Guaranteed-Rate I/O Version 2 Guide

007–4244–001

CONTRIBUTORS

Written by Lori Johnson

Illustrated by Chrystie Danzer

Production by Karen Jacobson

Engineering contributions by Andrew Gildfind, Ken McDonell

COPYRIGHT

© 2004 Silicon Graphics, Inc. All rights reserved; provided portions may be copyright in third parties, as indicated elsewhere herein. No permission is granted to copy, distribute, or create derivative works from the contents of this electronic documentation in any manner, in whole or in part, without the prior written permission of Silicon Graphics, Inc.

LIMITED RIGHTS LEGEND

The electronic (software) version of this document was developed at private expense; if acquired under an agreement with the USA government or any contractor thereto, it is acquired as "commercial computer software" subject to the provisions of its applicable license agreement, as specified in (a) 48 CFR 12.212 of the FAR; or, if acquired for Department of Defense units, (b) 48 CFR 227-7202 of the DoD FAR Supplement; or sections succeeding thereto. Contractor/manufacturer is Silicon Graphics, Inc., 1500 Crittenden Lane, Mountain View, CA 94043-1351.

TRADEMARKS AND ATTRIBUTIONS

IRIX, Silicon Graphics, SGI, and the SGI logo are registered trademarks and CXFS is a trademark of Silicon Graphics, Inc., in the United States and/or other countries worldwide.

UNIX and the X device are registered trademarks of The Open Group in the United States and other countries.

Record of Revision

Version	Description
001	February 2004 Original publication

Contents

About This Guide . . . . **vii**
Related Publications vii
Obtaining Publications viii
Conventions viii
Reader Comments ix

1. Introduction **1**
What Does GRIO Do? 1
Terminology 2
GRIOv1 and GRIOv2 Differences 2
Overview 4

2. How GRIO Works **5**
Traffic Control 5
Stream Use and Real-Time File Setup 6
Software Components 6
`ggd2` Daemon 6
Qualified Bandwidth 7
Managing Bandwidth: Encapsulation and Distributed Bandwidth Allocator 7
GRIO Server Relocation and Recovery 9

3. Setting Up GRIO **11**
Installation Requirements 11
Deployment Considerations for Cluster Volumes 11
Data Layout 12

Choosing a Qualified Bandwidth 12
Local Volumes and Cluster Volumes 15
  Local Volume Domain Configuration 16
  Cluster Volume Domain Configuration 16
Licensing 17
Starting GRIO 18
  Starting GRIOv2 when GRIOv1 is Active 18
  Starting GRIOv2 when GRIOV1 is Not Active 18
Monitoring GRIO 19

4. GRIO API Overview 21
`grio_avail()` 21
`grio_bind()` 22
`grio_get_stream()` 22
`grio_modify()` 23
`grio_release()` 23
`grio_reserve()` and `grio_reserve_fd()` 24
`grio_unbind()` 26

Index 27

About This Guide

This publication provides information about GRIO version 2, the second-generation guaranteed-rate I/O product from SGI. It is supported with CXFS multiOS CXFS 3.1.1 or later plus any required patches.

Related Publications

The following publications contain additional information that may be helpful:

- *CXFS Administration Guide for SGI InfiniteStorage*
- *CXFS MultiOS Client-Only Guide for SGI InfiniteStorage*
- *IRIX Admin: Disks and Filesystems*
- *XVM Volume Manager Administrator's Guide*
- Man pages:
  - `fx`(1M)
  - `ggd2`(1M)
  - `griomon`(1M)
  - `grioqos`(1M)
  - `grioadmin`(1M)
  - `open`(2)
  - `grio_avail`(3X)
  - `grio_bind`(3X)
  - `grio_modify`(3X)
  - `grio_release`(3X)
  - `grio_reserve`(3X)
  - `grio_unbind`(3X)

- griotab(4)
- grio2(5)
- grioqos(5)

Obtaining Publications

You can obtain SGI documentation as follows:

- See the SGI Technical Publications Library at http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, release notes, man pages, and other information.
- If it is installed on your SGI system, you can use InfoSearch, an online tool that provides a more limited set of online books, release notes, and man pages. With an IRIX system, enter `infosearch` at a command line or select **Help > InfoSearch** from the Toolchest.
- On IRIX systems, you can view release notes by entering either `grelnotes` or `relnotes` at a command line.
- On Linux systems, you can view release notes on your system by accessing the `README.txt` file for the product. This is usually located in the `/usr/share/doc/`*productname* directory, although file locations may vary.
- You can view man pages by typing `man` *title* at a command line.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
variable	Italic typeface denotes variable entries and words or concepts being defined.

`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.)
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address:

  techpubs@sgi.com

- Use the Feedback option on the Technical Publications Library Web page:

  http://docs.sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  Technical Publications
  SGI
  1500 Crittenden Lane, M/S 535
  Mountain View, California 94043–1351

SGI values your comments and will respond to them promptly.

Chapter 1

Introduction

GRIO version 2 (GRIOv2) is the second-generation guaranteed-rate I/O product from SGI.

Note: When it is necessary to distinguish between the previous version (version 1) and the current version (version 2), this guide uses the terms *GRIOv1* and *GRIOv2*. Where the term *GRIO* is used without qualification, it refers to version 2.

