Getting Started With BDSpro

Document Number 007-3274-002

CONTRIBUTORS

Written by Pam Sogard
Illustrated by Dany Galgani
Edited by Judy Helfand
Production by Cindy Stief
Engineering contributions by Larry McVoy and Ethan Solomita
St Peter's Basilica image courtesy of ENEL SpA and InfoByte SpA. Disk Thrower image courtesy of Xavier Berenguer, Animatica.

Getting Started With BDSpro
Document Number 007-3274-002

Contents

List of Figures

List of Tables

About This Guide

Getting Started With BDSpro explains how to add BDSpro™, the Silicon Graphics implementation of the Bulk Data Service protocol (XBDS™), to a Network File System (NFS®) implementation. This guide contains information to help server and site administrators understand the BDS protocol, evaluate the suitability of BDSpro for their site, and include BDSpro in an existing NFS implementation. To use this guide successfully, administrators should be experienced in managing NFS and XFS™ (the X filesystem) on large servers.

Note: This guide also contains information to help application developers make use of BDS services.

What This Guide Contains

This guide comprises three chapters, which cover these topics:

- Chapter 1, “BDS Fundamentals” explains how BDS works, describes its advantages over standard NFS implementations, and explains the conditions under which it should be used.
- Chapter 2, “Preparing for BDSpro” describes hardware and software requirements for BDSpro and explains how to modify your existing configuration if you determine that changes are needed.
- Chapter 3, “Setting Up and Testing BDSpro” explains how to include BDS in an existing NFS implementation and check BDSpro performance.

Conventions Used in This Guide

These type conventions and symbols are used in this guide:

Bold	Literal command-line arguments, such as options and flags
Italics	Executable names, filenames, IRIX commands, manual and book titles, and new terms
`Fixed-width type`	Error messages, prompts, and onscreen text
`Bold fixed-width type`	User input, including keyboard keys (printing and nonprinting)
“”	(Double quotation marks) References in text to document section titles
()	(Parentheses) Following IRIX commands, surround reference page (man page) section numbers
#	IRIX shell prompt for the superuser (*root*)
%	IRIX shell prompt for users other than superuser

Related Documentation

These documents contain additional information that is related to BDSpro:

- *BDSpro Release Notes*
- *IRIX Admin: Disks and Filesystems 007-2835-xxx*
- *IRIX HIPPI Administrator’s Guide 007-2229-xxx*
- *ONC3/NFS Administrator’s Guide 007-0850-xxx*

See also the online reference pages (man pages): bds(1M), lmdd(1), mount(1M), exportfs(1M), fstab(4), xlv_assemble(1M), xlv_make(1M), fcntl(2), open(2), read(2), write(2), filesystems(4), and malloc(3C).

Chapter 1

BDS Fundamentals

Bulk Data Service (BDS) is a non-standard extension to NFS that handles file transactions over high-speed networks. BDS exploits the data access speed of the XFS filesystem and data transfer rates of high-speed networks, such as the high performance parallel interface (HIPPI) and fiberchannel, to accelerate standard NFS performance. BDSpro is the Silicon Graphics implementation of XBDS.

This chapter contains the following sections to help you understand and evaluate BDS performance:

- “What BDS Offers”
- “How BDS Works”
- “BDS and Buffering”
- “Write Buffering”
- “When BDS Makes Sense”

BDS Requirements

You can use BDSpro on Silicon Graphics systems running IRIX 6.2 (or later). The NFS product must also be installed on BDS hosts, and hosts must be connected to a high-speed network (such as HIPPI or fiberchannel) running the transmission control protocol / internet protocol (TCP / IP) suite.

Note: BDSpro 1.0 also requires that applications using it perform page-aligned input and output (I / O).

Figure 1-1 illustrates the XBDS protocol relative to existing Open Network Computing (ONC) protocols.

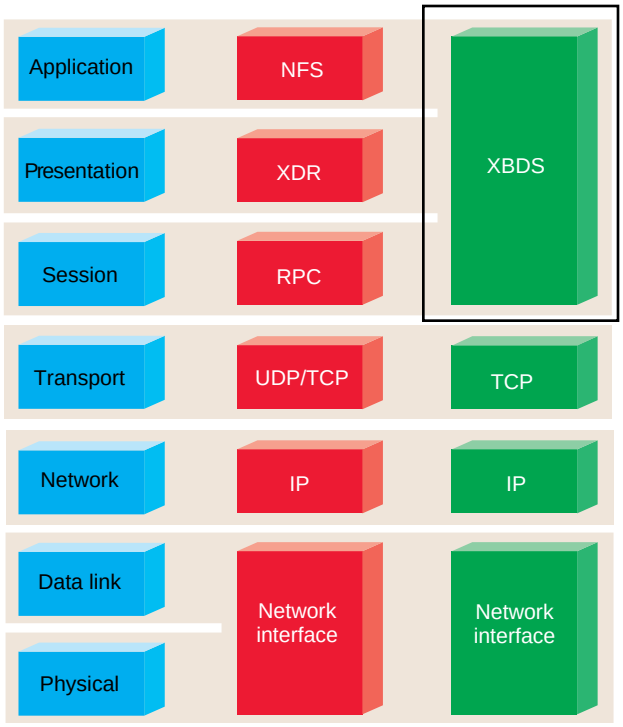

Figure 1-1 XBDS Protocol Compared with ONC Protocols

What BDS Offers

BDS is implemented as enhancements to NFS on the client system and a daemon process on the server. Figure 1-2 illustrates the BDSpro client-server model and the NFS client-server model on Silicon Graphics systems.

