from one Oracle 12c home to another 12c home.

Database Downgrade

Sometimes after a successful database upgrade, you might have to downgrade the database to its previous release for various factors; hence, you should have a back-out action plan in mind when you do an upgrade procedure. The following is the database downgrade procedure which lets you downgrade the database from 12c to a previous Oracle database release:

Perform the current database full backup.

Disable the database vault, if it exists.

Ensure that the database COMPATIBLE initialization parameter is set to the Database version that supports a direct upgrade (refer to Table 13-2).

Drop the sysman user if OEM is configured. You will have to reconfigure the OEM after you finish the downgrade process.

Disable the CLUSTER_DATABASE initialization parameter, and stop the database as follows:

```
export ORACLE_SID=PRDDB1
SQL> sqlplus / as sysdba
SQL> alter system set cluster_database=false scope=spfile;

srvctl stop database –d PRDDB
```

Start up the database in downgrade mode and then execute the downgrade script, as follows:

```
SQL> STARTUP DOWNGRADE;
SQL> SPOOL /tmp/dbdowngrade.log
SQL> @?/rdbms/admin/catdwgrd.sql
```

The `catdwgrd.sql` script under the Oracle 12c /rdbms/admin home downgrades the 12c Database components to the previous release. If you encounter any ORA issues during the course of downgrade, fix the issue and rerun the downgrade script once again.

After completing the script, shut down the database and exit from the SQL prompt.

Set the previous Oracle version to ORACLE_HOME, PATH OS env variables.

Start up the database in UPGRADE mode to reload the previous version components by running the catreload.sql script from the Oracle 11g /rdbms/admin home. Examples are as follows:

```
export ORACLE_HOME=/u01/app/oracle/product/11.2.0/db_1
sqlplus / as sysdba

SQL> STARTUP UPGRADE
SQL> spool /tmp/dbdowngrade.log
SQL> @?/rdbms/admin/catreload.sql
```

Enable the `CLUSTER_DATABASE` parameter, and shut down and start up the database.

```
SQL> ALTER SYSTEM SET CLUSTER_DATABASE=TRUE SCOPE=SPFILE;
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP
```

Recompile all invalid objects running the `utlrp.sql` script.

```
SQL> @?/rdbms/admin/utlr.sql
```

Downgrade the database version in the OCR using the srvctl downgrade command from the Oracle 12c database home.

```
srvctl downgrade database –d PRDDB –o /u01/app/oracle/product/11.2.0/db_1 –to_version 11.2.0.3.0
```

Summary

This chapter explains the core advantages of upgrading and why an organization needs to upgrade its current (stable) environment with new releases. Before demonstrating a two-node cluster and a RAC database upgrade with a several hands-on examples and slide-decks, all the important prerequisites and precautions that are required for a smooth and successful upgrade were explained. You have also learned a step-by-step procedure for downgrading a successful, failed, or partially upgraded Clusterware and Database to the previous version.

CHAPTER 14

RAC One Node

by Syed Jaffar Hussain

RAC One Node, widely acknowledged as "the always-on single–server database," was first introduced with Oracle 11gR2 (11.2.0.1) Enterprise Edition to render High Availability solutions for single-instance databases. RAC One Node is a single-instance RAC database that runs on a single node in a cluster at any given point. The core objective of RAC One Node features is to minimize the impact of single-instance database availability during planned and unplanned server outages through its exceptional online database migration capabilities. In addition, with the online database conversion capabilities, a RAC One Node database can easily be scaled out to a fully functional traditional RAC database.

The Big Picture

RAC One Node typically provides a traditional cold-failover solution to confront unplanned database server outages by automatically starting up the impacted database, either on the same node or on a preconfigured candidate node in the cluster. RAC One Node provides an option for online database migration between active nodes of a cluster for any prolonged planned database or server maintenance outages. Also, when a node with RAC One Node existence becomes overloaded, the database can be easily moved online to another node in the same cluster. Relocating a RAC One Node database instance from one node to another in a cluster requires no application downtime, as it is performed absolutely online. Moreover, when an application workload ramps up over time and business demands additional database resources to handle the workload efficiently, the RAC One Node database can be easily and quickly upgraded to a multinode, fully functional, and traditional RAC database without any application downtime.

RAC One Node fully supports Oracle Data Guard, and runs on Exadata machines and Oracle virtual machines (VM). With RAC One Node, multiple single-instance databases can be consolidated into a single server with less overhead. In a nutshell, RAC One Node reduces planned & unplanned database outages and significantly improves overall database availability with its unique online migration capabilities.

Note RAC One Node is not supported on third-party clustering solutions.

Some of the key benefits of RAC One Node are as follows:

- Easy and fast online database conversion to a traditional RAC database to meet and satisfy increased workload demands
- Quick online database migration from one node to another node in a cluster during planned software/hardware downtimes and when the node becomes overloaded
- Uninterrupted single-instance service, except during unplanned outage

- Ability to automatically fail over a database during unplanned circumstances with no admin intervention
- Runs over Oracle's Exadata machine
- Provides better server consolidation solutions where many single-instance databases can be consolidated into a single server and manage the CPU resources with the use of instance caging feature
- Certified compatibility with Oracle VM environments
- Automated failover of a database when the database or cluster availability becomes unhealthy
- Less expensive than third-party vendor cold-failover technologies

Oracle RAC One Node comes with an additional cost factor. You will need an additional licensing agreement with Oracle to be able to use RAC One Node in your production environment and also to access the support for the product. The license cost is subject to the number of CPUs on the node where RAC One Node is configured; hence, all the nodes where RAC One Node is configured must be licensed. However, the licensing agreement includes a ten-day-per-year rule allowing the database to be active without additional cost for ten days on a nonlicensed node upon database relocation. For more details about licensing and cost, visit the official Oracle website or contact your local Oracle Support.

Upgrading to 11.2.0.2 or Higher

To upgrade a pre-11.2.0.2 RAC One Node database to 11.2.0.2 or higher, the following steps need to be completed:

1. Upgrade the existing Grid Infrastructure (GI) using the out-of-place upgrade method, if not upgraded already
2. Install 11.2.0.2 or higher RAC or RAC One Node RDBMS binaries
3. Use Database Upgrade Assistant (DBUA) tool to upgrade the existing RAC One Node database

---

■ **Note** Pre-11.2.0.3 DBUA is not aware of and doesn't sense the RAC One Node database. Hence, after upgrading, the database type will be automatically be set to RAC instead of RACOneNode. Therefore, after database upgrade, you need to explicitly convert the database back to RACOneNode using the `srvctl convert` command.

---

4. As a workaround, before upgrading the database, first convert the RAC One Node database to RAC using the `racone2rac.sh` and then use `DBUA` to upgrade the RAC database. After completing the upgrade, convert the database back to RACOneNode using the `srvctl convert` command

Deploying RAC One Node Binaries

Although you can very well make use of the DBCA tool from a typical RAC RDBMS (binaries) home to create a RAC One Node database, optionally, you can also deploy RAC One Node binaries in a separate Oracle Home on the nodes, as per your licensing terms. However, before you install RAC One Node software on any node, ensure the Clusterware software is configured and the cluster stack is up and running on the local node.

The following procedure exhibits a step-by-step procedure of RAC One Node software installation:

- Go to the RDBMS software staging location and initiate the `./runInstaller` from the command prompt.
- Select the `Oracle RAC One Node installation` option on the `Grid Installation Option` and click `Next`, as shown in Figure 14-1.