What Does GRIO Do?

GRIO does the following:

- Enables a user application to reserve part of a system's I/O resources for its exclusive use
- Guarantees delivery of data from a storage device at a predefined rate, regardless of any other I/O activity on the system or on other nodes in the cluster
- Ensures that the rate at which a process issues I/O does not exceed its guarantee and will throttle the I/O if necessary

GRIO includes the following features:

- Support for CXFS filesystems shared among nodes in a cluster as well as locally attached XFS filesystems
- A simple filesystem-level performance qualification model (rather than the often complex device-qualification model used in GRIOv1)
- A range of tools for monitoring and measuring delivered bandwidth and I/O service time

Terminology

GRIO uses the following terminology:

- *Quality of service (QoS)* refers to the performance properties of a system service (such as worst-case bandwidth or I/O service time).
- *Qualified bandwidth* is the maximum bandwidth that can be delivered by a filesystem (and the XVM volume on which it resides) in a given configuration under a realistic application workload such that all applications are delivered an adequate quality of service.
- *Reservation* is the set of quality-of-service parameters requested by a user application. Reservation requests are forwarded to the `ggd2`(1M) bandwidth management daemon.
- *Guarantee* is the assurance made by the system to a user process that it will deliver data from a storage device at the reserved rate regardless of any other I/O activity on the system or on other nodes within its cluster.
- *Stream* is the object within the kernel that encodes the reservation's quality-of-service parameters and maintains the necessary scheduling and monitoring state required to fulfill the guarantee.

GRIOv1 and GRIOv2 Differences

Although you can have both the GRIOv1 and GRIOv2 subsystems installed on the same machine, only one of them can be active. For more information, see "Starting GRIO" on page 18.

Table 1-1 summarizes the primary differences between GRIOv1 and GRIOv2.

Table 1-1 Differences Between GRIOv1 and GRIOv2

	GRIOv1	GRIOv2
Reservation-granting daemon:	`ggd`	`ggd2`
Userspace library:	`libgrio`	`libgrio2`
Logical volumes:	XLV	XVM
Filesystems supported:	Local XFS filesystems only	Local XFS filesystems and shared CXFS filesystems
Multiple-node support:	No	Yes
Qualification model:	*Device-level:* the maximum sustainable bandwidth for each **hardware component** in the I/O path (including the storage devices themselves, the SCSI and Fibre Channel busses, system interconnects, and bridges), is qualified separately	*Filesystem-level:* the maximum sustainable bandwidth is measured across the entire **filesystem** under a realistic application workload. The qualified bandwidth is stored as follows: • XFS: in the `/etc/griotab` file • CXFS: in the cluster database

	GRIOv1	GRIOv2
Monitoring service	Limited administration tools	Comprehensive tools for measuring and monitoring delivered quality-of-service levels, including collection of per-stream performance metrics. You can use the information provided by the quality-of-service infrastructure to choose the tradeoff between resource utilization and delivered I/O performance that is most appropriate for a given application mix, workload, and production environment. For more information, see the `griomon`(1M), `grioqos`(1M), and `grioadmin`(1M) man pages.
Control of non-GRIO-managed I/O	No control	Cluster-wide encapsulation and control. When GRIOv2 begins managing a filesystem, every node with access to that filesystem is notified. From that point on, all user and system I/O that does not have an explicit reservation is encapsulated. This means that I/O that is not managed by GRIO is automatically associated with a system-managed kernel stream. The `ggd2` daemon allocates otherwise unused bandwidth to these streams, which allows non-GRIO–managed I/O to be processed even when there are active reservations in the system. `ggd2` dynamically adjusts the amount of bandwidth allocated for this purpose based on monitoring of filesystem demand and utilization.

For more information, see the `ggd2`(1M), `griotab`(4), `grioadmin`(1M), `grioqos`(1M), `griomon`(1M), and `grioqos`(5) man pages

Overview

This guide provides the information you need to administer GRIO. It discusses the following:

- Chapter 2, "How GRIO Works" on page 5
- Chapter 3, "Setting Up GRIO" on page 11
- Chapter 4, "GRIO API Overview" on page 21

How GRIO Works

This chapter discusses the following:

- "Traffic Control"
- "Stream Use and Real-Time File Setup" on page 6
- "Software Components" on page 6
- "ggd2 Daemon" on page 6
- "Qualified Bandwidth" on page 7
- "Managing Bandwidth: Encapsulation and Distributed Bandwidth Allocator" on page 7
- "GRIO Server Relocation and Recovery" on page 9

Traffic Control

GRIO is a component on the XFS and CXFS I/O path that runs in every node with access to a GRIO-managed filesystem. When active, all I/O on a machine and in the cluster is controlled by the GRIO scheduler.

I/O falls into the following categories:

- *GRIO I/O*: I/O for applications that have made an explicit GRIO reservation
- *Non-GRIO I/O*: all other buffered and system I/O

GRIO ensures that applications with reserved bandwidth receive data at the requested rate, regardless of other I/O activity on the node and elsewhere within the cluster. GRIO will throttle an application if it attempts to use more bandwidth than it has reserved.