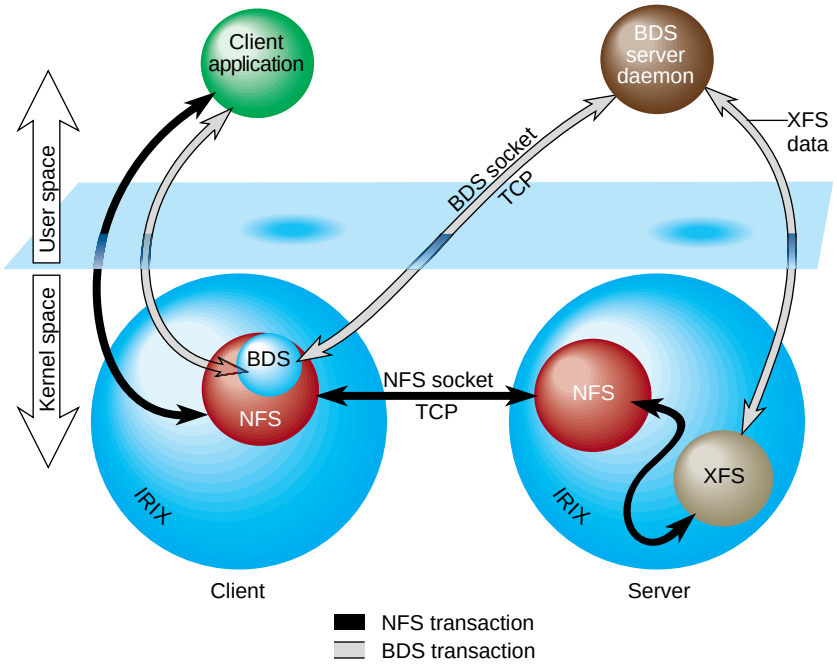

Figure 1-2 The BDSpro Client-Server Model

The hardware and software used on a network and its loading patterns determine the ultimate speed of NFS and BDS transactions. Because these factors vary greatly on individual networks, it is impossible to predict the performance gains that BDS will deliver to a particular network. However, to guage BDS advantages over standard NFS, it is useful to compare BDSpro to NFS performance under ideal network conditions.

Table 1-1 compares BDSpro transfer speeds with NFS configurations.

Table 1-1 BDSpro Performance Compared With Standard NFS

Product	Network Configuration	Read Rate
NFS (version 2)	UDP over HIPPI	2.5 MB per second per channel
NFS (version 3)	TCP/IP over HIPPI	19 MB per second per channel
BDSpro	TCP/IP over HIPPI	88 MB per second per channel

How BDS Works

To achieve high throughput, BDS relies on the ability of the operating system to perform *direct* input and output (see the O_DIRECT option on the open(2) IRIX reference page for details). With direct I/O, the operating system reads and writes data from disk directly to a user buffer, bypassing an intermediate copy to the kernel buffer cache that is standard for other types of I/O.

In a network transaction such as an NFS read or write, the time saved by bypassing the buffer cache is doubled, since the bypass occurs on both the client and the server systems. Figure 1-3 and Figure 1-4 and the discussions that follow them illustrate this difference in detail.

Standard NFS Transactions

Figure 1-3 illustrates the sequence of events in a standard NFS transaction.

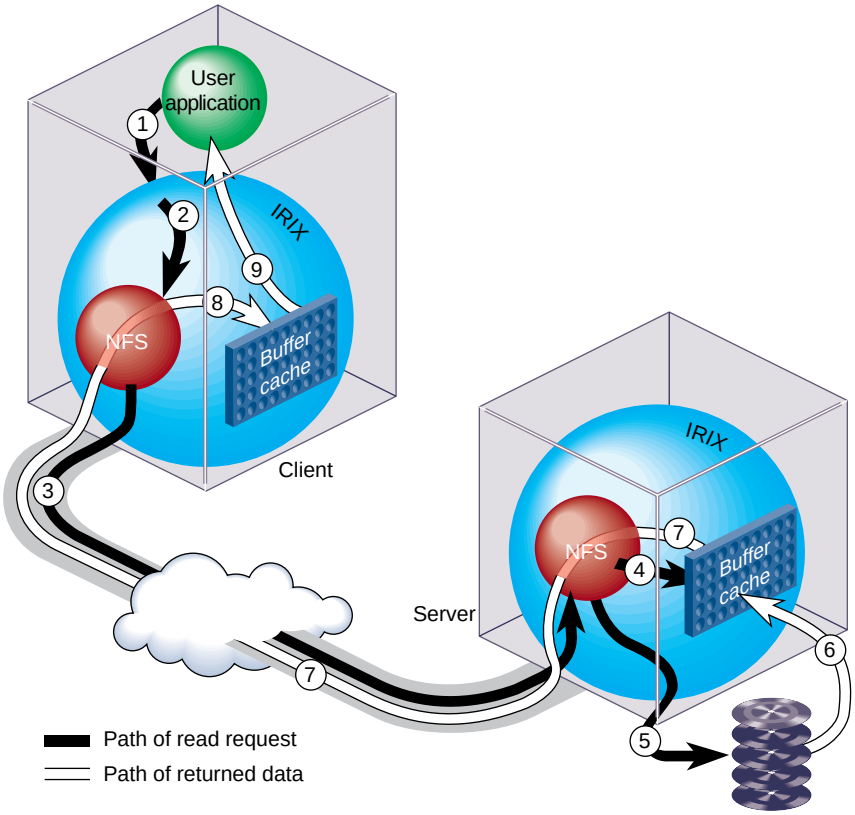

Figure 1-3 Events in a Standard NFS Transaction

These events take place in Figure 1-3:

1. The application issues a read for remote data.
2. The search for the data in the local buffer cache fails, triggering an NFS read.
3. An NFS read is sent to the remote server.
4. The search of the buffer cache on the remote server fails.
5. The server reads from the filesystem on disk.

6. Data is moved to the server's buffer cache.
7. The buffer data is sent to the network.
8. The client receives the data in its buffer cache.
9. The data is sent from the buffer cache to the application.

BDS Transactions

Figure 1-4 illustrates the sequence of events in an unbuffered BDS transaction.

