Getting Started With BDSpro

Document Number 007-3274-001

CONTRIBUTORS

Written by Pam Sogard
Illustrated by Dany Galgani
Edited by Christina Cary
Production by Lorrie Williams
Engineering contributions by Larry McVoy
Cover design and illustration by Rob Aguilar, Rikk Carey, Dean Hodgkinson,
Erik Lindholm, and Kay Maitz

Getting Started With BDSpro
Document Number 007-3274-001

About This Guide

Getting Started With BDSpro is written for server and site administrators who plan to add BDSpro™, the Silicon Graphics© implementation of the Bulk Data Service protocol, to a Network File System (NFS®) implementation. This guide contains information to help administrators understand the BDS protocol (XBDS™), prepare a site for BDSpro, and include BDS extensions to an existing NFS implementation. To use this guide successfully, you should be experienced in NFS administration and in managing XFS™ on large servers.

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_make(1M), fcntl(2), open(2), read(2), write(2), filesystems(4), and malloc(3C).

Contents

List of Figures

List of Tables

Chapter 1

BDS Fundamentals

Bulk Data Service (BDS) is a non-standard extension to NFS that handles large file transactions over high-speed networks. BDS exploits the data access speed of the XFS filesystem and data transfer rates of network media, such as HIPPI and fiberchannel, to accelerate standard NFS performance. The BDS protocol, XBDS, modifies NFS functions to reduce the time needed to transfer files of 100 megabytes or larger over a network connection. BDSpro is the Silicon Graphics implementation of XBDS.

Figure 1-1 illustrates the XBDS protocol relative to existing ONC protocols.

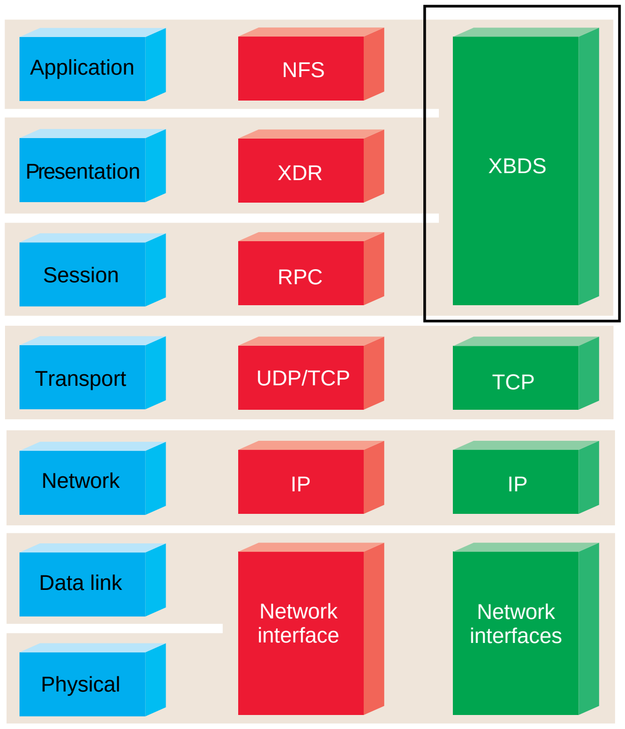

Figure 1-1 XBDS Protocol Compared with ONC Protocols

You can use BDSpro on Silicon Graphics systems running IRIX 6.2 (or later). Hosts must be connected to a high-speed network (such as HIPPI or fiberchannel) running the transmission control protocol / internet protocol suite (TCP / IP).

Note: A Linux version of BDS that runs as a user-level application is available for nonIRIX systems. If you would like to implement BDS on other platforms, contact your Silicon Graphics sales representative for information on BDS for multi-vendor networks.

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 on Silicon Graphics systems.

