



Global File System (GFS) User Manual

Software Version	GFS-5.1.1
Document Version	1.0
Document Published	October 2002

Important Notice

Copyright © 2002 Sistina Software, Inc. All rights reserved.

Sistina Software, Inc.

1313 5th Street SE, Suite 111, Minneapolis, Minnesota 55414 USA

This product and related documentation are protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior consent of Sistina Software, Inc. and its licensors, if any.

The product(s) described in this manual may be protected by one of more U.S. patents, foreign patents, or pending applications.

All products mentioned in this document may be trademarks or registered trademarks of their respective owners.

This publication is provided as is without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

This publication could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein; these changes will be incorporated in new revisions of the publication. Sistina Software, Inc. may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

Preface

This document provides information necessary to install, configure and maintain GFS. It provides detailed explanations for all the normal tasks you will accomplish using GFS. More complex operations and examples of typical and tested GFS configurations are included.

This version of the User Manual is for GFS 5.1.1. This document does not describe the functionality or use of prior versions.

Text Conventions

Convention	Description	Examples
Prompts	Root command line prompt	<code>host-a\$</code>
	User command line prompt	<code>host-a%</code>
Commands	GFS and Unix commands	<code>gfsconf -p Cidev</code>
Variables	Variables that can be substituted in a command	<code>Pool_Name</code>
User Input	Text provided by user	<code>some_text</code>
File Names	Literal name of a file or path.	gfsconf.cf
New Terms and Important Notes or Warnings	Emphasis of a point or introduction of a term.	<i>failover</i> <i>Warning: Warning text.</i>

Contents

Chapter 1 - Introduction

What is GFS?	1
Features	2
For More Information	4

Chapter 2 - GFS Theory

Shared Block Storage	5
Cluster Volume Namespace - Pool	5
Locking	6
GFS Lock Modules	6
MultiFence	8
Cluster Applications	8

Chapter 3 - Determining Your Configuration

Decision Tree	9
Topology Illustrations	10

Chapter 4 - Getting Started

Installing GFS	15
Updates	18
Linux/GFS Limitations	22
Collecting Configuration Information	22
Installing Net-Telnet Perl Module	23
Preparing Devices	24
Creating Pools	25
Cluster Information Device and gfs_conf	27
Starting GFS	33
Congratulations	37
Ready After Boot	37

Chapter 5 - Configuration Procedures

Two GFS File Systems with Fibre Channel (FC) and lock_gulmd Lock Server ...	39
GFS with Fibre Channel and lock_swdmepd Lock Server	47
GFS with Fibre Channel and gulm Lock Server	53
GFS with Parallel SCSI and lock_gulmd Lock Server	59
GFS with GNBD	65
GFS as a Local File System	71
GFS on a Server's Root File System	71

Chapter 6 - Advanced Configuration Procedures

Adding New Storage Hardware	73
Setting up Failover for the lock_swdmepd Server	77
Adding a Node to a Cluster	77
Removing a Node from a Cluster	78
Adding Journals to a Cluster	78
Updating the GFS License	79
Using Context Dependent Path Names	80
Pool Multipathing	82
Enabling Large File Support	82

Chapter 7 - Performance

Mount Options	85
Pools	86
GFS File System Tunables	89

Chapter 8 - Fibre Channel Hardware

Fibre Channel Background Information	91
Fibre Channel Switches and Hubs	94
FC Under Linux	97
HBAs Used by Sistina Under Linux	99

Chapter 9 - Troubleshooting

Contacting Sistina	103
Requesting Technical Support	104
Solutions to Common Problems	105

Appendix A - Command/Module Summary

Appendix B - Forms

Glossary

Index

Chapter 1 - Introduction

What is GFS?

Current network technologies allow multiple nodes to share block storage devices. File systems that allow these nodes to simultaneously access (read and write) files on these devices are called *shared storage file systems*. This is in contrast to traditional network file systems where a server controls the devices and is the focal point of all data sharing.

The Global File System (GFS) is a shared storage device cluster file system for Linux. GFS supports multi-client journaling and rapid recovery from client failures.

Nodes within a GFS cluster physically share storage by a variety of methods. These methods currently include Fibre Channel (FC), multi-ported SCSI devices and network block devices. New methods for sharing block storage, such as iSCSI and Infiniband, will be supported in the future.