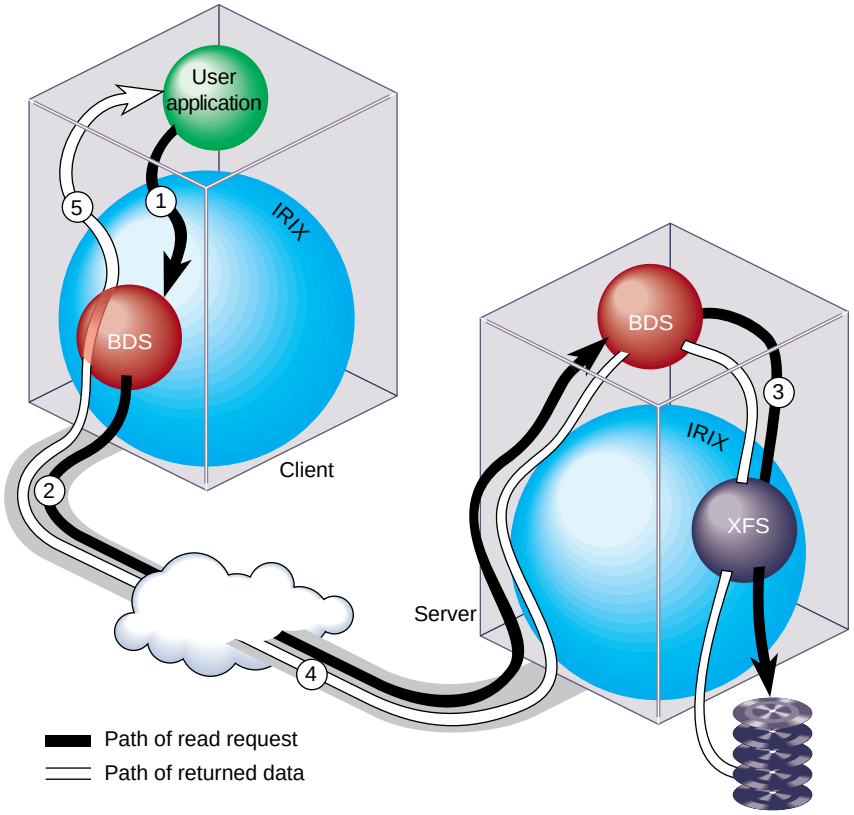

Figure 1-4 Events in a BDSpro Transaction Without Buffering

These events take place in Figure 1-4:

1. The application issues a read for remote data.
2. A BDS read is sent to the remote BDS server.
3. The BDS server reads directly from the filesystem on disk.
4. The BDS server writes the data to the network.
5. Data is mapped to the user's address space (page-flipped) directly from the network.

BDS and Buffering

To increase throughput, BDS performs read buffering automatically. It performs write buffering if explicitly directed to do so (see "Using Write Buffering" on page 21).

The gains derived from buffering are a function of the speed of the network and filesystem in a particular configuration. However, in most BDS implementations, buffering improves performance significantly. For example, in laboratory tests, BDSpro achieved a 40 MB per second throughput rate without buffering. This rate increased to 87 MB per second with read buffering, and with write buffering, performance increased to 93 MB per second (the maximum bandwidth of the HIPPI connection).

When a connection requires buffering, BDS allocates two memory buffers for each open file. The size of these buffers is either calculated by BDS or specified by the user (for details, see "How Buffer Size Is Determined" on page 11 and "Specifying a Buffer Size" in Chapter 3).

BDS performs no buffering under these conditions:

- When it cannot determine a buffer size (for writes only)
- If read requests are not sequential
- If multiple clients are accessing a file from the same host
- When the buffer size is set to zero (see "Disabling Buffering" on page 23)

Read Buffering

BDS performs read-ahead buffering; that is, as it writes data to the network from one buffer, it simultaneously fills a second buffer with data in anticipation of the next request. This concurrent disk and network I/O enhances BDS performance significantly.

Figure 1-5 illustrates read-ahead buffering in a BDS transaction.

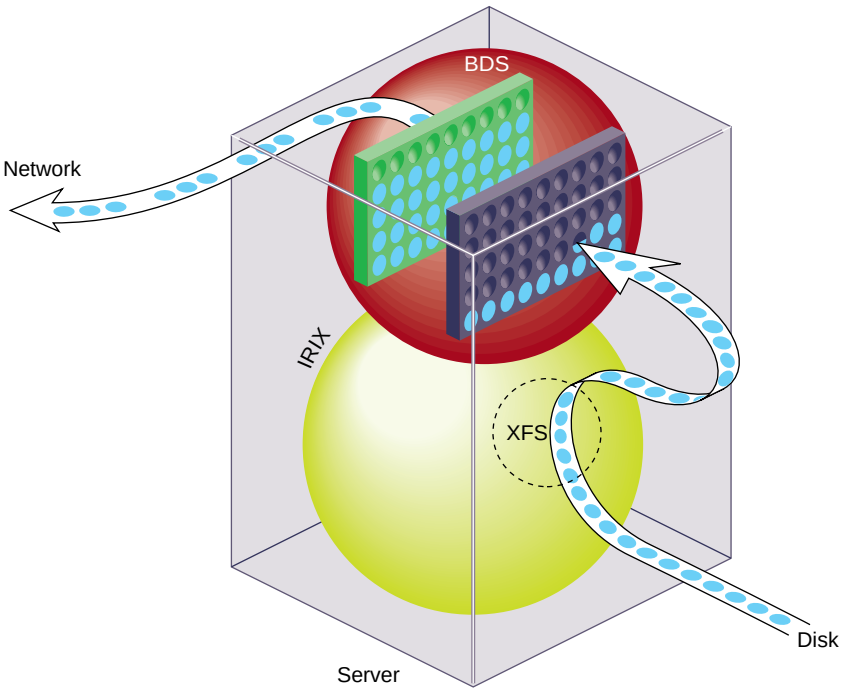

Figure 1-5 Read Buffering in a BDSpro Transaction

As Figure 1-5 illustrates, BDS begins network transfers of read data from a full buffer—no data is transferred to the network until this buffer is full. While the contents of the first buffer is being transferred to the network, the second buffer is being filled from disk in preparation for subsequent read requests.

Write Buffering

BDS normally writes data to disk as the data is received from the network, as shown in Figure 1-4. If you prefer, you can specify write buffering. BDS performs write-behind buffering; that is, it fills a buffer with data before writing any of the data to disk. Because a delay occurs between receiving and writing the data, write buffering poses some risks, and it should be used judiciously (see “Using Write Buffering” on page 21 for details).