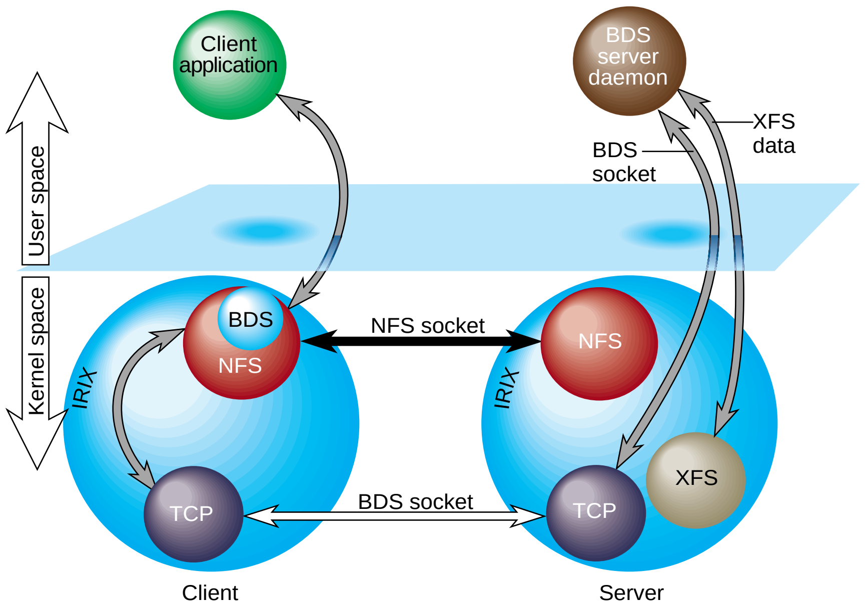

Figure 1-2 The BDSpro Client-Server Model

The hardware and software used on a network and its loading patterns determine the ultimate speed of NFS transactions. Because these factors vary greatly on individual networks, it is impossible to predict the performance gains that BDS will provide for a particular network. However, to evaluate BDS performance potential, it is useful to consider BDSpro comparisons to standard NFS under constant network conditions.

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

Table 1-1 BDSpro Performance Compared With Standard NFS[a]

Product	Network Configuration	Read Rate
NFS (version 2)	UDP over HIPPI	2.5 MB per second per channel
NFS (version 3)	UDP over HIPPI	16 MB per second per channel
BDSpro	TCP/IP over HIPPI	70 MB per second per channel

a. Rates are based on NFS read and write sizes of 48 KB on files of 100 MB or larger. MB is defined as 1024×1024 bytes.

How BDS Works

To achieve high throughput, BDS relies on the ability of the operating system to perform *direct input* and *output* operations (see the **O_DIRECT** option of the open(2) and fcntl(2) IRIX reference pages for details). In direct I/O operations, 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, the time saved by bypassing the buffer cache is doubled, since the bypass occurs on both the client and the server systems (see Figure 1-4).

Direct I/O is based on the assumption that buffer alignment and size are known constants. For this reason, the network applications that use BDS must perform page-aligned I/O. Applications that are tuned for XFS performance are ideal for use with BDS, since XFS also requires this alignment.

Standard NFS Transactions

Figure 1-3 illustrates the sequence of 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 local buffer cache fails.

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 buffer cache on the server.
7. The buffer data is sent to the network.
8. The client receives the data in buffer cache.
9. The data is sent from the buffer cache to the application.