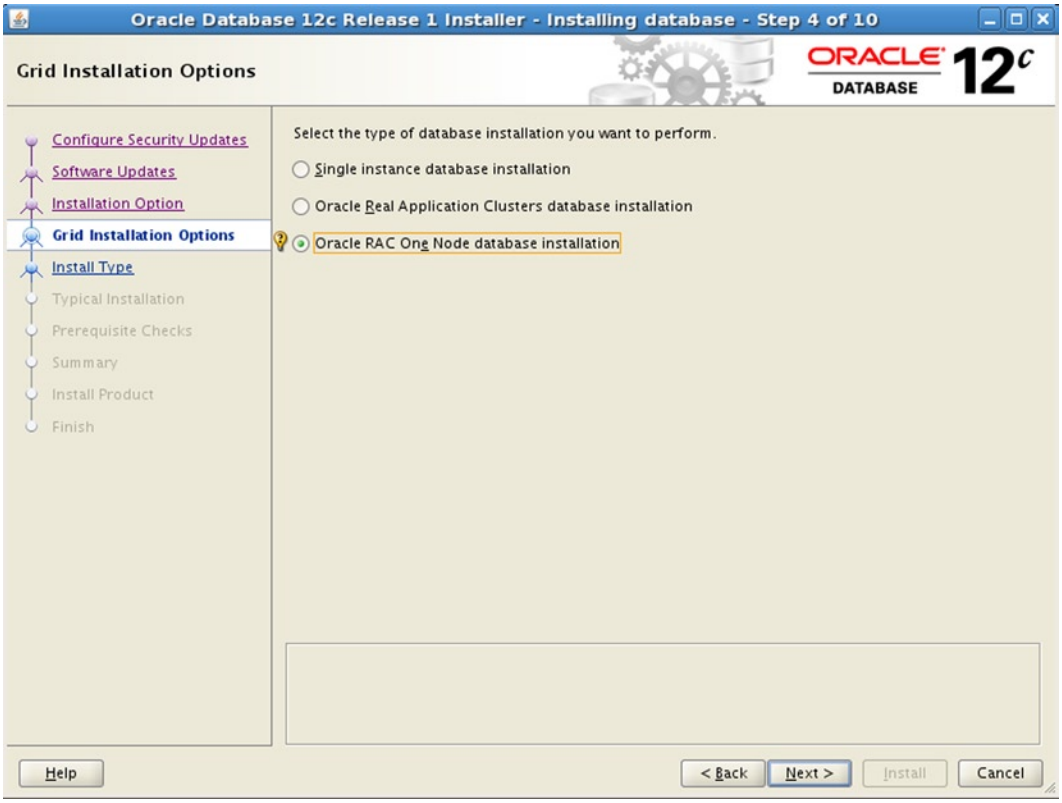


Figure 14-1. *Oracle RAC One Node database installation option*

- On the `Node Selection` screen, select the appropriate node names from the given list where RAC One Node software binaries should be installed and click Next.

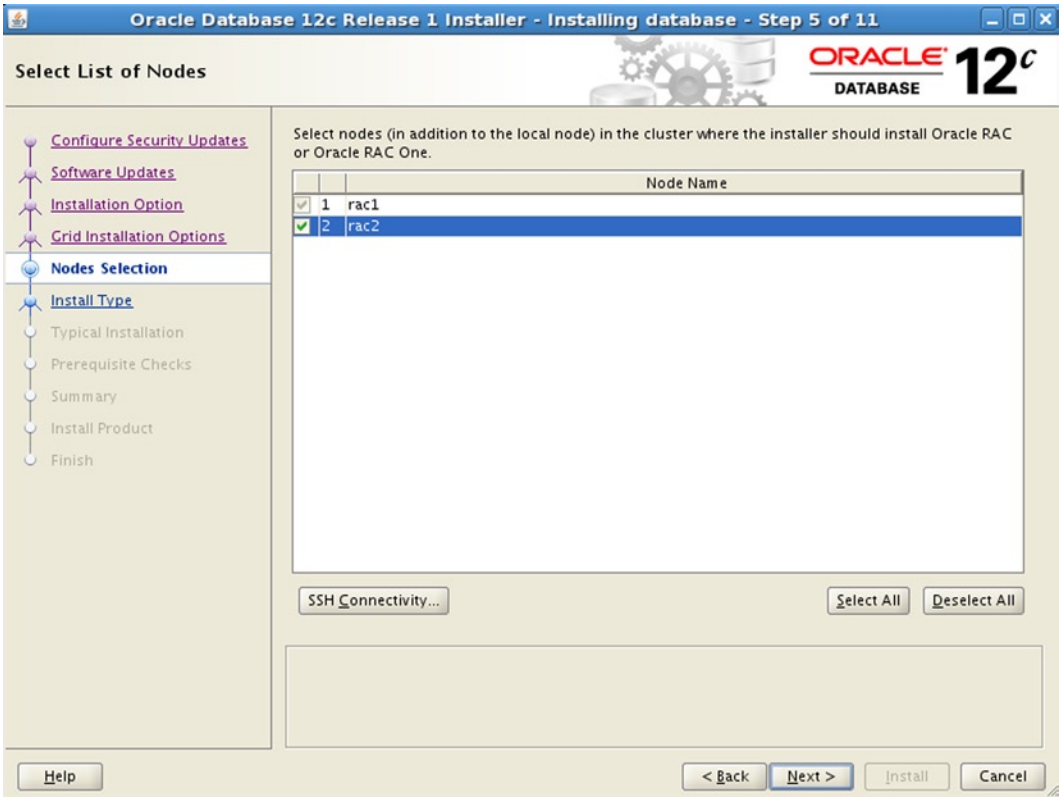


Figure 14-2. *Node selection*

- Input the software location details on the Specify Installation Location screen and click Next (Figure 14-3).

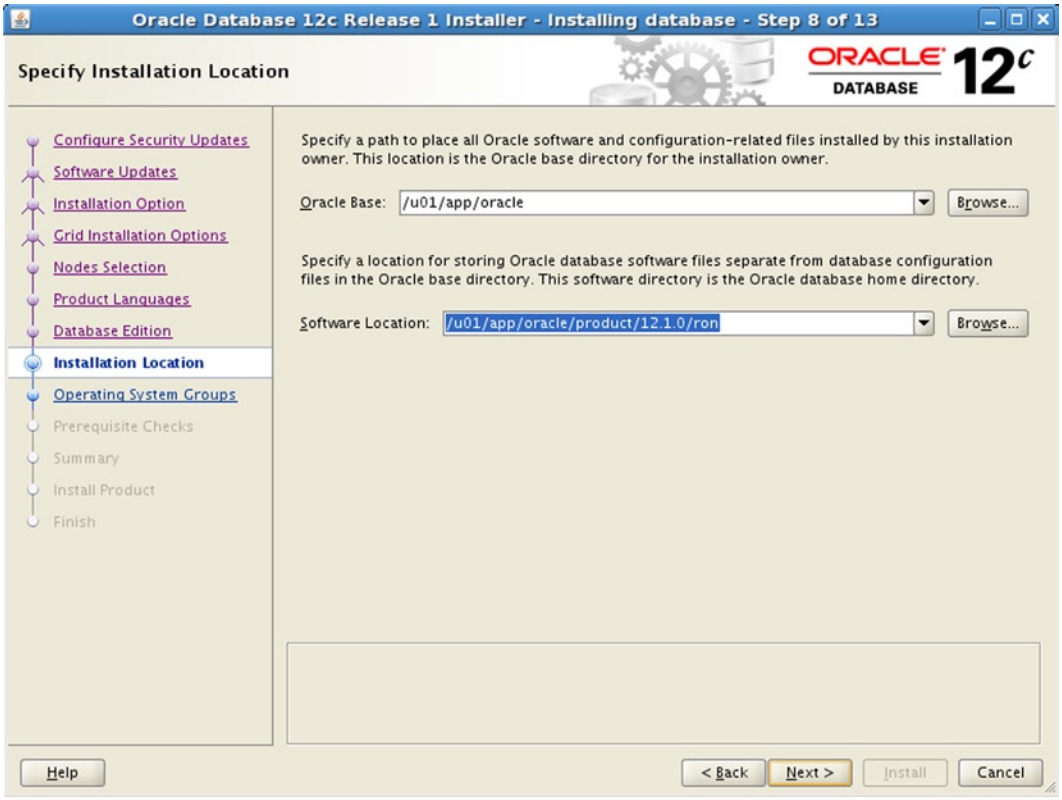


Figure 14-3. *Specify Installation Location screen*

- From here on, just follow the typical installation procedure, running through the rest of the interactive screens, and click Next to move forward.