Stream Use and Real-Time File Setup

In order to use a GRIO reservation, a file must be read or written using direct, synchronous I/O requests. The `open`(2) man page describes the use and buffer alignment restrictions of the direct I/O interface. A GRIO reservation can be made for any file within an XFS or CXFS filesystem created on an XVM volume. However, for optimal performance, files should be created on a dedicated real-time subvolume.

To allocate a file on the real-time subvolume of an XFS or CXFS filesystem, you must use the `fcntl`(2) `F_FSSETXATTR` command to set the `XFS_XFLAG_REALTIME` flag. You can only issue this command on a newly created file. It is not possible to mark a file as real-time after non-real-time data blocks have been allocated to it.

Software Components

GRIO functionality is distributed between the following main components:

- `ggd2` daemon (see "ggd2 Daemon" on page 6)
- `libgrio2` library, which implements the GRIO userspace API (see Chapter 4, "GRIO API Overview" on page 21)
- The operating system kernel, including the following:
  - Stream management
  - I/O scheduler
  - Cluster integration
  - Messaging

`ggd2` Daemon

The `ggd2`(1M) daemon is a user-level process started at system boot time that manages the I/O bandwidth for a collection of XVM volumes. It does the following:

- Activates/deactivates the GRIO kernel scheduler
- Processes client requests to reserve and release bandwidth
- Tracks bandwidth utilization

- Manages unallocated bandwidth
- Prevents oversubscription
- Enforces GRIO software licenses
- Broadcasts to the relevant kernels the amount of bandwidth per filesystem that may be used for non-GRIO I/O

Qualified Bandwidth

The qualified bandwidth for a filesystem is the maximum I/O load that it can sustain while still satisfying requirements for delivered bandwidth and I/O service time.

You must determine a specific qualified bandwidth for each GRIO-managed filesystem.

The qualified bandwidth is specified in the `griotab` file for local IRIX filesystems or in the cluster database for shared filesystems.

The `ggd2` daemon is responsible for managing the allocation of available bandwidth between different applications.

You can adjust the qualified bandwidth as needed to make the best use of your system, taking into account the tradeoff between resource utilization and delivered I/O performance. For more information, see "Choosing a Qualified Bandwidth" on page 12.

Managing Bandwidth: Encapsulation and Distributed Bandwidth Allocator

The `ggd2` daemon tracks the total qualified bandwidth and ensures that the total workload never exceeds the qualified bandwidth.

When `ggd2` begins managing a filesystem, every node with access to that filesystem is notified. Each node in turn creates a dedicated system stream for that filesystem, which is called the *non-GRIO stream*. From that point on, all user and system I/O that does not have an explicit GRIO reservation is encapsulated by this stream and then managed by the GRIO scheduler. For a local filesystem, there is a single non-GRIO stream. For a shared filesystems, there is a non-GRIO stream on each node with access to the filesystem.

Note: The scheduling for all non-GRIO I/O within a GRIO-managed filesystem received from different applications and system services is on a first-come-first-served basis.

To keep the total throughput of the filesystem high even when there are active GRIO streams, `ggd2` attempts to allocate the unreserved portion of the qualified bandwidth for use by non-GRIO applications. This bandwidth is effectively lent for short periods of time until `ggd2` receives a new request for guaranteed-rate bandwidth, at which point it is reclaimed. GRIO applications have priority over non-GRIO applications.

The `ggd2` daemon periodically adjusts the amount of bandwidth allocated to the individual non-GRIO streams for its managed filesystems. This functionality is referred to as the *distributed bandwidth allocator* (DBA). The DBA is responsible for determining how unreserved bandwidth is distributed between the nodes with access to the filesystem. By default, the DBA runs every two seconds, constantly allocating free bandwidth to nodes based on a range of dynamically monitored demand and utilization metrics.

Calls to reserve bandwidth may block until the next DBA cycle. Therefore, applications must be prepared for delays when setting up guaranteed-rate streams. Refer to `grio_reserve()`(3X) for more information. To help manage this, you can use the `-r` option to cause `ggd2` to keep a cache (or reserve) of bandwidth that is unavailable for non-GRIO use, from which new GRIO reservations can be processed directly. Any additional bandwidth that is available is free to be used by either future GRIO or DBA streams. Figure 2-1 represents these concepts.