Figure 1-6 illustrates how write-behind buffering works.

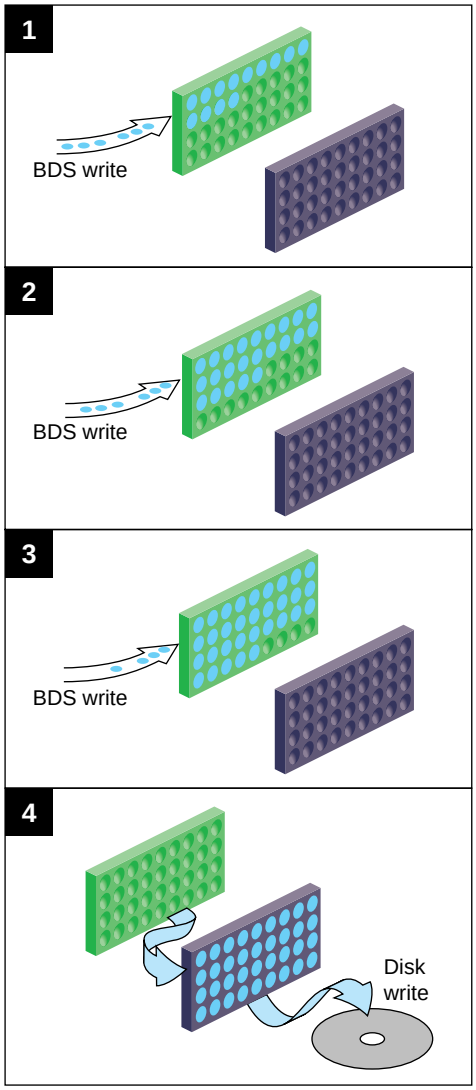

Figure 1-6 Write Buffering in a BDSpro Transaction

How Buffer Size Is Determined

When data is located on an XLV logical volume, BDS calculates a buffer size based on the size of the disk stripe. This is the most efficient buffer size, because it optimizes XLV access and minimizes disk contentions (see “Tuning XLV Performance” in Chapter 2). When the disk is not striped, BDS uses the application’s I/O request size to set the buffer size.

The calculated buffer size is the default, but you can override this default by specifying a buffer size by several methods (see “Specifying a Buffer Size” in Chapter 3 for details). The buffer size setting that is in effect applies to both read and write buffering.

When BDS Makes Sense

While BDS offers clear advantages to standard NFS implementations in some operating environments, Silicon Graphics recommends it over standard NFS only in certain circumstances. Real network throughput rates, the applications running on a network, and the size of files involved in network operations determine whether BDS is a desirable addition to your current NFS implementation.

For transactions in which the read or write request size is 128 KB or larger, BDS is a sound alternative to standard NFS because it offers much faster performance (see Table 1-1 for speeds achieved with BDSpro). Furthermore, performance improves as the request size increases—BDS achieves optimum performance when the read and write request size is the same as the BDS buffer size (see “Specifying a Buffer Size” on page 22 for details).

You should consider adding BDS if your NFS implementation meets these criteria:

- Your applications use large read and write request sizes (greater than 128 KB) in requests to remote filesystems.
- Network hardware is a high-speed medium (such as HIPPI or fiberchannel) with a potential transfer rate of 100 MB per second or higher.
- The applications that you use do not rely on data caching.

Chapter 2

Preparing for BDSpro

This chapter explains what BDSpro requires to achieve its full potential and how to modify your current setup if you determine that changes are needed. The chapter contains these sections to help you prepare for running BDSpro:

- “Installation Prerequisites”
- “Disk and Controller Configuration Requirements”
- “Changing the Maximum DMA Size”
- “Measuring XFS Rates”
- “Tuning XLV Performance”
- “Sample Performance Results”

Installation Prerequisites

BDSpro requires IRIX version 6.2 (or higher), BDSpro software, and NFS (version 2.0 or 3.0) installed on client and server systems. Server and client systems must also contain current NFS rollup patches and the *mount* command patch; server systems also require the current IRIX and networking rollout patches. Server and client systems must be connected to a HIPPI, fiberchannel, or other high-performance network. (See the *BDSpro Release Notes* for detailed requirements and instructions on software installation).

Disk and Controller Configuration Requirements

When selecting disks for use with BDS, choose a brand and model that excel in large sequential access operations. Disks with this characteristic offer better BDS performance and are therefore a better choice.

BDS installations that use fast-and-wide SCSI controllers with speeds of 20 MB per second (not UltraSCSI or Fiber Channel) must optimize the number of disks on a single

SCSI bus. For example, a system containing four IBM drives with a transfer rate of 5 MB per second completely saturated a controller with a 20 MB per second transfer rate. The IBM drives provided by SGI spin at about 7 MB per second on the outer zone and 5 MB per second on the inner zone. Assuming a transfer rate of 5 MB per second, it is apparent that 4 drives will completely saturate a single 20 MB per second controller.

To optimize a factory-shipped SCSI box for BDSpro, reconfigure it from a single-channel to a two-channel device. Silicon Graphics provides SCSI boxes with an eight-drive capacity that are configured for one SCSI channel. This single channel configuration is inefficient for BDS— one channel can service only four of the disks before the maximum bandwidth is reached (4 disks @ 5 MB / disk / second = 20 MB / second). However, by configuring the SCSI box with two channels, its bandwidth is doubled and the additional channel can service the remaining four disks.

For maximum sequential performance with the minimum number of disks, purchase more controllers and use one controller for every three or four disks.

Changing the Maximum DMA Size

IRIX imposes a limit on the maximum size of direct memory access (DMA) operations. This limit affects XFS, since direct I / O is a DMA operation. In IRIX 6.2, the default maximum DMA size is 4 MB. Frequently, this limit must be increased on BDSpro servers to achieve optimum performance.

To change the maximum DMA size, reset the *maxdmasz* variable using *systune* (see the systune(1M) reference page).

The values of *maxdmasz* are expressed in pages, which are 16 KB on 64-bit systems. Change these values to the size that you need, and then reconfigure and reboot the server.