Deploying a RAC One Node Database

There is no substantial difference in the database creation procedures between RAC and RAC One Node databases. The steps remain the same except for selecting the database type option. Simply follow the same database creation steps that you have already been following over the years.

■ **Note** Prior to 11gR2 Patchset 1 (11.2.0.2), one could create only a single-instance or RAC database with the Database Configuration Assistant (DBCA). Beginning with 11.2.0.2, DBCA is capable of creating a RAC One Node database.

Satisfying Prerequisites

The following are some prerequisites you need to have in place before creating a RAC One Node database:

- GI must be configured, and the cluster stack should be up and running across the nodes
- RAC One Node software or RAC binaries must be installed across nodes

- ASM instance should be up and running
- Required ASM disk groups need to be prepared and mounted among the ASM instances

Initiating DBCA's Creation Process

On the local node command prompt, initiate the DCBA tool from the `$ORACLE_HOME/bin` location, as shown in the following example. Doing so will start the database creation procedure. Our explanation that follows focuses mainly on those aspects of creation unique to creating RAC One Node database using DBCA. Here is the command to invoke DBCA:

```
$ORACLE_HOME/bin/dbca
```

Follow these steps on the `Database Template` screen to create a `RAC One Node database`:

- Select the `Oracle RAC One Node database` option from the `Database Type` drop-down list.
- Choose the `Admin-Managed database` type from the `RAC Configuration Type` drop-down list.
- Under the `Select Template` section, select the appropriate database template option, as shown in Figure 14-4.

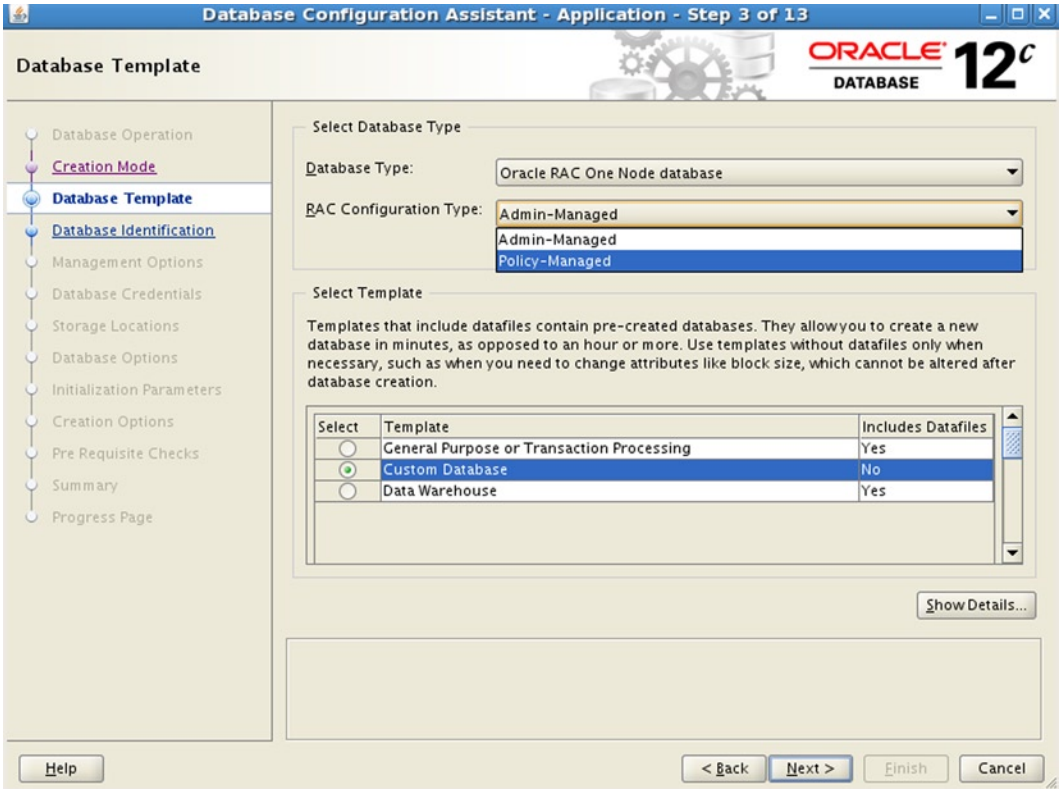


Figure 14-4. *The DBCA Database Template screen*

- Click Next to proceed.

On the Database Identification screen (Figure 14-5), do the following:

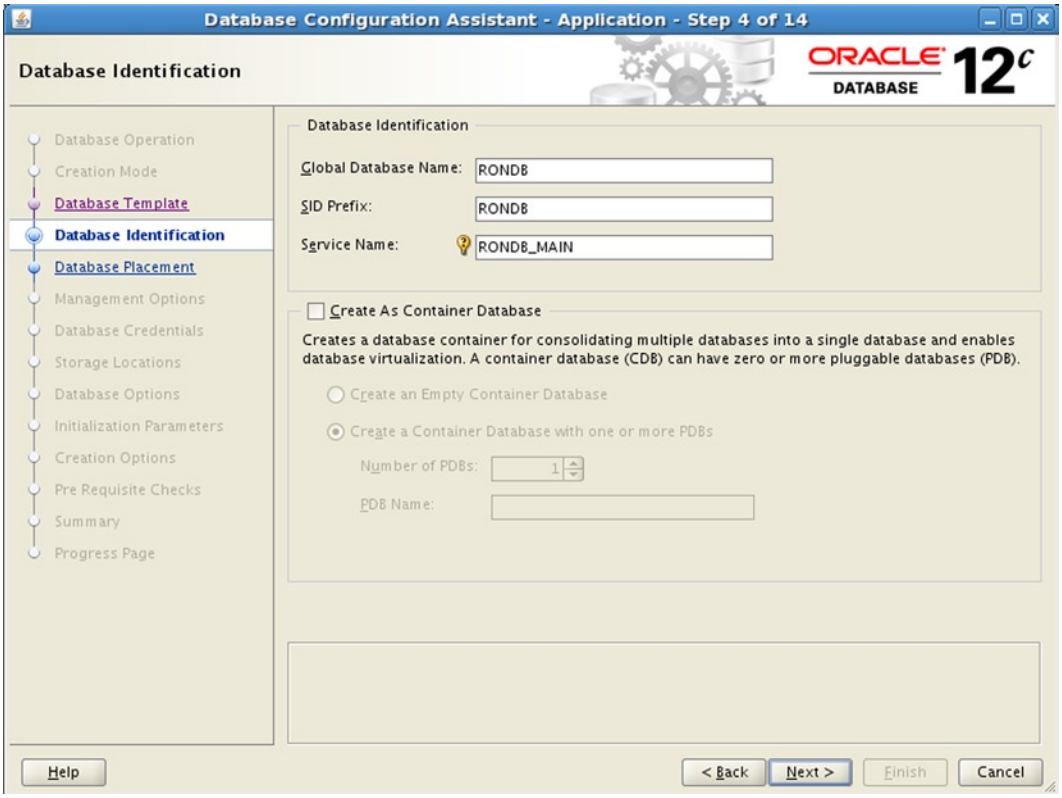


Figure 14-5. *Database Identification screen*

Input `Global Database name`, `SID prefix`, and a `service name` which will be later used by the application to connect to the database. You need to avoid making the service name the same as the database name, as you will get an error if you do so.

Click `Next` to continue to the next steps, which are illustrated in Figure 14-6.