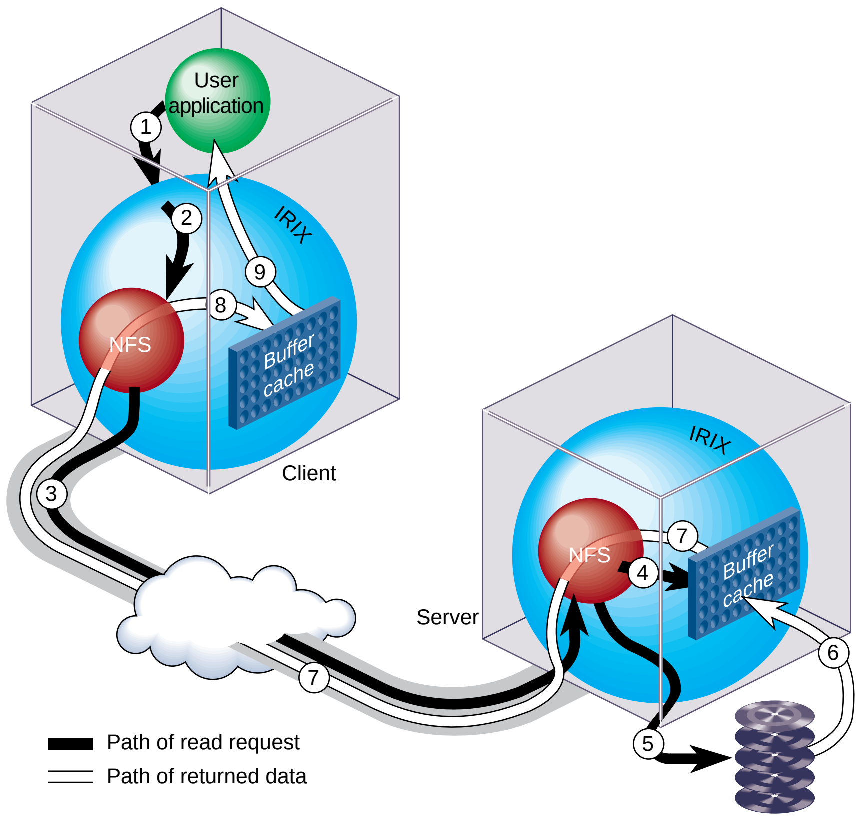

Figure 1-3 Events in a Standard NFS Transaction

BDS Transactions

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

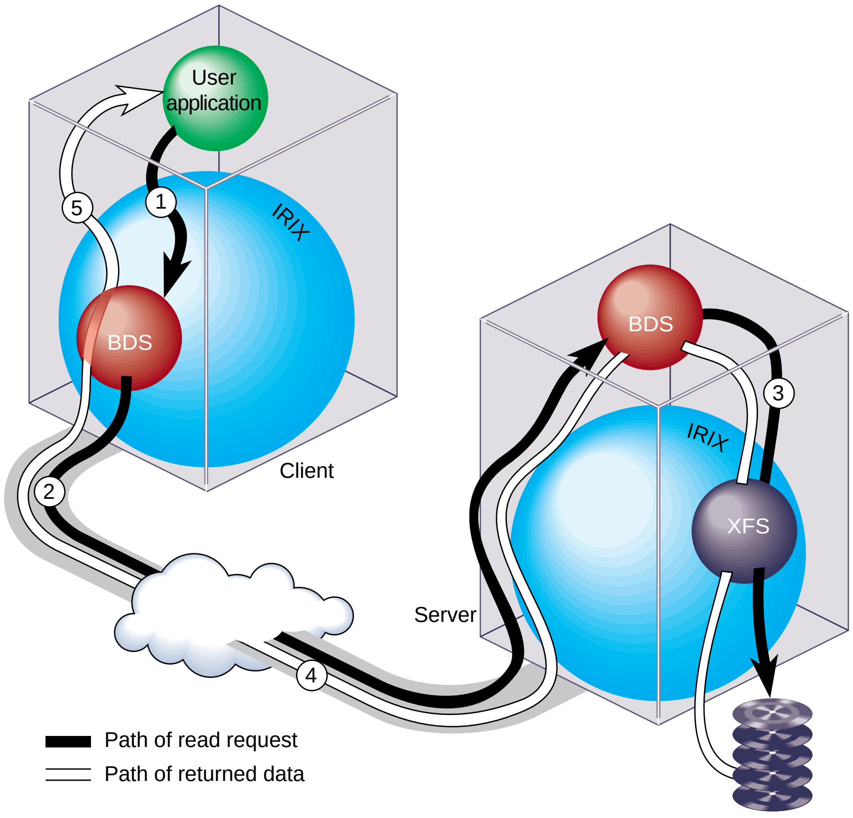

Figure 1-4 Events in a BDSpro Transaction

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 performs an aligned write to the network.
5. The data is page-flipped directly from the client’s network buffer to the application.