GFS is architecturally fully symmetric – all nodes are peers and cooperate to eliminate bottlenecks or single points of failure.

GFS uses read and write caching while maintaining full UNIX file system semantics.

GFS offers several advantages over a traditional client-server file system:

- Availability of the file system is ensured since failure of a single node, or even multiple nodes, in the cluster does not eliminate file system access to the remaining nodes.
- Load sharing a mixed workload among multiple clients sharing storage devices is simplified by the client's ability to quickly access any portion of the dataset on any of the storage devices via the GFS file system.
- Pooling storage devices into a unified logical volume equally accessible to all nodes in the system is possible. This simplifies storage management and reduces administration complexity.

- Scalability in capacity, connectivity and bandwidth can be achieved without the limitations inherent in network file systems like NFS which are designed around a centralized server.

Features

- **Symmetric Shared Storage File System** - GFS allows data sharing on shared storage via a cluster-wide, coherent file system. GFS is symmetric in that there are no server nodes (unlike NFS). Each node of the GFS cluster has direct access to the shared storage and is able to manipulate data on the storage in consistent and coherent manner via the GFS file system.
- **POSIX-Compliant** - GFS implements all file system elements required by POSIX.1 controls.
- **64-bit Files and File System** - All file sizes, offsets and block addresses in GFS are 64 bits.
- **True Cache Consistency** - A change to a file or directory on one node shows up immediately on all nodes in the cluster. Other distributed file systems take time to propagate changes from one node to another.
- **Dynamic Inodes** - There are no preallocated inode tables in GFS. Each inode in the file system is just a file system block, so there can be as many files in the file system as there are file system blocks. This means that as long as there is free space in the GFS file system, new files can be created without fear of running out of inodes.
- **Platform Independent Metadata** - All on-disk structures are written in a platform-independent format. Differences in structure packing and endianness are completely handled by GFS internals. This is highly important for GFS since all nodes need to understand and manipulate metadata as members of the cluster. Without this capability, it would not be possible to have a cluster file system that is both architecture (IA-32, Alpha and more) *and* OS heterogeneous.
- **Multi-Client Journalled** - GFS is a multi-client, journal file system. Multiple journals enhance GFS in two areas.
 - The journal no longer becomes a single point of contention in the file system and allows for greater scaling.
 - Multiple journals provide the capability of online recovery of the file system when a member node dies or is MultiFenced.

- **Bootable File System** - GFS can be a bootable file system. This allows it to be used as a standalone journal file system on a single node, as well as a cluster-wide shared root file system when combined with its context dependent path name (CDPN) capability.
- **Context Dependent Path Names** - Context Dependent Path Names (CDPN) make sharing GFS between different nodes, node types, and even operating systems much easier. It provides for the creation of links that point to different locations dependent on context (hostname, OS, architecture and so on). This feature coupled with GFS's ability to be a bootable file system can be harnessed to provide single root boot for a cluster.
- **Direct I/O Support** - Provides buffer cache bypass for applications that want to do direct I/O to the file system. This feature provides a huge speed improvement for applications like databases.
- **Full Shared mmap() Support** - GFS provides the ability to make cache-consistent, shared memory mappings of files. A memory read from an mmap()ed file will always see the absolute latest version of the file. A memory write into an mmap()ed file will be immediately seen by all other nodes in the cluster.
- **Cluster Wide File System Quotas** - GFS provides cluster wide BSD style quotas (UID/GID). Quotas can be either exact or *fuzzy* so as to meet the requirements of a wide range of applications. Fuzzy quotas allow for greater file system performance with only a small degradation in their exactness.
- **Online Growable** - GFS is an online growable file system. Both data and journal space can be added to the file system while it is online. When the file system is expanded, it is immediately recognized and available to all members of the cluster.
- **Cluster Wide Quiesce** - GFS supports the ability to *quiesce* the file system and then release it again. This capability can be combined with hardware that supports snapshotting or split mirrors to ensure that a consistent image of the file system is maintained when using these capabilities.
- **Read-only File System Support** - This feature allows mirrors or snapshots that have been previously taken to be mounted and made accessible in a read-only manner for general access.