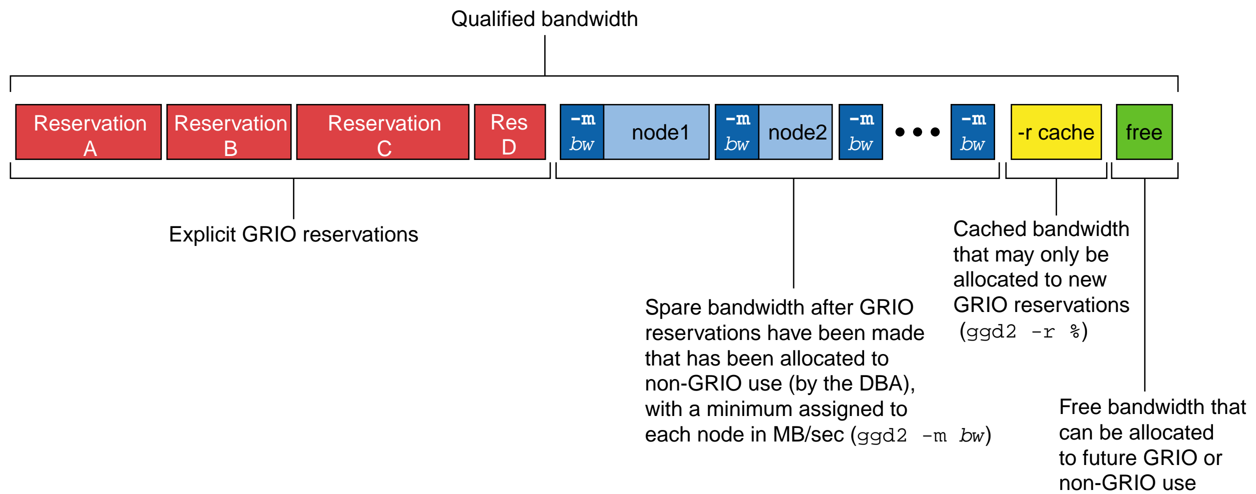

Figure 2-1 Qualified Bandwidth

A user process can request a reservation using the `grio_reserve()` and `grio_reserve_fd()` library calls. Requests are forwarded to the `ggd2` that is actively managing the target volume domain. Requests to filesystems in the local domain are immediately sent to the local instance of `ggd2`. Requests to filesystems cluster domain are forwarded to the GRIO server, which may be running on a different node in the cluster.

GRIO Server Relocation and Recovery

Each instance of `ggd2` maintains reservation and bandwidth state that must be kept consistent with one or more kernels.

If `ggd2` fails, a new `ggd2` instance will receive from the local kernel all of the information necessary to reestablish the following:

- Local volume reservations
- Cluster volume reservations (if the `ggd2` that failed was the GRIO server for the cluster)
- All of the DBA state for non-GRIO I/O

If GRIO server node fails, a new GRIO server is automatically elected and all of the cluster volume reservations and DBA state is reestablished by that instance of `ggd2`.

You can also choose to manually migrate the GRIO server to another CXFS administration node in the cluster.

Setting Up GRIO

This chapter discusses the following:

- "Installation Requirements"
- "Deployment Considerations for Cluster Volumes"
- "Data Layout" on page 12
- "Choosing a Qualified Bandwidth" on page 12
- "Local Volumes and Cluster Volumes" on page 15
- "Licensing" on page 17
- "Starting GRIO" on page 18
- "Monitoring GRIO" on page 19

Installation Requirements

To operate in local volume domain on an IRIX node, you must install the `eoe.sw.grio2` product.

To enable clustered GRIO support on IRIX, you must install both `eoe.sw.grio2` and `cxfs.sw.grio2_cell`.

In a cluster deployment, **every** node must be GRIO-enabled. Consult the CXFS multiOS release notes to determine whether your platform has been GRIO-enabled.

Deployment Considerations for Cluster Volumes

You must observe the following constraints when setting up GRIO filesystems:

- If any of the logical units (LUNs) on a particular device will be managed as GRIO filesystems, then all of the LUNs should be managed as GRIO filesystems. Typically, there will be hardware contention between separate LUNs, both in the storage area network (SAN) and within the storage device. If only a subset of the LUNs are managed, I/O to the unmanaged LUNs could still cause

oversubscription of the device and could in turn violate guarantees on the managed filesystems.

- A storage device containing GRIO–managed filesystems should not be shared between clusters. The GRIO daemons running within different clusters are not coordinated, and unmanaged I/O from one cluster can cause guarantees in the other cluster to be violated.

Data Layout

To set up a filesystem on a RAID device such that you achieve correct filesystem device alignment and maximize I/O performance, remember to do the following:

- Ensure that each data partition is correctly aligned with the internal disk layout of its LUN
- Set XVM stripe parameters correctly
- Pass correct volume geometry (stripe unit and width) to `mkfs_xfs`(1)

For more information, see the `grio2`(5) man page.

Choosing a Qualified Bandwidth

You can adjust the qualified bandwidth to reflect the specific trade-off between delivered quality of service and utilization of the storage infrastructure for your situation.

The following affect the qualified bandwidth you will choose:

- The hardware configuration
- The application work flow and I/O load
- The specific quality–of–service requirements of applications and users

Typically, the first concern is that the required bandwidth can be delivered by the storage system. The second concern is the timeliness or service times observed for individual I/Os.

Determining qualified bandwidth is an iterative process. There are several strategies you can use to determine and fine-tune the qualified bandwidth for a filesystem. For example:

- Establish a given bandwidth and then adjust so that the quality-of-service requirements are met. Do the following:
  1. Make an initial estimate of the qualified bandwidth. You can use the fixed storage architecture parameters (RAID performance, number of HBAs, etc.) to estimate the anticipated peak bandwidth that can be delivered. The qualified bandwidth is then determined as an appropriate fraction of this peak.
  2. Configure `ggd2` (either using `griotab` or the cluster database) appropriately.
  3. Run a test workload.
  4. Monitor the delivered performance.
  5. Refine the estimate as needed.
- Establish that quality–of–service requirements are satisfied and then adjust to maximize throughput. To do this, increase the load until the storage system can no longer meet the application quality-of-service requirements; the qualified bandwidth must be lower than this value.
- Explore the space of possible workloads and test whether a given workload satisfies both bandwidth and application quality-of-service requirements.