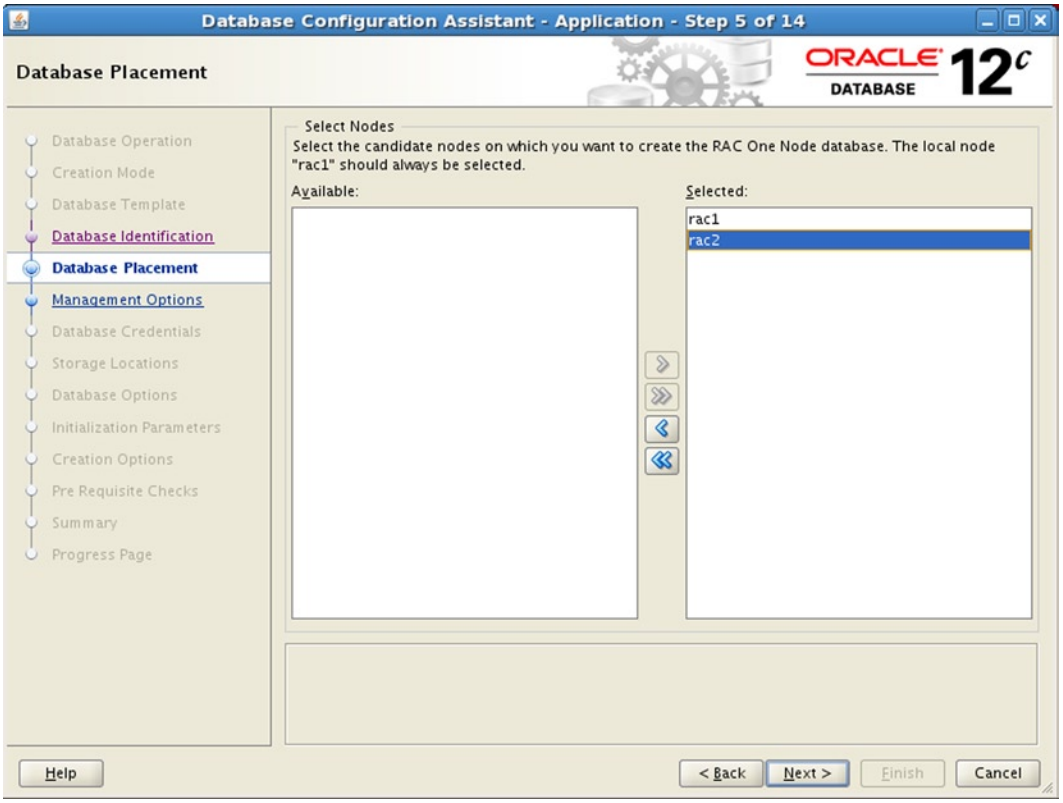


Figure 14-6. *Database placement and node selection options*

Select required nodes from the available nodes list under the `Select Nodes` section. If you leave selection to the local node, you will get a warning message that you have selected only one node. However, when multiple nodes are selected, the database will be initially created on the local node and the list of other nodes will be kept as preferred available nodes for online migration purposes.

Click `Next` to continue further. From here on, follow the typical database creation procedure steps that you have been using it over the years to create a new database.

Parameters Specific to RAC One Node

You might be wondering whether RAC One Node database–specific parameters exist. As of now, there are no parameters specific to RAC One Node database; however, the following parameters on RAC One Node reflect a RAC database type:

```
cluster_database = TRUE
 instance_type = RDBMS
 instance_number = <instance_number>
```

Managing RAC One Node Database

This section focuses on RAC One Node database administration concepts. You will learn how to extract RAC One Node database configuration details and identify the predefined candidate server list for instance failover or online migration. You'll also learn to distinguish the type of database and relocate the database online.

Verifying Configuration Details

Verify the RAC One Node database configuration details after creating the database. The following example extracts useful information about the RAC One Database configuration:

```
$ srvctl config database -d RONDB

          Database unique name: RONDB
      Database name: RONDB
      Oracle home: /u01/app/oracle/product/12.1.0/ron
      Oracle user: oracle
      Spfile: +DG_RONDB/RONDB/spfileRONDB.ora
                  Password file: +DG_RONDB/rondb/orapwrondb
      Domain:
      Start options: open
      Stop options: immediate
      Database role: PRIMARY
      Management policy: AUTOMATIC
      Server pools: RONDB
      Database instances:
      Disk Groups: DG_RONDB
      Mount point paths:
      Services: RONDB_MAIN
      Type: RACOneNode
      Online relocation timeout: 30
      Instance name prefix: RONDB
      Candidate servers: rac1,rac2
      Database is administrator managed
```

Let's have a closer look at some of the key configuration elements in the preceding listing.

Type**:** This is one of the vital parameters that require your attention. The parameter helps to identify the database type: RAC or RAC One Node.

`Online relocation timeout`**:** Displays the default timeout duration (in minutes) to complete any ongoing transactions of a session on the instance before it gets terminated on the local node as part of an online database movement.

- Typically, when online database relocation is initiated, the instance will wait for the specified amount of time to allow active transactions to complete before it gets terminated. If a session is unable to complete all its active transactions within the allotted time frame, the transaction will then be canceled and the session will be terminated subsequently. The `-w` parameter provides an option to specify a user-preferred timeout duration of up to 720 minutes.

`Candidate servers`: Lists the names of all candidate nodes that are eligible for instance failover operations. In the event of instance failure, it is important to know that Oracle will select a server randomly, in no particular order, to fail over the instance. Hence, it is strongly recommended to use SCAN to connect to the database to avoid any connection failures on instance migration or failover over different nodes.

Verifying the Online Relocation Status

Execute the following command to verify the database's online relocation status:

```
# srvctl status database -d RONDB
Instance RONDB_1 is running on node rac1
      Online relocation: INACTIVE
```

This output illustrates that the RONDB_1 instance is currently active on node rac1. More importantly, no online database relocation is taking place at this moment.

Stop and Start the Database

Like that of other typical RAC and non-RAC databases, a RAC One Node database's shutdown/startup can be performed either from the SQL*Plus prompt or using the srvctl utility. Though the srvctl usage is highly recommended for such database management operations, you can also use the SQL*Plus utility to start/stop the database if srvctl is unable to complete the job.

The following cluster command stops the database:

```
# srvctl stop database -d RONDB
```

In contrast to a RAC and Non-RAC database, starting the RAC One Node database has something new to offer. On instance failure or stop, Oracle will randomly select an eligible candidate server from the predefined candidate server list to bring up the instance automatically. Presuming that Node A and Node B are listed in the candidate server, and the database is running on Node A prior to instance failure or startup, the instance will pick either a different server or the same server from the list to start a replacement instance. Therefore, under some circumstances, there will is no guarantee in the context that the new instance will restart on the same node.

When you want to start the database on a particular node using the srvctl utility, you must use the `–n` argument to specify the name of the node on which the database should start. For automatic instance failover, Oracle might select the candidate server in the order displayed in the config output.

Use the following example to start up the database:

```
$srvctl start database –d RONDB -n rac1
```

Performing Online Database Relocation

Just for a moment, imagine yourself in any of the following circumstances surrounding common maintenance activity:

- You are planning a prolonged maintenance activity on the database server
- The node on which your RAC One Node database is running becomes overloaded, running out of resources and capacity
- You are applying Oracle patches on the node

- You need to move the database over to another node in the cluster to minimize the downtime during a planned outage, perhaps as a prelude to decommissioning an older node
- You are performing a server upgrade or patching

In these contexts, database availability is critically important. The uniquely powerful online database relocation feature supersedes other traditional third-party failover technologies. An online database movement from one node to another in a cluster is one of the key features of RAC One Node. The ability to perform database relocations without disrupting applications or end-users can't be overemphasized when discussing RAC One Node.

The following example kicks off an online database migration procedure with a 5-minute timeout to complete any ongoing transactions in the current instance. The `-w` option specifies the preferred timeout in minutes. The `–v` option enables verbose output.

```
$srvctl relocate database -d RONDB -n rac2 -w 5 –v
```

The following will be displayed if the target node is not part of the predefined server availability list:

```
Added target node rac2
Configuration updated to two instances

Instance RONDB_2 started
Services relocated
Waiting for 5 minutes for instance RONDB_1 to stop.....
Instance RONDB_1 stopped
Configuration updated to one instance
```

If you are migrating the database over a new node on which instance was not configured previously, RAC One Node automatically prepares the prerequisites for the second instance: it creates UNDO tablespace, adds new redo logs, and so forth.