- **Global Network Block Device (GNBD)** - GNBD allows a block device to be exported or presented over a TCP/IP network to any number of clients which import it. This allows GFS to be used in environments where shared block storage devices are not available via Fibre Channel or SCSI.
- **Multipath Support in Pool** - GFS's cluster volume manager (Pool) supports both failover and load balancing if multiple I/O paths are available to the shared block storage being used.
- **Cross Architecture Support** - GFS file systems run on architecture that supports Linux so that clusters can be heterogeneous (that is, a mixture of IA-32, IA-64, Alpha and more).
- **Modular Locking** - GFS supports a modular lock framework that allows different lock servers/modules to be used for locking in the cluster. This permits new lock servers/modules (such as DLM) to be integrated into GFS as they become available.
- **Flexible MultiFence Framework** - GFS provides a framework that MultiFence agents can be added to in a simple manner. This means that, as new MultiFence mechanisms become available, they can be deployed quickly and easily.

For More Information

To learn more about GFS, please take a look at the other documents available at our website:

http://www.sistina.com/products_GFS_publications.htm

Chapter 2 - GFS Theory

Shared Block Storage

GFS is designed to be used on shared storage devices. Fibre Channel devices are designed to be accessed from a network of hosts, so they are a natural choice for use with GFS.

Fibre Channel storage devices (RAID or JBOD (Just a Bunch of Disks)), switches and HBAs (Host Bus Adapters) are all required when using a Fibre Channel Storage Area Network (SAN).

Sharing storage devices over an IP network (Fast Ethernet and Gigabit Ethernet) can be accomplished in many ways. Nodes with local storage can allow block level access to their storage via the network using drivers like GNBD. Once network access to shared blocks is available, GFS can be placed above it.

Finally, parallel SCSI devices can sometimes be attached to multiple hosts for concurrent access. Where this is possible, GFS may be used.

Cluster Volume Namespace - Pool

Pool is a simple volume manager which can be used simultaneously by multiple nodes in a shared storage environment. It does not depend on host-based device naming schemes, which can be inconsistent in SANs due to devices being added, removed or detected in different orders. A pool device is seen with the same name and same sub-devices by all nodes in a cluster.

Like other volume managers, Pool merges multiple physical block devices into a single logical block device. A pool can be striped across multiple devices (RAID0) to increase bandwidth.

In the future, a cluster capable version of LVM can be used with shared storage and GFS.

Locking

Within GFS, locking is split into two primary layers.

- The upper layer (*glock* layer) is abstract and is managed completely within GFS.
- The lower layer is external to the GFS module. This lower layer (the lock module layer) is modular, interchangeable and encapsulates the details of a particular locking mechanism.

A GFS lock module must incorporate both a locking mechanism and a clustering subsystem. The lock module itself may contain complete locking and clustering functions internally. The lock module could also interface with external lock or cluster managers (future lock modules will do this).

The GFS Lock Harness is a kernel module which allows GFS and various lock modules to work together. This interchangeable lock module capability allows GFS to be independent of specific locking mechanisms. This is advantageous because a variety of locking techniques with different characteristics can be made available. The GFS file system module remains unchanged when lock modules are swapped.

GFS Lock Modules

Different lock modules usually take different approaches to locking and clustering algorithms or topologies. Every lock module looks the same to GFS since GFS sees them all through the common Lock Harness *glue* layer.

There are general locking and clustering features which all lock modules must support. On the locking side this includes:

- 4 specific lock modes
- A 9-byte lock namespace
- 32-byte lock value blocks
- Request canceling capabilities
- A *no-queue* or *try* option on lock requests

A lock module must also support a variety of clustering capabilities:

- Monitoring liveness of cluster nodes
- Coordinating recovery steps
- Assigning unique, non-sparse ID numbers to nodes
- A MultiFence system

Other than these requirements (and a few others), a lock module is free to operate however it wishes. Current lock modules use a network server to store

lock state and arbitrate lock requests from cluster nodes. Future lock modules will incorporate more fault tolerant topologies and algorithms.

GULM

The Global Universal Lock Module (GULM) uses a client-server architecture. The GULM server program runs in user space, preferably on a dedicated node. The GULM client is the actual lock module which is installed on the GFS nodes. GULM clients and servers cannot run on the same node. A single GULM server can manage GULM clients linked with several GFS file system instances.