When BDS Makes Sense

While BDS offers clear advantages to standard NFS implementations in some operating environments, it is recommended 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.

File size is an especially important consideration in evaluating BDS for your site. For small and medium size files, standard NFS frequently offers satisfactory performance. NFS speed with files of this size is attributable to local data caching, which it performs to reduce (or eliminate) the need for time-consuming reads and writes over the network.

In transactions with large files, however, NFS caching is troublesome because it purges potentially valuable data from the cache to store the large file. With BDS, this problem is eliminated because BDS performs no data caching (see "How BDS Works" for details). Because of the cache bypasses (and for other performance reasons), BDS also offers much faster transfer speeds (see Table 1-1 for speeds achieved with BDSpro).

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

- Files of 100 MB or larger are routinely accessed over network connections.
- 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 perform page-aligned input and output, and they perform direct reads and writes when accessing XFS files.
- The applications that you use do not rely on data caching.

Chapter 2

Preparing for BDSpro

This chapter explains what BDSpro requires if it is 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"
- "Planning Disk and Controller Configurations"
- "Tuning XFS Performance"
- "Sample Performance Results"

Installation Prerequisites

BDSpro requires that IRIX version 6.2 (or higher) and NFS (version 2.0 or 3.0) are installed on client and server systems. It also requires that the applications that use BDSpro services perform page-aligned I/O and direct reads and writes (see "How BDS Works" in Chapter 1).

BDSpro software must be installed on both client and server systems. See the *BDSpro Release Notes* for instructions on software installation.

Planning Disk and Controller Configurations

Silicon Graphics currently ships SCSI disks manufactured by Quantum and IBM. Quantum disks excel in random access and relatively small I/O operations, and IBM disks excel in large sequential access operations. Because of the operating characteristics of BDS, disks that excel in large sequential access operations offer better performance and are therefore a better choice.

Most BDS installations are likely to use fast-and-wide SCSI controllers with speeds of 20 MB per second. 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. If we assume a transfer rate of 5 MB per second, it is apparent that 4 drives will completely saturate a single 20 MB per second controller.

Silicon Graphics provides SCSI boxes with an eight-drive capacity that are configured for one SCSI channel. Since channel speed is 20 MB per second and drive speed is approximately 5 MB per second, eight drives offer more bandwidth than is needed. Therefore it is possible to configure SCSI boxes with two SCSI channels per box, doubling the bandwidth for each box.

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

Tuning XFS Performance

BDSpro performance is highly dependent on the local performance of XFS on the server system. In general, BDSpro requires a local XFS access speed that is twice as high as the speed at which BDSpro is to perform. For example, if BDSpro is to operate at a rate of 50 MB per second, the server XFS rate must be at least 100 MB per second. This XFS rate is required to offset the overhead that BDSpro protocols impose on network transactions.

To measure local XFS performance, use *lmdd* commands similar to those shown below.

This command creates *testfile*, a 500 MB file with a blocksize of 7 MB:

```
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 7 MB; the output 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
```

Inadequate XFS performance can frequently be corrected by properly configuring disks and by setting the correct size for direct memory access operations. These topics are explained in the sections that follow.

Configuring Disks for BDSpro

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 10 disks and the track 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, disk performance can be maximized.

Understanding Logical Volume Stripping

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.