1. RAC One Node successfully configures instance 2 and starts up the instance on the target host.
2. Database services are automatically relocated to the target host.
3. Transactional shutdown is issued on the first instance to shut down the database in the context.
4. RAC One Node will wait for 5 minutes for any active sessions to complete their ongoing transactions on the local instance. During this short period of time, the database will be an active/active cluster having two instances up and running. However, only instance 2 will receive new connections.
5. If a session couldn't complete its current transaction within the specified timeout, the transaction will be canceled and the session will be terminated.
6. However, if transactions are completed earlier than the given timeout, the instance will be stopped earlier.
7. All existing connections are gracefully migrated to instance 2.
8. If any active connections remain on instance 1, that instance will be stopped by a `shutdown abort` command.

In the alert.log of the first instance, you will notice the transactional shutdown, as mentioned in the step 5. In addition, you will also notice that Oracle updated the timeout settings for active session to complete the transactions. The command to look for in the log is the following:

```
ALTER SYSTEM SET shutdown_completion_timeout_mins=30 SCOPE=MEMORY;
```

When an active session can't complete the ongoing transaction within the defined timeout slot, the instance shutdown transaction will be replaced by a shutdown abort, which will kill the still-active transactions without waiting to complete.

While online database relocation is taking place, from the other window on the local node you can verify the relocation status by executing a `srvctl status` command, as shown in the following example:

```
$ srvctl status database -d RONDB

 Instance RONDB_1 is running on node rac1
 Online relocation: ACTIVE
 Source instance: RONDB_1 on rac1
 Destination instance: RONDB_2 on rac2
```

The online database relocation status shows an ACTIVE status during relocation. Furthermore, the output gives a clear idea about where the source and target instances are located and whether they are up and configured.

Now, have a look at the post–database online relocation verification by issuing the `srvctl` command in this next example:

```
$srvctl config database -d RONDB

 Database unique name: RONDB
 Database name: RONDB
 ...
 Services: RONDB_MAIN
 Type: RACOneNode
 Online relocation timeout: 30
 Instance name prefix: RONDB
 Candidate servers: rac2,rac1
 Database is administrator managed
```

THE OMOTION UTILITY

In this section, I will briefly discuss the utility that used to perform the online database. The Omotion utility was initially shipped with Oracle 11gR2 (11.2.0.1) on the Linux platform to support Oracle RAC One Node online database migration between nodes in the same cluster. The utility indeed plays a significant role in online instance movement from one node to another node with no downtime or disruption to the service availability.

In pre-11.2.0.2, the Omotion utility was explicitly used to perform an online instance movement between nodes in the same cluster. Starting in 11.2.0.2, Omotion functionality is now part of the `srvctl relocate` command to perform the instance relocation procedure.

Handling Unplanned Node and Cluster Reboots

Here is a delicate question for your consideration: how do you deal with unplanned server outages? Perhaps you are using a cold-failover technology from a third-party vendor, a technology such as IBM's HACMP, HP's Serviceguard, or Veritas' Cluster Server. Although this software provides a solution to handle unplanned outages, you are still required to have an administration team—composed of either DBAs or operating system administrators—to intervene to manage the situation. This could mean that it takes a while to complete the failover. Even if you can automate the process to handle unplanned node outages, how do you deal with the problem of hung and unresponsive databases?

A RAC One Node database requires no DBA or operating system administrator intervention to complete the activities for recovering from unplanned server outages. RAC One Node automatically performs cold failover without administrator intervention. If you suspect that a failover has occurred, you can execute the previously mentioned `srvctl status` command to verify the node on which the database is currently active.

Unlike the third-party cold-failover technologies, Oracle RAC One Node is tightly integrated with Oracle Clusterware and is closely monitored by the cluster. When a node suffers an unplanned outage, Oracle Clusterware will detect the failure and try to bring the database up on the same server in the first attempt. If the server is inaccessible for any reason, Oracle Clusterware will automatically relocate the impacted database onto a predetermined alternate server within the cluster. This is all managed automatically and avoids any DBA intervention.

Figure 14-7 depicts the cluster node reboot scenario in a two-node cluster environment where the RONDB database automatically fails over from Node A to Node B. As you can see in the following picture, once Node A became unavailable for some reason, the RONDB_1 instance switched automatically to Node B as RONDB_2, and at this point in time, Node B had two instances running, namely, RONDB_2 and the other database instance.

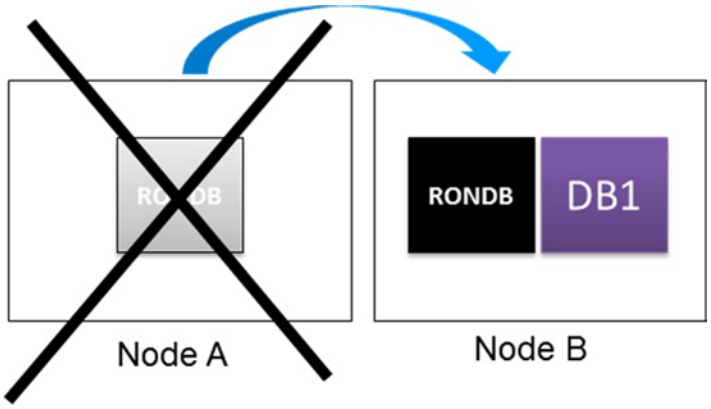


Figure 14-7. *The cluster node reboot scenario*

Although there is very little application downtime involved, Oracle RAC One Node's ability to manage automatic restart and relocate the impacted database services onto a survival node puts Oracle RAC One Node technology far ahead of traditional third-party cold-failover solutions and virtualization solutions.

Converting Between RAC One Node and Standard RAC

With Oracle RAC One Node, you can easily scale out the database to a fully functional, traditional RAC database, subject to the licensing terms, to meet and serve increased business workload demands. With RAC One Node, these scaling changes can be made online. No application downtime is needed.

Scaling Up to Standard RAC

The following command initiates the online upgrade of a RAC One Node database to a fully functional RAC database:

```
srvctl convert database -d RONDB -c RAC
```

The following slightly modified syntax is for use on a 12c database for database conversion:

```
$srvctl convert database -db db_unique_name -dbtype RAC [-node node_name]
```

In this command, the parameter `-c` (replaced as `–dbtype` in 12c) has two options: `RAC` and `RACONENODE.` Specify RAC to scale upwards, taking a RAC One Node database to a standard RAC database.

Execute the `srvctl convert database` command on the node on which RAC One Node is running. Otherwise, you will get the following error message:

```
PRKO-2159 : Option '-i' should be specified to convert an administrator-managed RAC database to its
equivalent RAC One Node database configuration
```

Specify the `–n` option when the RAC One Node database is not running on the local node. The following is the syntax to use:

```
$ rvctl convert database -d <dbname> -c RAC [-n <node_name>]
```

In this syntax, the parameter `-n` specifies the node name on which the RAC One Node database is running.

When the conversion procedure completes successfully, you can verify the database configuration to ensure that the database has successfully converted to RAC. Do so by issuing the `srvctl config` command as in the following example:

```
$ srvctl config database -d RONDB

Database unique name: RONDB
Database name: RONDB
..........
Database instances: RONDB_2
Disk Groups: DG_RONDB
Mount point paths:
Services: RONDB_MAIN
Type: RAC
Database is administrator managed
```

At this point, you now have a full-fledged RAC database with a single instance. You will have to add additional instances according to your business needs. You can do so using the following syntax:

```
$ srvctl add instance -d RONDB -i RONDB_1 –n rac2
```

After adding an instance, bring up the new instance using the following srvctl command:

```
$ srvctl start instance -d RONDB -i RONDB_1
```

The RONDB database now has two instances. Verify using the following command:

```
$ srvctl status database -d RONDB
```

```
Instance RONDB_2 is running on node rac2
Instance RONDB_1 is running on node rac1
```

Scaling Down to RAC One Node

The process to fall back or scale down a RAC database to RAC One Node is pretty simple and straightforward. However, the following prerequisites must be fulfilled:

- The RAC database must not contain more than one instance. If so, all instances except one must be stopped and removed from the RAC database to avoid `PRCD-1214 : Administrator-managed RAC database RONDB has more than one instance` error. Use the following commands to stop and remove instance 2:
  - Srvctl stop instance –d RONDB –i RONDB_2
  - Srvctl remove instance –d RONDB –i RONDB_2
- At the same time, ensure that the instances that are going to be removed are not in the preferred instances list of any database services. If they are, modify the existing database services accordingly.