All the locking and clustering capabilities are handled within GULM itself. The GULM server interfaces with the MultiFence system as a part of recovery operations. The GULM system reads its configuration information from either a CIDEV (Cluster Information Device) or a local copy of the GULM configuration file. This file contains GULM-specific information as well as cluster and fencing information.

DMEP

The DMEP lock module usually uses a client-server architecture. It can also use DMEP-capable storage devices in place of a server. The DMEP server program runs in user space on a dedicated, non-client node. The server program ostensibly emulates DMEP-capable storage devices, which makes it less efficient than the GULM server. A separate DMEP server program must be used for every DMEP-server-based file system.

The DMEP client modules contain most of the intelligence for clustering and locking and therefore are responsible for interfacing with the MultiFence system during recovery. This is different from the GULM system where only the GULM server interfaces with the MultiFence system.

The DMEP server program can optionally store all its internal state to disk. This allows the DMEP server to *failover* when used with other failover management software. In this mode, the DMEP server is much slower and file system performance suffers.

NoLock

The NoLock lock module allows GFS to be used on a single node as a local file system. No locking or clustering capabilities are provided.

MultiFence

The MultiFence system is used by both the GULM and DMEP lock modules. MultiFence disables a node's access to the shared storage at the block device interconnect or via power-cycling the fenced GFS node. Multiple methods can be listed as backup in case one fails.

When the clustering subsystem within the lock module detects an unresponsive node, it assumes the node has failed and is in need of recovery. Before recovering the file system for the failed node, the clustering system *fences* the node to prevent it from *reviving* and unexpectedly writing to the file system (thereby potentially corrupting the file system). MultiFence can be accomplished using special hardware like a Fibre Channel switch or a Network Power Switch. A specialized fencing agent program is used for each specific fencing method.

Cluster Applications

Parallel applications which need to synchronize access to files often use flock(2) file locks or fcntl(2) record locks. GFS makes both types of locks from applications consistent on all nodes. So, if a cluster application on one node flocks a file, the application on another GFS node will see the same lock.

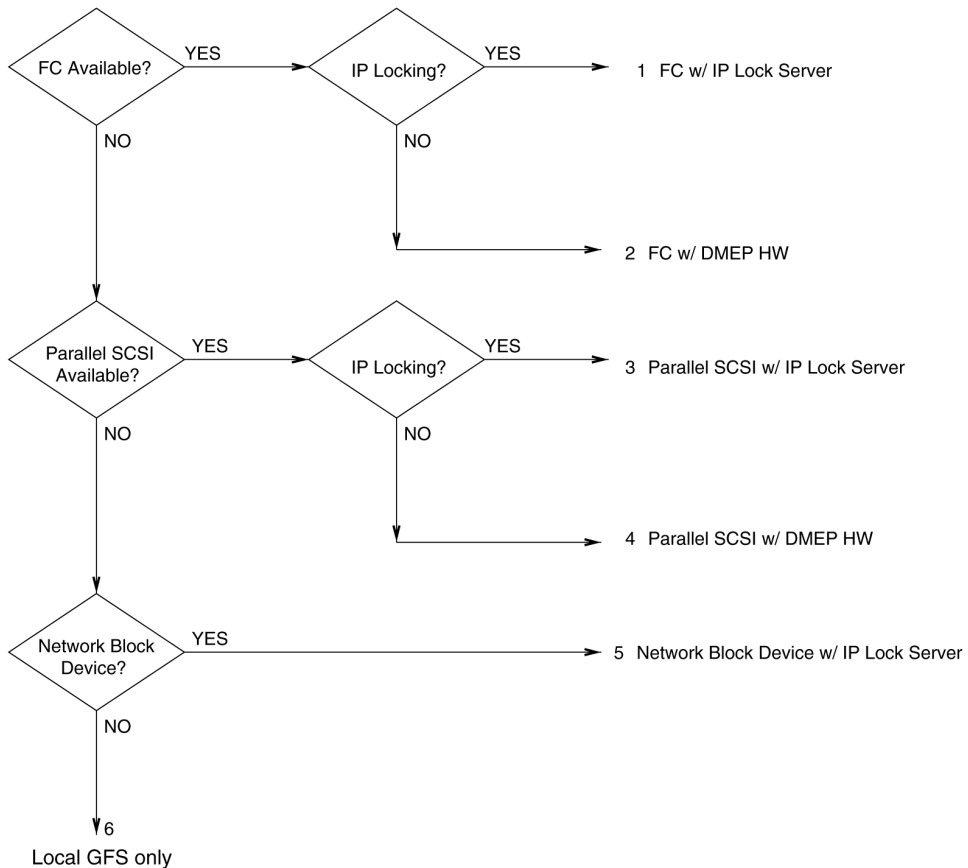
Chapter 3 - Determining Your Configuration