Measuring XFS Rates

BDSpro performance is highly dependent on the local performance of XFS and XLV on the server system; BDS is superfluous when local filesystem speed (or network speed) creates performance bottlenecks. You can often correct filesystem performance by properly configuring disks and by setting the correct size for direct memory access operations.

To measure local XFS performance, use *lmdd* commands similar to those shown below. If you determine that the results are inadequate, follow the tuning recommendations in "Tuning XLV Performance," which follows.

This command creates *testfile*, a 500 MB file with a transfer size of 7 MB; the return message confirms the new file:

```
server# lmdd of=/export/bds/testfile bs=7m move=500m direct=1
497.00 MB in 5.83 secs, 85.29 MB/sec
```

This command performs a direct read on *testfile* with a transfer size of 7 MB; the return message shows the XFS transfer rate:

```
server# lmdd if=/export/bds/testfile bs=7m move=500m direct=1
497.00 MB in 4.44 secs, 111.92 MB/sec
```

Tuning XLV Performance

XFS uses a logical volume manager, *xlv*, to stripe data across multiple disk drives. On striped disks, large XFS requests are split and sent to each disk in parallel. The high sequential performance of XFS is attributable to this parallelism. (See Chapter 7 of *IRIX Admin: Disks and Filesystems* for more information.)

The size of data transfers is an important consideration in planning logical volumes. For example, assume that a logical volume contains ten disks and the stripe size is 64 KB. In this case, transfers of 640 KB or larger are required to get all drives running simultaneously. If the data transfer size is 320 KB, only five drives are active in an I/O operation. Because only half of the available disks are used, a transfer size of 320 KB is very inefficient, reducing the total performance by half. With proper striping of logical volumes, however, you can maximize disk performance.

Disk Striping Fundamentals

The *xlv_make* utility is used to stripe the disks in a logical volume. By default, *xlv_make* divides the disk into tracks and uses one track from each disk in rotation to create a stripe. The amount of data that *xlv_make* allocates on a single drive before going to the next is called the *stripe unit*. The stripe unit and the number of disks in the logical volume determine the *stripe width*, or

stripe width = stripe unit × number of disks

Figure 2-1 illustrates a logical volume containing four disks. Notice from this figure that the stripe unit is set to two tracks instead of one (the default stripe unit size). If we assume a track size of 100 KB (track size is set by disk manufacturers), the stripe width for this logical volume is 800 KB.

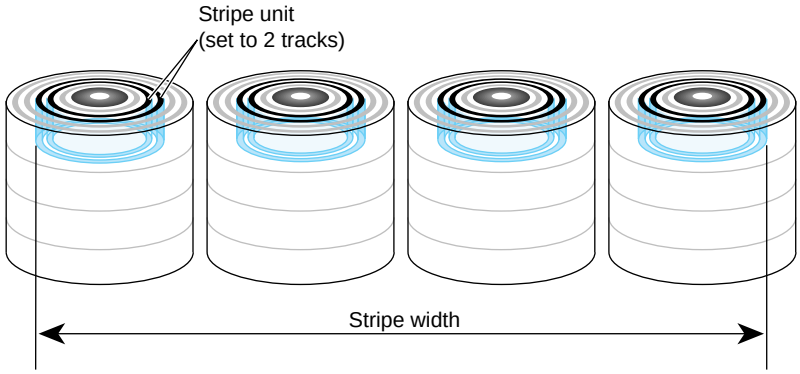

Figure 2-1 Effects of the Stripe Unit and Disk Number on Stripe Width

Determining the Size of Stripe Units

When you create a logical volume, you can specify a stripe size using the *stripe_unit* argument of *xlv_make* (see the xlv_make(1M) reference page). Specifying the proper size of the stripe unit is the key to optimizing I/O performance. In most cases, the objective in setting the stripe size is to achieve a particular bandwidth; but you might also need to adjust the stripe size to accommodate an application that uses a fixed transfer size.

The transfer size should be a multiple of the system's page size. The transfer size should also be a multiple of the stripe width (800 KB in Figure 2-1). For example, if an application needs the bandwidth of all four disks but is reading with a transfer size of 400 KB, you could set the stripe unit to one track instead of two to achieve the required bandwidth with half the transfer size.

Optimizing the Stripe Unit Size

It is not always advisable to use the smallest possible stripe unit. While small requests can be effective with read transfers because of the read-ahead assistance that SCSI track buffers offer, small stripe units degrade write performance.

For example, consider what happens when data is written using the default stripe unit size of one track. The write is broken into tracks and each track is sent to a different disk. When the data arrives at the controller, the controller first waits for the disk head to move to the beginning of the track before it writes the data. This wait, commonly referred to as a *rotational delay*, occurs between each track that is written to the same disk; as a result, using a one-track stripe unit reduces the write performance to half of the read performance.

It is possible to achieve higher write performance by using larger stripe units. Table 2-1 shows the effects of increasing the size of stripe units on XFS write performance.

Table 2-1 Effects of Stripe Unit Size on XFS Write Performance

Stripe Unit	Request Size	Write Performance
1 track = 100 KB[a]	1 track × 4 disks = 400 KB	1/2 read performance
2 tracks = 200 KB	2 tracks × 4 disk = 800 KB	2/3 read performance
3 tracks = 300 KB	3 tracks × 4 disks = 1.2 MB	3/4 read performance
4 tracks = 400 KB	4 tracks × 4 disks = 1.6 MB	4/5 read performance

a. Default size used by the *xlv_make* command.

Sample Performance Results

Table 2-2 shows the performance for BDSpro (version 1.0) using IBM drives with a 2 GB capacity and a HIPPI network. Three disks were configured on each controller; the transfer size was set to the stripe width size. Notice from Table 2-2 that BDS writes are significantly slower than XFS writes when write buffering is not used.

Table 2-2 Performance Results With Sample Configurations [a]

Disks	Unit	Width	XFS Read	XFS Write	BDS Read	BDS Write	BDS Buffered Write
7	256 KB	1792 KB	52	40	52	15	31
14	128 KB	1792 KB	79	43	69	28	42
14	256 KB	3584 KB	83	50	73	31	48
14	512 KB	7168 KB	84	53	74	33	53
36	256 KB	9216 KB	196	120	89	51	94
68	60 KB	4080 KB	189	121	81	51	92
68	120 KB	8160 KB	221	163	79	58	92

a. Read and write speed is expressed in MB per second.