When you've met all the prerequisites, issue the following command to convert a RAC database to a RAC One Node database:

```
$ srvctl convert database -d RONDB -c RACONENODE -i RONDB_2 –w 5
```

When the command completes, your standard RAC database will now be a RAC One Node database. To verify the database type, use the `srvctl config` command.

Managing RAC One Node with Cloud Control 12c

As stated earlier, an Oracle RAC One Node database can be thoroughly monitored and managed with Cloud Control 12c. This section is going to demonstrate some of the actions using sequences of screenshots. However, we won't describe how to install Oracle Enterprise Manager (OEM) agents and discover the targets, as there won't be any difference with regard to the process as it's usually done.

■ **Note** Management with OEM 11gR2 is possible, although making use of Cloud Control 12c is the preferred method. For this reason, we will limit our discussion to the use of Cloud Control 12c. If you currently have OEM 11gR2, then visit Oracle Support's website to obtain required patches to make RAC One Node work with OEM 11gR2.

Database Relocation with Cloud Control 12c

To perform relocation, go to the availability pulldown menu. Select *Cluster Database Operations* ➤ *Online Database Relocation*. The screen is shown in Figure 14-8.

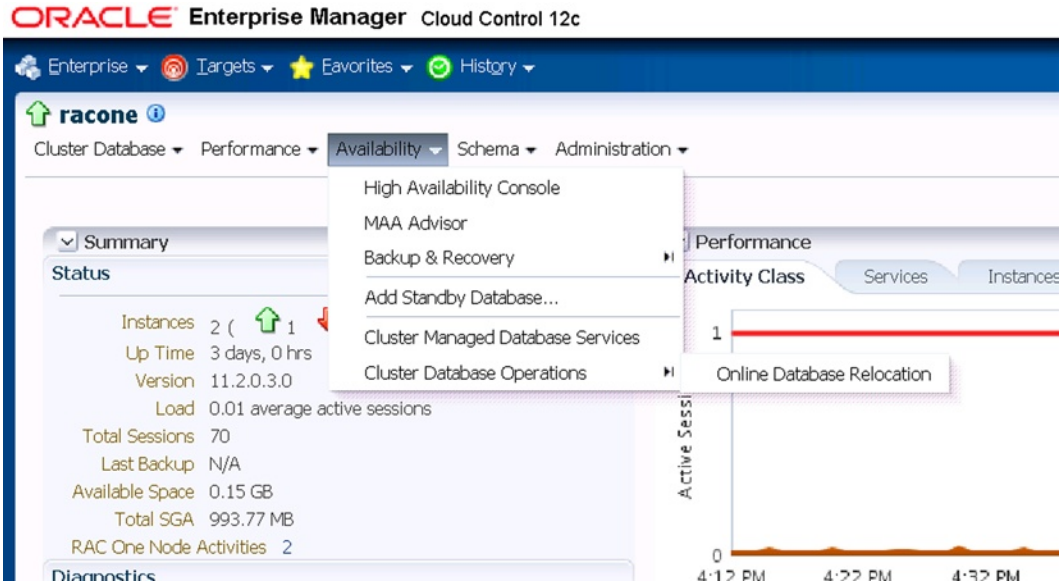


Figure 14-8. *Database Availability screenshot*

You then need to input the database user connect credentials to connect to the database. After you've input the host credentials, the relocation screen shown in Figure 14-9 will be displayed.

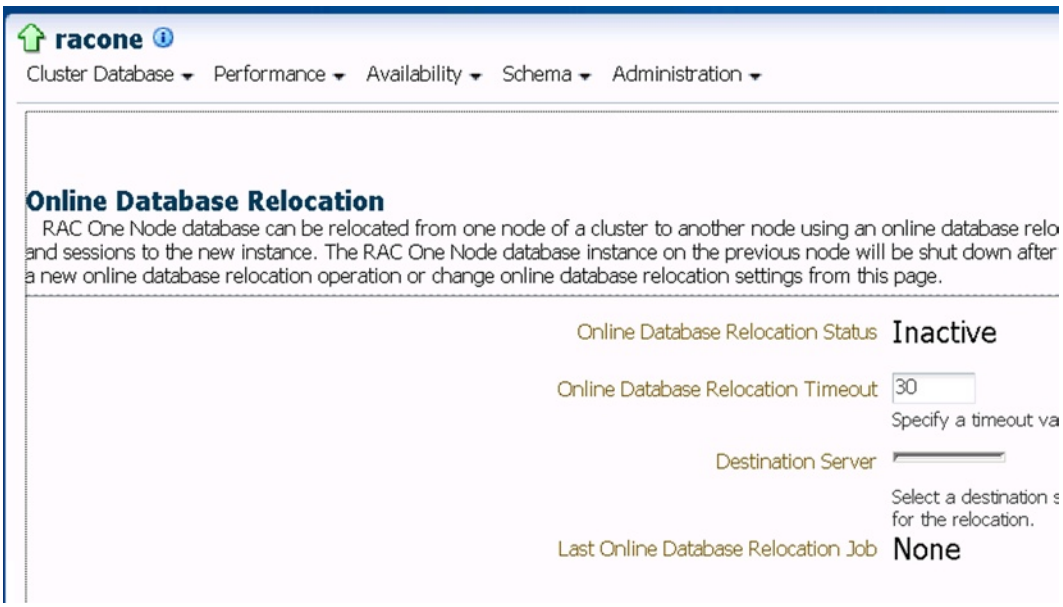


Figure 14-9. *The RAC One Node Online Database Relocation screen*

Verify the current online relocation status, which in this case is "Inactive." The online relocation timeout can be changed from the current (default) value. Whatever value you define will be passed to the `srvrctl` utility's `relocate` statement in the `-w` parameter.

The destination server is empty if only one server is available to relocate to; if your cluster contains more servers, then the destination server will contain a list with available servers. If a previous relocation took place, the name of the relocation job would be listed as well.

Provide the required relocation details and click the Start Online Database Relocation button. This button is shown in Figure 14-10. A job will be created to execute the relocation.

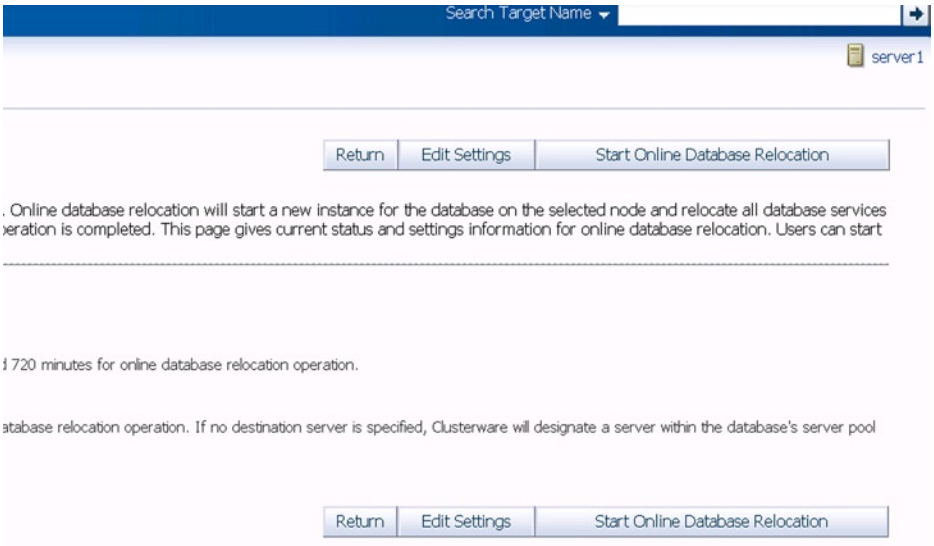


Figure 14-10. *The buttons at the bottom of the Online Database Relocation screen*

When the job is submitted, a message will be displayed. Then, in the background, the relocation takes place. You see such a submission in Figure 14-11.