Although the hardware configuration provides a basis for calculating an estimate, remember that the qualified bandwidth is also affected by the particular work-flow issues and the quality-of-service requirements of individual applications. For example, an application that has large tunable buffers (such as a flipbook application that does aggressive RAM caching) can tolerate a greater variation in I/O service time than can a media broadcast system that must cue and play a sequence of very short clips. In the first example, the qualified bandwidth would be configured as a larger proportion of the sustained maximum. In the second example, the qualified bandwidth might be reduced to minimize the system utilization levels and improve I/O service times.

A high qualified bandwidth will generally achieve the greatest overall throughput but with the consequence that individual applications may intermittently experience longer service times for some I/Os. This variation in individual service times is referred to as *jitter*; as the storage system approaches saturation, service-time jitter will typically increase. A lower qualified bandwidth means that total throughput will be reduced, but because the I/O infrastructure is under less stress, individual requests

will typically be processed with less variation in individual I/O service times. Figure 3-1 illustrates these basic ideas. The precise relationship between load on the storage system and variation in I/O service time is highly dependent on your storage hardware.

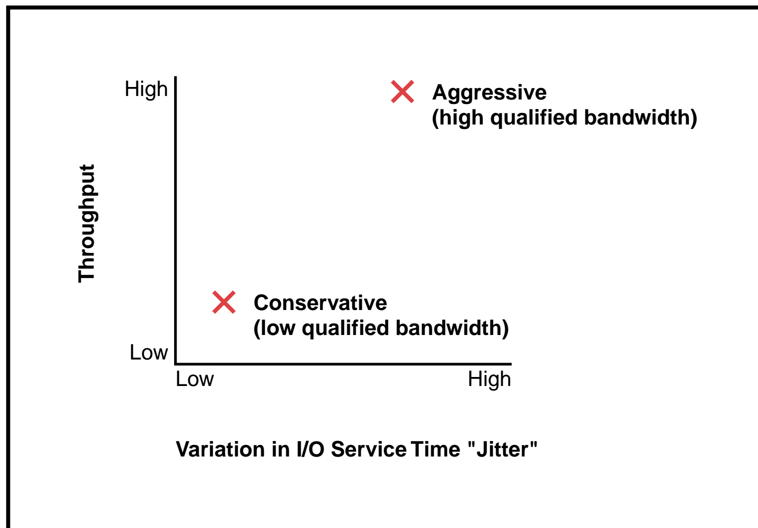

Figure 3-1 Tradeoff Between Throughput and Variation in I/O Service Time (Jitter)

Some storage devices (particularly those with real-time schedulers) can provide a fixed bound on I/O service time even at utilization levels close to their maximum. In this case, the qualified bandwidth can be set higher even where applications have tight quality-of-service requirements. The user-adjustable qualified bandwidth provides the flexibility required for GRIO to work with both dedicated real-time devices as well as more common off-the-shelf storage systems.

Note: In all cases, you must verify the chosen qualified bandwidth by testing the storage system under a realistic workload.

You can use the `grioqos`(1M) tool to measure the delivered quality-of-service performance. This tool extracts quality-of-service performance for an active stream without disturbing the application or the kernel scheduler. GRIO maintains very detailed performance metrics for each active stream. Using the `grioqos` command

while running a workload test lets you answer questions such the following for every active stream in the system:

- What has been the worst observed bandwidth over a 1-second period?
- What is the worst observed average I/O service time for a sequence of 10 I/Os?

For more information about GRIO tools and the mechanisms for accessing quality-of-service data within the kernel, see the `grioqos`(1M) and `grioqos`(5) man pages.

Local Volumes and Cluster Volumes

A managed volume can be one of the following:

- A *local volume* is attached to the node in question. This volume is in the *local volume domain*.

  Local volumes are always managed by the instance of the `ggd2` daemon running on the node to which they are attached.

- A *cluster volume* is used with CXFS filesystems and is shared among nodes in a cluster. This volume is in the *cluster volume domain*.

  All cluster volumes are managed by a single instance of the `ggd2` daemon running on one of the CXFS administration nodes in the cluster; this node is referred to as the *GRIO server.* There is one GRIO server per cluster.

  The GRIO server is elected automatically. You can relocate it by using the `grioadmin`(1M) command. The GRIO server must be a CXFS administration node. Client-only nodes will never be elected as GRIO servers.

  If a given CXFS administration node has locally attached volumes and has also been selected as the GRIO server, then the `ggd2` running on that node will serve dual-duty and will manage both its own local volume domain and the cluster volume domain.

For more information about CXFS, see "Cluster Volume Domain Configuration" on page 16 and *CXFS Administration Guide for SGI InfiniteStorage*.

Local Volume Domain Configuration

To configure GRIO for local volume domains, you must provide information in the /etc/griotab file.

The /etc/griotab file lists the volumes that should be managed by GRIO and the maximum qualified bandwidth they can deliver. This file is read at startup and whenever ggd2 receives a SIGHUP signal (such as when you issue a killall -HUP ggd2 command). See the ggd2(1M) and griotab(4) man pages for more information.

Cluster Volume Domain Configuration

You must use the cmgr(1M) cluster configuration tool to configure cluster volumes for GRIO.