Chapter 3

Setting Up and Testing BDSpro

To add BDSpro to an NFS implementation, you mount the filesystems that BDS is to use, and then test BDSpro performance on the filesystems. This chapter contains the following sections that explain setup and testing procedures:

- “Mounting Filesystems for BDSpro”
- “Using BDS from an Application”
- “Using Write Buffering”
- “Specifying a Buffer Size”
- “Starting and Stopping BDSpro”
- “Testing a BDSpro Setup”
- “Testing the Network”

Note: Be sure to review the information in Chapter 2, “Preparing for BDSpro,”and follow the recommendations that it contains before proceeding with BDSpro setup.

Mounting Filesystems for BDSpro

To mount filesystems on BDSpro clients, use the standard NFS *mount* command and the **-o bds** option. BDS services are not invoked unless this option appears on the *mount* command line. (See the fstab(4) reference page for complete information on the mount command.)

This example illustrates a BDS entry in a client */etc/fstab* file:

```
hip0-goliath:/ /bdsmnt -o bds, vers=3, rw 0 0
```

In the previous example, the root filesystem from server *goliath* is mounted to */bdsmnt* on the client. The interface hip0-goliath on the server connects to the HIPPI network where the client is also attached.

Exporting Filesystems for BDSpro

To make filesystems available to BDSpro, export them from the server with the standard NFS *exportfs* command. No special arguments to *exportfs* are required, and all standard *exportfs* arguments are valid (see the exportfs(1M) reference page for details).

Note: Only XFS type filesystems are supported with BDSpro.

Automatic BDS Mounting

You can add the **bdsauto** option to a *mount* command line so that BDS is used automatically whenever the transfer size exceeds a specified limit. When **bdsauto** is set, standard NFS is used for file I/O unless the transfer size that you specify is exceeded, in which case BDS is used.

This example illustrates a mount command that sets **bdsauto**:

hip0-goliath:/ /bdsmnt -o bds, bdsauto=2000000, vers=3, rw 0 0

The entry in the previous example sets **bdsauto** to two megabytes, so BDS will be used in file access operations if the transfer size is two megabytes or larger, even if the application is not modified to take advantage of BDS. NFS will be used if the transfer size is smaller than two megabytes.

Using BDS from an Application

In addition to the **bdsauto** option, you can specify BDS services from an application. To use BDS, the application must open files with either the **O_DIRECT** or, on systems running IRIX 6.5,the **O_BULK** argument (see **O_DIRECT** and **O_BULK** descriptions in the open(2)) reference page for details). After the file is opened, these arguments may be changed with the file control function (see the **FDIRECT** and **FBULK** options of the fcntl(2)) reference page for details).

The **O_DIRECT** and **O_BULK** arguments have significant differences:

- The **O_BULK** argument turns on write buffering, which may not be desireable for some applications (see “Using Write Buffering” on page 21)
- An open using **O_DIRECT** fails ifBDS is not available. Whereas if you use the **O_BULK** argument and BDS is not available, the open succeeds and NFS is used for the transaction instead of BDS.

Whenever direct I/O and bulk I/O are used in the same application, bulk I/O takes precedence. If write buffering is in effect and an application must be certain that data is written to disk, the application should use the *fsync* system call (see the fsync(2) reference page for details). BDS performance is significantly reduced when you use *fsync*.

Using Write Buffering

To turn on write buffering, use the **-writebehind** option in the command to start the BDS server (see the bds(1M) reference page). On systems running IRIX 6.5 or higher, you can also specify write buffering from an application when the application opens a file (see “Using BDS from an Application” on page 20 for details).

Consider these risks before using write buffering:

- A server failure can result in the loss of data if the failure occurs before the buffer is written to disk.
- Write buffering delays error reporting, since an error is reported only after a complete buffer is transferred to disk.
- Errors might be reported to a different application accessing the same file on the same client.

If write buffering is in effect and an application must be certain that data is written to disk, the application should use the *fsync* system call (see the fsync(2) reference page for details). However, using *fsync* compromises the benefits of write buffering.

Specifying a Buffer Size

By default, BDS calculates the size of its read and write buffers based on the size of the logical volume stripe (see "Tuning XLV Performance" in Chapter 2). If the filesystem is not striped, BDS sets the buffer size to the application's read size.

You can also specify a buffer size by these methods:

- Use the **-buffersize** argument in the command to start the bds server (see the bds(1M) reference page for details).
- Include the **bdsbuffer=** option on the *mount* command line.

Precedence in Setting Buffer Size

Buffer size specifications have this order of precedence:

1. The **buffersize** argument on the server
2. The **bdsbuffer** option in the mount command on the client.
3. The BDS calculated size based on disk stripe size.
4. The BDS calculated size based on the application read size.

This example sets the buffer size using a mount option on the client:

hip0-goliath:/ /bdsmnt -o bds, bdsauto=2000000, bdsbuffer=4194304, vers=3, rw 0 0

The entry in the previous example sets the buffer size to 4 MB. Because this parameter is set when the *bdsmount* filesystem is mounted, BDS will not use a calculated buffer size for *bdsmount* I/O; it will allocate buffers of 4 MB.

However, assume that this command was used to start BDS on the server:

bds **-buffersize 2097152**

In this case, BDS uses a buffer size of 2 MB, overriding the 4 MB specified in the filesystem mount on the client.

Recommendations for Setting Buffer Sizes

The amount of memory used in buffering is affected by the number of users accessing the server simultaneously and buffer size. If you find that default or user-specified buffer sizes are requiring too much memory, you can specify a smaller buffer using any size specification method (see “Specifying a Buffer Size” on page 22).

Disabling Buffering

You can disable BDS buffering by specifying a buffer size of zero (0). If you disable buffering, you can expect significant performance reductions. For example, laboratory tests with read-ahead buffering demonstrate a sustained performance of 87 MB per second, whereas without bufferring, performance measured about 54 MB per second; with write-behind buffering, speeds of 93 MB per second dropped to 45 MB per second without buffering.