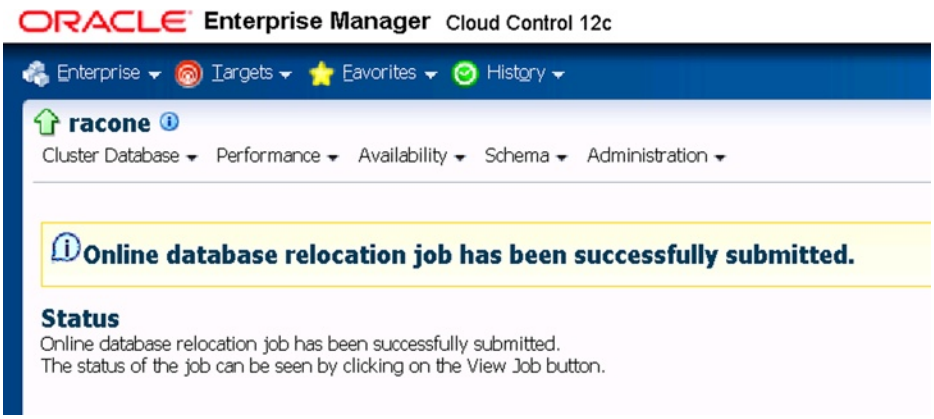


Figure 14-11. *Startup of an online database relocation job*

Figure 14-12 shows that the relocation is complete and that the job was successful. Figure 14-13 shows the resulting status. You can see that one instance is up, and the other is down. The database has been successfully relocated to the instance that shows as up.

Figure 14-12. *Relocation job status*

Figure 14-13. *EM12c reporting one instance as up and the other as down*

Third-Party Cold Failover vs. RAC One Node

A third-party technology such as HP's Serviceguard, IBM's HACMP, or Veritas' Cluster Server can be used to address unplanned server outages. These technologies manage a database in active/passive configuration to offer failover solutions to a single-instance database. In the event of a server failure, the passive instance on the other node will be started manually or with automated scripts. However, application connections need be redirected to the active instance.

With RAC One Node, Oracle offers a similar solution to deal automatically with long unplanned as well as planned downtimes. However, Oracle RAC One Node has the upper hand over any traditional third-party database failover solution. The following are some things that Oracle RAC One Node provides for that the competition does not:

- Database and OS patching while the database continues to be available
- Online storage migration and consolidation when ASM is used
- Easy upgrade to a full RAC database, with no downtime
- Monitoring of database and Clusterware health status
- Faster database failure notification to clients, leading to quicker reconnections
- Automatic restart of an instance in the event of problems, either on the same node in the cluster or on a different node, whichever is needed

Most importantly, RAC One Node provides a single-vendor solution. You can easily implement it without having to tie together multiple products. Support comes from a single source.

Summary

In a nutshell, this chapter explains the concepts and overall benefits of Oracle's RAC One Node feature. We've also covered deployment of RAC One Database binaries and how to create a RAC One Node database using the DBCA tool. Also, we demonstrated how to convert a database online and showed how to scale out to a full RAC database to achieve load balancing solutions. When you need High Availability solutions for your single-instance database to confront planned and unplanned outages, RAC One Node is the perfect choice.

Index

A

B

C

D

E

F

G

H

I, J, K

L

M

N

O

P, Q

R

S

T

U, V

W, X, Y, Z

Expert Oracle RAC 12c

Syed Jaffar Hussain
Tariq Farooq
Riyaj Shamsudeen
Kai Yu

Apress®

Expert Oracle RAC 12c

Copyright © 2013 by Syed Jaffar Hussain, Tariq Farooq, Riyaj Shamsudeen, and Kai Yu

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

ISBN-13 (pbk): 978-1-4302-5044-9

ISBN-13 (electronic): 978-1-4302-5045-6

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

President and Publisher: Paul Manning
Acquisitions Editor: Jonathan Gennick
Developmental Editor: Tom Welsh
Technical Reviewers: Arup Nanda and Sandesh Rao
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Steve Weiss, Tom Welsh
Coordinating Editor: Katie Sullivan
Copy Editor: Brendan Frost
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com/9781430250449. For detailed information about how to locate your book's source code, go to www.apress.com/source-code.

Contents

About the Authors

Syed Jaffar Hussain has over 20 years of I.T. experience that includes more than 14 years of production Oracle database administration. Oracle has honored him with the prestigious Oracle ACE Director role and named him DBA of the Year for 2011, both for his excellent knowledge and for contributions to the Oracle community. He is an Oracle Certified Master (OCM) for Oracle Database 10g—a status granted only after passing extensive challenges in a hands-on environment. He is also an Oracle Database 10g RAC Certified Expert. Syed Jaffar is coauthor of *Oracle 11g R1/R2 Real Application Clusters Essentials* (Packt Publishing, 2011), and he blogs regularly at `http://jaffardba.blogspot.com`. He is a frequent speaker, has presented at various Oracle conferences, including Oracle OpenWorld, and has delivered many technical Oracle-oriented webinars. Also, he actively participates in most of the social networking sites: you can follow him on `twitter/facebook/linkedin` on sjaffarhussain id, and he can be reached at `sjaffarhussain@gmail.com`.

Tariq Farooq is an Oracle technologist/architect/problem-solver and has been working with various Oracle technologies for 20+ years in very complex environments at some of the world's largest organizations. He is the founding President of the IOUG Virtualization & Cloud Computing Special Interest Group. He is an active community leader/organizer, speaker, author, forum contributor, and blogger. He is the founder of `www.BrainSurface.com`, a social networking & IT collaboration site with thousands of signed-up users from the various Oracle communities. Tariq founded, organized, and chaired various conferences, including VirtaThon, the largest online-only conference for the various Oracle domains; the CloudaThon & RACaThon series of conferences; and the first-ever Oracle-centric conference at the Massachusetts Institute of Technology (MIT) in 2011. He was the founder and anchor/show host of the VirtaThon Internet Radio series program. Tariq is an Oracle RAC Certified Expert, holds a total of 14 professional Oracle certifications, and is the author of 100+ articles, whitepapers, and other publications. Tariq has presented and spoken at almost every major Oracle conference/event all over the world, including Oracle OpenWorld, COLLABORATE, BrainSurface, VirtaThon, IOUG, OOUG, OUGLC, TCOUG, UKOUG, The OTNExpert+ Conference, and others. Tariq has been awarded the Oracle ACE and ACE Director awards from 2010 to 2013.

Riyaj Shamsudeen is an industry-recognized RAC expert, database internals specialist, and performance tuning specialist with 20 years of experience in implementing, using, and tuning RAC and Oracle products. He is an Oracle ACE Director and proud member of the OakTable network. Riyaj has coauthored four books about Oracle Database and performance. He is an active blogger (at http://orainternals.wordpress.com), President of OraInternals, and frequent international speaker in major conferences such as UKOUG, HOTSOS, OpenWorld, and RMOUG. Social: www.linkedin.com/in/riyajshamsudeen; @riyajshamsudeen.

Kai Yu is a Senior Principal Engineer and technologist in Dell's Oracle Solutions Engineering Lab specializing in Oracle RAC, Oracle Virtualization, and Cloud. With 18+ years of experience working on Oracle technology, he has implemented and managed many large, mission-critical Oracle RAC databases, including those in his tenure with an IT organization having more than 2,000 RAC databases. Kai is a well-known author of technical articles and a frequent international speaker at Oracle conferences in 16 countries such as OpenWorld, Collaborate, UKOUG, EMEA OUG Harmony, Norway OUG, OTN Asia/Pacific tour, and Latin America tours. He has also keynoted at Oracle Architect Day.