To mark a filesystem as GRIO-managed and set its qualified bandwidth, use the following commands:

```
# /usr/cluster/bin/cmgr
Welcome to SGI Cluster Manager Command-Line Interface

cmgr> modify cxfs_filesystem fsname in cluster clustername
cmgr> set grio_managed to true
cmgr> set grio_qualified_bandwidth to qualified_bandwidth
cmgr> done
```

The value for *qualified_bandwidth* is specified in bytes per second. For example, the following sets the qualified bandwidth to 200 MB/s (200*1024*1024):

```
cmgr> set grio_qualified_bandwidth to 209715200
```

To show the current status of a shared filesystem:

```
cmgr> show cxfs_filesystem fsname in cluster clustername
...
               GRIO Managed Filesystem: true
               GRIO Managed Bandwidth: qualified_bandwidth
...
```

Note: In cmgr, you must unmount a filesystem before you can modify it.

A prompting mode is also available for `cmgr`. For more information, see the *CXFS Administration Guide for SGI InfiniteStorage*.

If you have installed the `cxfs.sw.grio2_cell` subsystem and turned on GRIO, the `ggd2` daemon will automatically query the cluster configuration database for GRIO volume configuration information. `ggd2` dynamically tracks updates to the cluster database.

Licensing

The GRIO FLEXlm licensing regime controls a number of configuration parameters including the total number of active streams and the total aggregate qualified bandwidth of filesystems under management. Separate license types are provided for the local and cluster volume domains, and license constraints are enforced for each volume domain separately.

The `ggd2` daemon checks the license at startup, whenever it detects a configuration change, or when it receives a `SIGHUP` signal.

License enforcement for streams is straightforward. The license for a given volume domain specifies a maximum number of active streams. All reservation requests above this limit are denied.

In the case of bandwidth, a license specifies the maximum total aggregate qualified bandwidths for all volumes within the volume domain. The `ggd2` daemon validates the configuration at startup and whenever the configuration is changed:

- For the local domain, `ggd2` tracks changes to `/etc/griotab` (`ggd2` is notified of changes with a `SIGHUP`)
- For the cluster volume domain, `ggd2` tracks the relevant cluster database entries for cluster volume qualified bandwidth

If the configuration of a volume domain is altered and becomes unlicensed, `ggd2` enters a passive mode in which all further requests pertaining to that domain, with the exception of release requests, are denied. A message is sent to the system log and that volume domain will remain deactivated until the configuration returns to a licensed state, at which time another message will be logged indicating the domain is again active.

For more information, see the `license.dat`(5) man page.

Starting GRIO

Although you can have both the GRIOv1 subsystem and the GRIOv2 subsystem installed on the same machine, only one of these subsystems can be active. The subsystem that is turned `on` in `chkconfig` is started by default at boot time and remains in effect until the `chkconfig` setting is changed and the machine is rebooted.

Starting GRIOv2 when GRIOv1 is Active

Suppose you were running GRIOv1 and wanted to switch to GRIOv2. After performing the configuration tasks discussed in this guide, you would do the following:

1. Turn off GRIOv1 (`grio`) and turn on GRIOv2 (`grio2`):

```
# chkconfig grio off
# chkconfig grio2 on
```

2. Reboot the system to allow the kernel to be reinitialized with the GRIOv2 scheduler.

You do not need to manually start GRIOv2 because the daemon is automatically started upon reboot when the `chkconfig` setting is `on`.

Note: If GRIOv1 is still enabled when you perform a GRIOv2 library call, the return will be `ENOSYS`. If you do not have either the GRIOv1 or GRIOv2 kernel initialized, the return will be `EAGAIN`, indicating that the subsystem has not yet initialized and the application should retry the request.

Starting GRIOv2 when GRIOV1 is Not Active

If you have not run GRIOv1 during the current boot session, you can start GRIOv2 by doing the following:

1. Turn on GRIOv2:

```
# chkconfig grio2 on
```

2. Start GRIOv2:

```
# /etc/init.d/grio2 start
```

You must perform the manual start only once. When the machine is rebooted, GRIOv2 will be restarted automatically as long as its chkconfig setting remains on.

Monitoring GRIO

You can use the griomon(1M) tool to monitor active streams within the system and display their high-level performance metrics, such as the currently allocated bandwidth and total bytes transferred.

GRIO API Overview

User processes communicate with the ggd2 daemon using the following core library calls:

- "grio_avail()"
- "grio_bind()" on page 22
- "grio_get_stream()" on page 22
- "grio_modify()" on page 23
- "grio_release()" on page 23
- "grio_reserve() and grio_reserve_fd()" on page 24
- "grio_unbind()" on page 26

The process that initially reserves bandwidth by calling grio_reserve() or grio_reserve_fd() is referred to as the *owning process*. Any streams not already released when their owning process exits will be automatically released. Processes can share streams. The ownership of a GRIO stream is nontransferable.

grio_avail()

Synopsis:

```
#include <grio2.h>

int grio_avail(
         const char *fs,
         grio_off_t *bytes, grio_msecs_t *msecs)

cc ... -lgrio2
```

The grio_avail() call returns the currently available guaranteed-rate bandwidth for a specified filesystem. The returned bandwidth is the qualified bandwidth of the filesystem minus bandwidth reserved for active GRIO streams and any bandwidth statically allocated for nonguaranteed-rate I/O. While ggd2 temporarily allows unreserved bandwidth to be used for servicing nonguaranteed-rate I/O, this

bandwidth is reclaimed when a GRIO reservation is received and is therefore considered available.