Starting and Stopping BDSpro

BDSpro software is ready to run after it is installed. To start and stop BDS, use these command on the server:

/etc/init.d/BDSpro start
/etc/init.d/BDSpro stop

This *start* command starts BDS automatically during the server startup sequence. The command reads a file called */etc/config/BDSpro.options*, which you can create to specify BDS options, such as write buffering and buffer sizes.

To enable or disable the automatic starting of BDSpro during the server startup, use these commands:

chkconfig BDSpro on
chkconfig BDSpro off

Testing a BDSpro Setup

To test a BDSpro setup, run the BDSpro server in false disk mode. False disk mode is analogous to sending the data to */dev/null* or receiving it from */dev/zero*. In false disk mode, the BDSpro server simulates data movement to and from the disk; network data is unaffected. This mode is used to verify network performance.

Use this command to start the BDSpro server in false disk mode:

```
server# /usr/etc/bds -devnull -devzero -touch -log
```

The *-touch* option tells BDSpro to touch all data before sending it, which simulates XFS overhead. (See the bds(1M) reference page for a description of all options).

On the client, first mount the server:

```
client# mkdir /mnt
client# mount -o bds server:/ /mnt
```

Note: The mount will fail if you have not yet upgraded the client kernel to use BDSpro, but you can still test BDSpro with *lmdd*, since this command contains a user level implementation of the XBDS protocol (see "Debugging Without Kernel-Level BDSpro," later in this chapter).

After the filesystem is mounted, try reading a file using a command similar to this (remember that no data is actually read in false disk mode):

```
client# lmdd if=/mnt/unix direct=1 bs=1m move=20m
20.97 MB in .43 secs, 48.31 MB/sec
```

If you enter two *lmdd* commands in succession, performance improves on the second read. This results from the overhead that BDS incurs on its first sever access; this overhead is not incurred in the second read:

```
client# lmdd if=/mnt/unix direct=1 bs=1m move=20m
20.97 MB in .43 secs, 48.31 MB/sec

client# lmdd if=/mnt/unix direct=1 bs=1m move=20m
20.97 MB in .29 secs, 73.34 MB/sec
```

Using the BDSpro Debugger

If BDSpro performance is not what you expected, you can use verbose debugging by adding the **-debug** option to the *bds* command line. You can debug in either false or real disk mode. When you use real data, debugging prints timing data for the network transfer and the filesystem transfer. In false disk mode, only network timing is displayed.

Use the following command on the client to generate debugging on the server:

```
client# lmdd of=debugfile bs=4m move=12m direct=1
```

The previous *lmdd* command writes data across the network to the BDS server using a block size of 4MB. The BDS transfer size is whatever the remote client is using for read or write requests.

The example that follows shows the bds debugging output on the server. Because data is read from the network and written to the disk, timing results on reads (`readn`) are network times and timing results on writes (`write`) are XFS times.

```
server# bds -debug
```

```
readn: 7fff2e18 72.0000  @ 3.0000 /sec
V3 filehandle: F xid=2878 uid=0 gid=0 fhandle len=68, buflen=16.0000E,
oflags=9
Want file handle 68 bytes
readn: 10619dd8 68.0000  @ 0.8314M/sec
setting bds_xfs_align to 4095
V3 filehandle: setting rbuflen to 7.9688M, wbuflen to 0
bdsid 7312, sprocid 7320
V3 filehandle write(4, 7fff2e18, 72) = 72
writen: V3 filehandle: 72.0000  @ 94.6328K/sec
V3 filehandle: A xid=2878 uid=0 gid=0 bytes=24.0000
V3 filehandle write(4, 7fff2b50, 24) = 24
writen: V3 filehandle: 24.0000  @ 89.0000 /sec
readn: 7fff2e18 72.0000  @ 0.8917M/sec
V3 filehandle: W xid=2877 uid=0 gid=0 off=0x0 len=4.0000M
getbuf returning buffer 0
readn: 441c000 4.0000M @ 84.6113M/sec
write(direct): sz 4.0000M off 0 @ 82.1743M/sec
V3 filehandle write(4, 7fff2e18, 72) = 72
writen: V3 filehandle: 72.0000  @ 269.00 /sec
V3 filehandle: A xid=2877 uid=0 gid=0 bytes=4.0000M
freeing buffer 0
readn: 7fff2e18 72.0000  @ 0.8803M/sec
V3 filehandle: W xid=2879 uid=0 gid=0 off=0x400000 len=4.0000M
```

```
getbuf returning buffer 1
readn: 4018000 4.0000M @ 109.40M/sec
write(direct): sz 4.0000M off 4.0000M @ 103.86M/sec
V3 filehandle write(4, 7fff2e18, 72) = 72
writen: V3 filehandle: 72.0000  @ 90.9600K/sec
V3 filehandle: A xid=2879 uid=0 gid=0 bytes=4.0000M
freeing buffer 1
readn: 7fff2e18 72.0000  @ 0.9537M/sec
V3 filehandle: W xid=2880 uid=0 gid=0 off=0x800000 len=4.0000M
getbuf returning buffer 0
readn: 441c000 4.0000M @ 108.11M/sec
write(direct): sz 4.0000M off 8.0000M @ 105.79M/sec
V3 filehandle write(4, 7fff2e18, 72) = 72
writen: V3 filehandle: 72.0000  @ 88.1104K/sec
V3 filehandle: A xid=2880 uid=0 gid=0 bytes=4.0000M
freeing buffer 0
readn: 7fff2e18 72.0000  @ 0.8273M/sec
V3 filehandle: C xid=2881 uid=0 gid=0
V3 filehandle write(4, 7fff2e18, 72) = 72
writen: V3 filehandle: 72.0000  @ 86.8047K/sec
V3 filehandle: A xid=2881 uid=0 gid=0 bytes=0
hip0-ebony.engr.sgi.com moved 12.58 MB in 1.58 secs, 7.96 MB/sec
```

Debugging Without Kernel-Level BDSpro

The *lmdd* debugging tool included with BDSpro has a user level implementation of the BDS protocol. *lmdd* tries to use the kernel level BDS protocol, but if that is not present (or not enabled), *lmdd* uses the user level protocol.