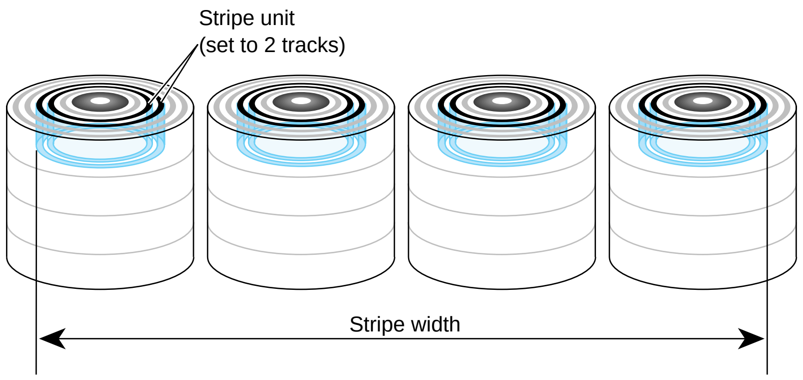

Figure 2-1 Effects of the Stripe Unit and Disk Number on Stripe Width

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_unit 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). If the 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.

Changing the Maximum DMA Size

IRIX imposes a limit on the maximum size of 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 16KB on 64-bit systems. Change these values to the size that you need, then reconfigure and reboot the server.

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 slower than XFS writes. This is due to the synchronous nature of BDS write operations.

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

Disks	Stripe Unit	Stripe Width	XFS Read	XFS Write	BDS Read	BDS Write
6	64KB	384 KB	29	14	26	10
6	128KB	768 KB	30	18	27	14
6	256KB	1536 KB	29	21	28	16
9	64KB	576 KB	42	20	38	15
9	128KB	1152 KB	43	26	37	19
9	256KB	2304 KB	42	31	38	22
12	64KB	768 KB	54	26	45	19
12	128KB	1536 KB	54	33	44	23
12	256KB	3072 KB	57	41	48	27
15	64KB	960 KB	65	32	52	21
15	128KB	1920 KB	64	41	53	26
15	256KB	3840 KB	68	50	52	30
18	64KB	1152 KB	75	37	58	24
18	128KB	2304 KB	73	47	57	29
18	256KB	4608 KB	80	57	62	33
21	64KB	1344 KB	84	43	64	26
21	128KB	2688 KB	84	54	52	31
21	256KB	5376 KB	90	66	68	31
24	64KB	1536 KB	92	48	68	29

Table 2-2 (continued) Performance Results With Sample Configurations [a]

Disks	Stripe Unit	Stripe Width	XFS Read	XFS Write	BDS Read	BDS Write
24	128KB	3072 KB	92	60	63	33
24	256KB	6144 KB	100	73	71	30
27	64KB	1728 KB	101	53	71	30
27	128KB	3456 KB	99	64	72	35
27	256KB	6912 KB	108	81	72	34
30	64KB	1920 KB	108	58	70	32
30	128KB	3840 KB	108	71	72	37
30	256KB	7680 KB	118	86	72	40

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 that filesystems that BDS is to use and test BDSpro performance on the filesystems. The chapter contains these sections:

- "Exporting Filesystems for BDSpro"
- "Mounting Filesystems for BDSpro"
- "Stopping and Restarting BDSpro"
- "Testing a BDSpro Setup"
- "Testing a BDSpro Setup"

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.

Exporting Filesystems for BDSpro

To make filesystems available to BDSpro, export them from the server with 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.

Mounting Filesystems for BDSpro

To mount filesystems on BDSpro clients, use the standard NFS *mount* command and the **-o bds** option (see the fstab(4) reference page for complete information). The example that follows illustrates a BDS entry in a client */etc/fstab* file:

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

In this 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.

Stopping and Restarting BDSpro

BDSpro software is ready to run after it is installed. You can start the *bds* daemon on the server from the command line if it is not already running (see the bds(1M) reference page for a list of options).

Note: If it is necessary to stop the server, stop all client applications first.

When the *bds* daemon starts, it binds to port 2050 (NFS port + 1). Do not stop the daemon during an active connection. Doing so causes the socket to linger while it tries to process all remaining packets. If you attempt to restart the daemon during this period, TCP prevents the restart to allow time for packet processing to finish (approximately two minutes).