For more information, see the grio_avail(3X) man page.

grio_bind()

Synopsis:

```
#include <grio2.h>

int grio_bind(grio_descriptor_t fd, grio_stream_id_t *stream_id);

          cc ... -lgrio2
```

The grio_bind() call binds one or more open file descriptors to a GRIO stream. Once bound, all I/O to or from the file descriptors will receive the quality-of-service guarantees of the stream.

Binding a file descriptor increments the reference count of a stream by 1. The file descriptor remains bound to the stream until it is either closed or explicitly unbound with grio_unbind().

The file descriptor must be capable of GRIO I/O. That is, it must refer to an open file on an XFS or CXFS filesystem and be configured for direct I/O.

For more information, see the grio_bind(3X) man page.

grio_get_stream()

Synopsis:

```
#include <grio2.h>

int grio_get_stream(
          grio_descriptor_t fd,
          grio_stream_id_t *stream_id);

cc ... -lgrio2
```

The grio_get_stream() call returns the ID of the stream to which it is bound.

For more information, see the grio_get_stream(3X) man page.

grio_modify()

Synopsis:

```
#include <grio2.h>

int grio_modify(
          grio_stream_id_t *stream_id,
          grio_off_t *bytes, grio_msecs_t *msecs,
          int flags)

cc ... -lgrio2
```

The grio_modify() call changes the properties of an existing GRIO stream. You can increase or decrease reserved bandwidth by using this call.

Note: grio_modify() is a synchronous call and, when increasing a reservation, may block while bandwidth is reallocated. This delay can be in the order of 1 or 2 seconds and applications should be designed to accommodate this delay if necessary. While a call to grio_modify() is being processed, I/O to the stream continues uninterrupted at its existing rate.

For more information, see the grio_modify(3X) man page.

grio_release()

Synopsis:

```
#include <grio2.h>

int grio_release(grio_stream_id_t *stream_id)

cc ... -lgrio2
```

The grio_release() call removes a GRIO stream ID from the system and releases the primary reference taken when it was created. When all remaining references to

the stream are removed, the stream will be destroyed and its associated bandwidth will be returned to the system.

The grio_release() call hides the stream. Attempts to bind new file descriptors using grio_bind() will fail with a return value of ENOENT. However, the quality-of-service guarantees of the stream will remain in effect until all remaining bound file descriptors are either unbound or closed, and any in-flight I/O to the stream completes.

This behavior gives an application some flexibility in controlling the extent of a GRIO guarantee. For instance, by binding a file descriptor to a stream and immediately releasing the stream, an application can create a temporary reservation that persists for as long as the file descriptor remains open. Alternatively, if a process does not explicitly release a stream, the guarantee persists until that process exits.

For more information, see the grio_release(3X) man page.

grio_reserve() and grio_reserve_fd()

Synopsis:

```
#include <grio2.h>

int grio_reserve(
          const char *path,
          grio_off_t *bytes, grio_msecs_t *msecs,
          int flags,
          grio_stream_id_t *stream_id)

int grio_reserve_fd(
          grio_descriptor_t fd,
          grio_off_t *bytes, grio_msecs_t *msecs,
          int flags,
         grio_stream_id_t *stream_id)

cc ... -lgrio2
```

The `grio_reserve()` and `grio_reserve_fd()` calls reserve guaranteed rate bandwidth to or from a GRIO-managed filesystem. If successful, they set up a GRIO stream in the kernel with the requested properties and return its stream ID:

- `grio_reserve()` makes a filesystem-level reservation. The target filesystem is identified with a path that must be either the filesystem mount point or the device special file on which it is located. Before guaranteed rate I/O can be performed, an open file descriptor must be bound to the new stream using `grio_bind()`.
- `grio_reserve_fd()` makes a file-level reservation. It takes an open file descriptor on the target filesystem. In addition to reserving bandwidth, the file descriptor is bound to the newly created stream. The file must therefore be suitable for guaranteed rate I/O and satisfy the normal requirements of `grio_bind`.

The requested bandwidth is specified as the number of bytes delivered every *msecs* milliseconds. *msecs* is referred to as the *reservation interval*. This value provides additional information to the GRIO scheduler about an application's ability to tolerate variation in I/O service time. (For example, an application request of 1MB every tenth of a second suggests a tighter requirement than 100 MB delivered every second, even though both requests describe the same average data rate.) GRIO uses this information as a hint only, and honors the expressed bandwidth over an implementation-defined scheduling interval.

`grio_release()` should be called when the stream is no longer required.

The process that creates a stream with these calls is said to be the *owning process*. Any streams not already released when their owning process exits will be automatically released. The ownership of a GRIO stream is not transferable.

GRIO streams are reference-counted. When created, a new stream has a reference count of 1. This primary reference remains until the reservation is released using `grio_release()` or the owning process exits.

A stream persists until its reference count drops to 0. Binding a file descriptor using `grio_bind()` adds a reference. Unbinding using `grio_unbind()` or closing a file descriptor removes a reference. In-flight I/O will also add references to a stream for short periods of time.