If you need to debug on the client system, you can do so by mounting the filesystem without the **-o bds** option. In this case, the filesystem is mounted with kernel level BDS disabled.

To see local debugging output, use *lmdd* with the **debug=1** option, as shown in this example:

```
client# lmdd if=/bds/test direct=1 bs=4m move=12m debug=1
OS did not know O_DIRECT (8000) on /bds/test
0.003 bds_open(3, /bds/test, 32768, 3)
restart(/bds/test)
open(hip0-mahogany:/export/bds1/test) = 3
opened /bds/test
read at 0.000, rwnd=0 swnd=0
```

```
0.001: R xid=2 uid=0 gid=0 off=0x0 len=4.0000M
0.124: A xid=2 uid=0 gid=0 bytes=4.0000M
/bds/test moved=4194304 wanted=4194304 seekp=4194304 @ 22.8867M/sec
read at 0.176, rwnd=0 swnd=0
0.176: R xid=3 uid=0 gid=0 off=0x400000 len=4.0000M
0.300: A xid=3 uid=0 gid=0 bytes=4.0000M
/bds/test moved=4194304 wanted=4194304 seekp=8388608 @ 23.3921M/sec
read at 0.348, rwnd=0 swnd=0
0.348: R xid=4 uid=0 gid=0 off=0x800000 len=4.0000M
0.373: A xid=4 uid=0 gid=0 bytes=4.0000M
/bds/test moved=4194304 wanted=4194304 seekp=12582912 @ 21.4247M/sec
12.58 MB in 0.54 secs, 23.49 MB/sec
```

Testing the Network

BDSpro is typically used on HIPPI because of its high performance (consult the *IRIS HIPPI Administrator's Guide* for information on installing and configuring a HIPPI network). If BDSpro testing shows inadequate performance, network problems might be the cause. In this case, you can use the *ttcp* test program to verify that the network is functioning properly. *ttcp* is a client/server program that moves data between systems and reports performance results.

To use *ttcp*, enter this command on the client to start the test:

```
client% ttcp -s -r -l 524288
ttcp-r : buflen=524288, nbuf=2048,align=16384/0, port=5001 tcp
ttcp-r : socket
```

Then, enter the following command on the server to start *ttcp* and send data to the client. The output of *ttcp* shows transfer rates:

```
server% ttcp -s -t -T -l 524288 -n 200 hip-client
ttcp-t: buflen=524288,nbuf=200, align=16384/0, port=5001 tcp -> hip-client
ttcp-t: socket
ttcp-t: connect
ttcp-t: 104857600 bytes in 1.42 real seconds = 73843.38 KB/sec +++
```

The server should be sending data at approximately 65 to 70 MB per second. If you omit the **-T** option, which touches the data to measure caching effects, the rate should be approximately 90 MB per second. If the network is not performing at the expected level, determine the cause of the problem and correct it. The problem may be caused by one of these conditions:

- The data is not being transferred on the HIPPI network.

  By default, IRIX designates the Ethernet interface as the primary network interface and assigns the hostname (the name in the */etc/sys_id* file) to this interface. To assign a hostname to the HIPPI interface, IRIX appends the prefix *hippi* to the hostname. (For example, if the Ethernet interface is named *frosty*, the HIPPI interface is named *hippi-frosty*).

  Check the hostname of the HIPPI client that you specified in the *ttcp* command to verify that it is the HIPPI hostname (and not the Ethernet hostname) for this client. Remember to specify a HIPPI interface when mounting the filesystem also (see "Mounting Filesystems for BDSpro," earlier in this chapter).

- The server is running a debugging or sema metering kernel.

  Reboot with a non-debug kernel, which performs much faster.

- The client or server is not running IRIX 6.2 or later.

  IRIX 5.3 (or earlier) does not offer the HIPPI performance that BDSpro requires. Upgrade to IRIX 6.2 or later, which support BDSpro.

- The connection is passing through a router.

  Use *netstat -i* to determine whether the HIPPI interface on the server connects to the HIPPI network where the client resides. If the client and server are not on the same network, a high-performance router will be required to support HIPPI speeds.

If you try all of these measures and performance is still not adequate, contact your Silicon Graphics support provider for additional assistance.

Appendix A

Frequently Asked Questions

The questions and answers listed in this appendix can help with troubleshooting BDSpro.

1. Where is BDS server software stored?

   The BDS server is now located in */usr/etc* instead of */usr/sbin*. This is more consistent with other IRIX daemons.

2. How do I change the configuration options passed to the server?

   The file */etc/config/BDSpro.config* is passed to the BDS server as arguments. By default, it turns on logging and sends the results to */var/adm/bds.log*.

3. BDS is using a lot of memory. How can I limit it?

   Now that BDS uses buffering to speed up file accesses, you may find that buffering can take up a lot of memory, particularly if you have many different applications accessing files. See "Specifying a Buffer Size" in Chapter 3.

4. Why am I not seeing big improvements in write performance?

   The improvements are a result of enabling write buffering. Write buffering is not enabled by default because it has the potential to lose data in the event of a server failure, and because, if a write error occurs, it is not reported until a subsequent I/O operation. See "Using Write Buffering" in Chapter 3 for more information.

5. Why is *mount* locking up?

   As of BDSpro 2.0, you cannot mount a filesystem using BDS unless a BDS daemon running on the server. Just start up BDS on the server and the problem should be solved.

6. I think my BDS server is dead or misbehaving. How do I restart it?

   A script in */etc/init.d* starts and stops the BDS daemon. As root, type this command:

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

   This stops all BDS daemons that were running and restarts them. Clients accessing BDS are automatically reconnected to the new BDS daemons.

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-3274-002.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-965-0964
- To send your comments by **traditional mail**, use this address:

  Technical Publications
  Silicon Graphics, Inc.
  2011 North Shoreline Boulevard, M/S 535
  Mountain View, California 94043-1389