Kai has served IOUG's Oracle RAC SIG as president, and in two chair positions. He has served the IOUG collaborate conference committee, as well as managing IOUG's RAC boot camp, HA boot camp, and Enterprise Manager 12c deep dive. He was awarded the Oracle ACE Director title in 2010, given the Oracle ACE Spotlight in 2011 by Oracle Technology Network (OTN), and named the 2011 OAUG Innovator of Year Award by the Oracle Applications User Group (OAUG). In 2012, *Oracle Magazine* awarded him the Oracle Excellence Award: Technologist of the Year: Cloud Architect. Kai has been active in sharing his Oracle knowledge on his Oracle blog at http://kyuoracleblog.wordpress.com/.

About the Technical Reviewers

Arup Nanda (Email: arup@proligence.com; Twitter: @arupnanda; LinkedIn: linkedin.com/in/arupnanda/) has been an Oracle DBA for the last 18 years, spanning all aspects of the job from modeling to performance tuning. He specializes in RAC, Exadata, and High Availability solutions. He has authored four books on Oracle technology, written 500+ published articles, and presented almost 300 training sessions in 22 countries. He is an Oracle Certified Professional, Oracle ACE Director, member of the OakTable Network, member of the Board of Directors of Exadata SIG, and member of the Editorial Board for *SELECT Journal* (the IOUG publication). Acknowledging his accomplishments and community involvement, Oracle awarded him DBA of the Year in 2003 and Enterprise Architect of the Year in 2012. He blogs at http://arup.blogspot.com.

Sandesh Rao is a Senior Director running the RAC Assurance Team within RAC Development at Oracle Corporation specializing in performance tuning, high availability, disaster recovery, and architecting cloud-based solutions using the Oracle Stack. With more than 14 years of experience working in the HA space and having worked on several versions of Oracle with different application stacks, he is a recognized expert in RAC, database internals, and solving big data–related problems. Most of his work involves working with customers in the implementation of projects in financial, retailing, scientific, insurance, biotech, and the tech space. His current position involves running a team that develops best practices for the Oracle Grid Infrastructure 12c including products like RAC (Real Application Clusters) and Storage (ASM, ACFS). You can find more details about him and his work at www.linkedin.com/pub/sandesh-rao/2/956/1b7.

Acknowledgments

I would like to dedicate this book to my parents (Mrs. and Mr. Saifulla), my wife Ayesha, my three little champs (Ashfaq, Arfan, and Aahil), and the entire Oracle community.

First and foremost, I would like to thank the Almighty for everything and giving me such a wonderful life. Also, my parents for teaching me all the good things in my life and making me what I am today. I owe a very big thank-you to my wife and kids for sacrificing their family time and letting me concentrate on this assignment, and encouraging me all through this phase. Beyond a doubt, this project wouldn't have been possible without the tremendous moral support and encouragement from my wife, friends, and colleagues. Thank you everyone once again.

I would like to take this opportunity to thank my management (Khalid Al-Qathany, Hussain Mobarak AlKalifah, and Majed Saleh AlShuaibi), my dearest friends (Khusro Khan, Mohammed Farooqui, Naresh Kumar, Shaukat Ali, Ahmed, Mohammed Abdul Kaleem, Mohammed Arifuddin Ghori, Haris Afsar Ahmed, Sachin, Ibrahim Ali, Chand Basha, Mohammed Anees, Asgar Ali, Gaffar Baig, Hakeem, Mohsin), my fellow colleagues, well-wishers, supporters, nears and dears for their immense support and constant encouragement.

I am also very thankful from the bottom of my heart to the official technical reviewers (Sandesh Rao and Arup Nanda) for taking some time out from their busy schedule to review our book and for providing their valuable inputs. I can't conclude the list without mentioning the efforts by the official members of Apress, who put everything into this project. This is for every individual who was involved on this project on behalf of Apress: "You people just rock and have been of great help to us during the book-writing journey; looking forward to working with the team in future; thank you everyone." A very special thanks to Jonathan Gennick for having confidence in us and giving us the opportunity to write for Apress Publications.

I would like to extend my sincere thanks to my unofficial technical reviewers, in no particular order, Jeff Smith, Bernhard de Cock Buning, Asad Khan, Yury Veli Kanov, Khusro Khan, Shautkat, Aman Sharma, and Inam Ullah Bukhary, who helped me a lot, in terms of reviewing my part of chapters, providing their inputs, plus correcting my grammar.

I would like to conclude this by thanking my fellow coauthors, Riyaj Shamsudeen, Kai Yu, and Tariq Farooq, for the immense contribution, dedication, and hard work that they have displayed over the past year on this book. The book wouldn't have been feasible without your help/support/assistance/contribution. I really enjoyed working with you guys and am looking forward to other opportunities to work with you again in the coming times.

—Syed Jaffar Hussian

I would like to take this opportunity to express my endless gratitude and thanks to the Almighty, my parents (Mr. and Mrs. Abdullah Farooq), my wife (Ambreen), especially my kids (Sumaiya, Hafsa, Fatima, and Muhammad-Talha), and everyone else around me, both at home and at work, for continuing to stand behind me while I spent the better part of a year working on this project.

Authoring a book is a painstakingly lengthy and laborious process; this project would not have been possible without the help and guidance of the Apress team: many, many appreciative thanks to them and the book review team, particularly Jonathan Gennick. And finally, kudos and thanks to my awesome friends and coauthors (Syed, Kai, and Riyaj) for the amazing team effort and making this book happen.

—Tariq Farooq

I dedicate this book to you, the readers of the book. I thank my family—my wife Nisha and kids Shibi, Irfan, and Imran—for their great support and encouragement. Especially, I should thank my youngest son, Imran, for his understanding while Dad is busy writing "the book" (as he fondly quotes the book in the air!). My sincere thanks to Apress staff.

Many thanks to my current and past clients (you know who you are; I am thanking you without naming you) for giving me an opportunity to probe complex issues and learn so much more about RAC; Oracle-L listers and peers for asking me thought-provoking questions; and OakTable members for helping me to go to the next level when I am stuck on issues. My sincere thanks to the ContentConcepts guys (`http://contentconcepts.in`) for the preliminary editing of my chapters.

—Riyaj Shamsudeen

I want to thank all the people who have assisted this book, especially the Apress team of Acquisitions Editor Jonathan Gennick, Developmental Editor Tom Welsh, and Copy Editor Brendan Frost for their efforts, patience, and skills in transforming the technical writing into a finished book. I definitely would like to thank the technical reviewers, Arup Nanda and Sandesh Rao, for their time and effort in checking the technical content of the entire book as well as providing their very valuable comments and advice for improving the quality of this book.

I would like to dedicate this book to my parents, who encouraged me to pursue my education and career in computer technology. I would like to thank my wife Jin and my daughter Jessica for their consistent support and providing me the time to work on this book at night and on weekends.

I would like to thank my management at Dell—David Mar, Ibrahim Fashho, Dr. Reza Rooholamini—for their long-time support and encouragement and my colleagues in the Dell Oracle Solution Engineering team for their assistance and many great discussions on Oracle RAC, Red Hat Linux, and Oracle Linux and storage. I also want to specially thank my mentors in Oracle who really inspired me to work with the Oracle community: Dr. Nadia Bendjedou, a senior director from Oracle, and Erik Peterson, General Manager of Oracle Mexico Development Center. Last but certainly not least I would like to thank my coauthors Syed, Tariq, and Riyaj for the amazing teamwork. In particular, I want to thank Syed for his great leadership on the team and his confidence and encouragement.

—Kai Yu

Thank You

Want More Books?

We hope you learned what you expected to learn from this eBook. Find more such useful books on www.PlentyofeBooks.net

Learn more and make your parents proud :)

Regards

www.PlentyofeBooks.net