It is possible, and frequently useful, for a stream to persist after it has been released. For more information, see the `grio_release()` man page.

Note: Both `grio_reserve()` and `grio_reserve_fd()` are synchronous calls and may block while bandwidth is reallocated. This delay is referred to as the *stream creation latency*.

In the worst case, this delay can be in the order of 1 or 2 seconds, although it may be significantly less depending on the configuration of a particular GRIO deployment.

The following are strategies to minimize the impact of this behavior:

- Reserve bandwidth well ahead of time
- Perform reservations in a dedicated thread
- Reuse a reservation wherever possible
- Configure `ggd2` to keep a proportion of the available free bandwidth uncommitted using the `-r` option.

For more information, see the `grio_reserve`(3X) and `ggd2`(1M) man pages.

grio_unbind()

Synopsis:

```
#include <grio2.h>

int grio_unbind(grio_descriptor_t fd);

cc ... -lgrio2
```

The `grio_unbind()` call unbinds a file descriptor from its GRIO stream. Unbinding a file descriptor decrements the reference count of its stream by 1.

Once unbound, I/O to or from the file descriptor may continue, but will be scheduled as regular, non-guaranteed rate I/O.

For more information, see the `grio_unbind`(3X) man page.

Index

A

administration node, 15
administration tools, 4
API overview, 21
available guaranteed-rate bandwidth, 22

B

bandwidth management, 7
binding open file descriptors, 22
buffered I/O, 5

C

change a GRIO stream, 23
chkconfig, 18
client-only node, 15
cluster database, 4, 13
cluster integration, 6
cluster volume, 15
cluster volume domain, 15
clusters
  See "CXFS", 12
cmgr, 16
configuration
  cluster volume, 16
  local volume, 16
CXFS, 1, 3, 6, 11, 11, 16, 22
CXFS administration node, 15
CXFS client-only node, 15
cxfs.sw.grio2_cell, 11, 17

D

daemon, 3
data layout, 12
DBA, 8
device alignment, 12
device-level qualification, 3
differences between GRIOv1 and GRIOv2, 2
distributed bandwidth allocator, 8
domains, 15

E

EAGAIN, 18
encapsulation of non-GRIO I/O, 4
ENOENT, 24
ENOSYS, 18
eoe.sw.grio2, 11
/etc/griotab, 3, 16

F

F_FSSETXATTR, 6
fcntl, 6
features, 1
file allocation and stream use, 6
file descriptor binding/unbinding, 22
file-level reservation, 25
filesystem-level
  qualification, 3
  reservation, 25
filesystems supported, 3
FLEXlm licensing, 17
flipbook application, 13
FS_XFLAG_REALTIME, 6

G

ggd, 3
ggd2, 3, 6, 7, 9, 13, 15, 17, 21, 22, 26
GRIO server, 15
GRIO-enabled platforms, 11
grio_avail(), 21
grio_bind(), 22, 24, 25
grio_get_stream(), 22
grio_modify(), 23
grio_release(), 23, 25
grio_reserve(), 8, 25
grio_reserve_fd(), 9, 25
grio_unbind(), 22, 25, 26
grioadmin, 4
griomon, 4
grioqos, 4, 14
griotab, 3, 13, 16
GRIOv1 and GRIOv2 differences, 2
guarantee, 2

H

HBA number, 13
how GRIO works, 5

I

I/O performance, 12
I/O scheduler, 6
I/O service times, 13
installation requirements, 11
introduction, 1

J

jitter, 13

L

latency in stream creation, 26
libgrio, 3
libgrio2, 3
library, 3
library calls, 21
license.dat, 18
licensing, 17
local volume, 15
local volume domain, 15
logical unit (LUN) management, 12
logical volumes, 3
LUN, 12
LUN management, 12

M

messaging, 6
modify a GRIO stream, 23
modify cxfs_filesystem, 16
monitoring GRIO, 21
monitoring service, 4
multiOS release (CXFS), 11
multiple-node support, 3

N

non-GRIO managed I/O, 4
non-GRIO stream, 7

O

open, 6
oversubscription, 7
overview, 4
owning process, 21, 25

Q

QoS, 2
qualification model, 3
qualified bandwidth, 2, 3, 12
quality of service, 2

R

RAID, 12, 13
RAM caching, 13
real-time data blocks, 6
real-time schedulers, 14
real-time subvolume, 6
releasing a GRIO stream, 23
releasing open file descriptors, 22
relocation and recovery of the GRIO server, 9
remove a GRIO stream, 23
reservation, 2
reservation interval, 25
reserve bandwidth, 25

S

SAN, 12
scheduler, 5
SIGHUP signal, 16, 17
software components, 6
starting GRIO, 18
storage area network (SAN), 12
stream, 2, 21, 23
  creation latency, 26
  management, 6
  use and file allocation, 6
stream ID, 23
stripe parameters, 12
stripe unit, 12

T

terminology, 2
testing, 14
traffic control, 5
turning GRIO on, 18

U

unallocated bandwidth, 7
unbind a file descriptor, 22, 26
userspace library, 3

V

volume geometry, 12
volumes, 3, 15

X

XFS, 3
XLV, 3
XVM, 3
XVM stripe parameters, 12