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/sbin/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 an *lmdd* 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=100m
100 MB in 1.54 secs, 65.01 MB/sec
```

Note: If you do not include the **move=100m** argument, the *lmdd* command will not stop.

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 data 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=20m 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, `readn` timing results are network times and `write` timing results are XFS times.

```
server# bds -debug
```

```
(null): F xid=24512 uid=0 gid=0 fhandle len=68
Want file handle 68 bytes
V3 filehandle: A xid=24512 uid=0 gid=0 bytes=0
V3 filehandle: W xid=24513 uid=0 gid=0 off=0 len=4.0M
readn: 1000c000 4.0M @ 84.5M/sec
V3 filehandle: lseek to 0 = 0
write: 4.0M @ 52.0M/sec
V3 filehandle: A xid=24513 uid=0 gid=0 bytes=4.0M
V3 filehandle: W xid=24514 uid=0 gid=0 off=4.0M len=4.0M
```

```
readn: 10410000 4.0M @ 0.5G/sec
V3 filehandle: lseek to 4.0M = 4.0M
write: 4.0M @ 50.2M/sec
V3 filehandle: A xid=24514 uid=0 gid=0 bytes=4.0M
V3 filehandle: W xid=24515 uid=0 gid=0 off=8.0M len=4.0M
readn: 1000c000 4.0M @ 83.2M/sec
V3 filehandle: lseek to 8.0M = 8.0M
write: 4.0M @ 56.5M/sec
V3 filehandle: A xid=24515 uid=0 gid=0 bytes=4.0M
V3 filehandle: W xid=24516 uid=0 gid=0 off=12.0M len=4.0M
readn: 10410000 4.0M @ 88.6M/sec
V3 filehandle: lseek to 12.0M = 12.0M
write: 4.0M @ 55.7M/sec
V3 filehandle: A xid=24516 uid=0 gid=0 bytes=4.0M
V3 filehandle: W xid=24517 uid=0 gid=0 off=16.0M len=4.0M
readn: 1000c000 4.0M @ 88.5M/sec
V3 filehandle: lseek to 16.0M = 16.0M
write: 4.0M @ 52.9M/sec
V3 filehandle: A xid=24517 uid=0 gid=0 bytes=4.0M
hip0-ebony.engr.sgi.com moved 20.00 MB in 0.75 secs, 26.64 MB/sec
```

Note: Network writes are frequently returned before data transfers are completed. Transfers are mapped into socket buffers and the write is returned, but the network continues to transfer data after these events take place.

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:

```
# lmdd of=debugfile bs=4m move=20m direct=1 debug=1
0.003 bds_open(3, debugfile, 32769, 5)
restart(debugfile)
open(hip0-mahogany:/export/bds1/debugfile) = 3
write at 0.051, rwnd=0 swnd=0
0.263 A xid=2 uid=0 gid=0 bytes=4.0M
bds_write(3, 1000c000, 4194304) = 4194304 @ 18.72MB/sec in 1 writes
write at 0.268, rwnd=0 swnd=0
0.430 A xid=3 uid=0 gid=0 bytes=4.0M
bds_write(3, 1000c000, 4194304) = 4194304 @ 24.62MB/sec in 1 writes
write at 0.435, rwnd=0 swnd=0
0.562 A xid=4 uid=0 gid=0 bytes=4.0M
bds_write(3, 1000c000, 4194304) = 4194304 @ 31.38MB/sec in 1 writes
write at 0.567, rwnd=0 swnd=0
0.693 A xid=5 uid=0 gid=0 bytes=4.0M
bds_write(3, 1000c000, 4194304) = 4194304 @ 31.61MB/sec in 1 writes
write at 0.697, rwnd=0 swnd=0
0.821 A xid=6 uid=0 gid=0 bytes=4.0M
bds_write(3, 1000c000, 4194304) = 4194304 @ 32.23MB/sec in 1 writes
20.00 MB in 0.83 secs, 24.23 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 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.

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

- 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.