Decision Tree

You can configure GFS to suit your unique hardware configuration. GFS can scale from a single computer's IDE boot disk up to many computers with Fibre Channel attached disk arrays on a Fibre Channel network.

To get started:

- 1** Answer the questions in the following decision tree to determine the topology that best suits your configuration.
- 2** Find the illustration of your topology from the choices in Figures 1-6.
- 3** Refer to the configuration procedures in Chapter 5 for information on how to implement your topology.
- 4** For additional capabilities, refer to the advanced configuration procedures in Chapter 6.



Topology Illustrations

Chapter 5 provides configuration procedures for each of the following sample topologies.

Fibre Channel with IP Lock Server (lock_swdmepd or lock_gulmd)

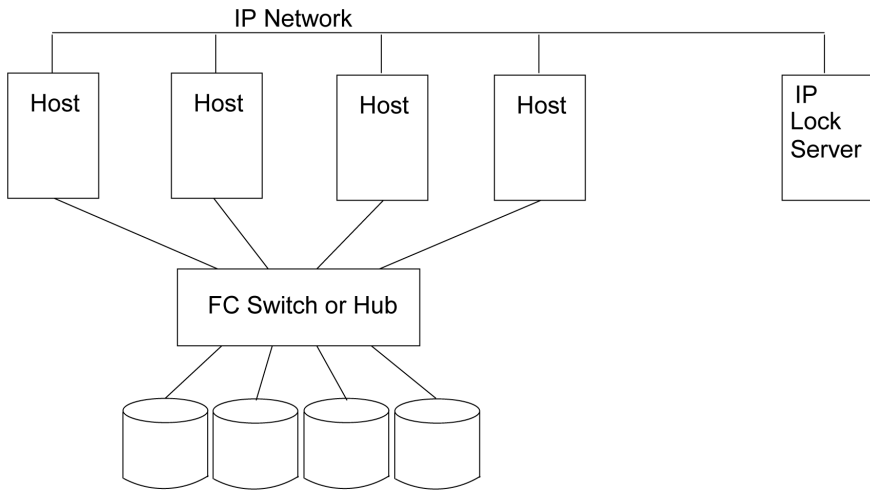


Figure 1. Fibre Channel with IP Lock Server

Fibre Channel with DMEP Hardware (lock_hwdmepd)

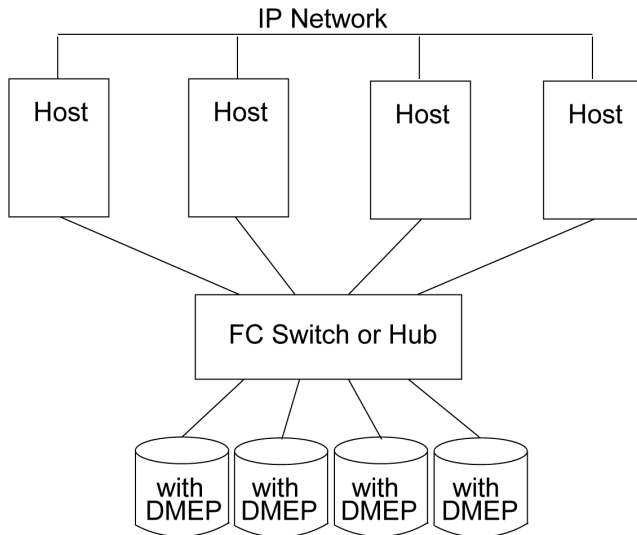


Figure 2. Fibre Channel with DMEP Hardware

Parallel SCSI with IP Lock Server (lock_swdmepd or lock_gulmd)

Note: lock_swdmepd servers are limited in scalability by SCSI constraints.

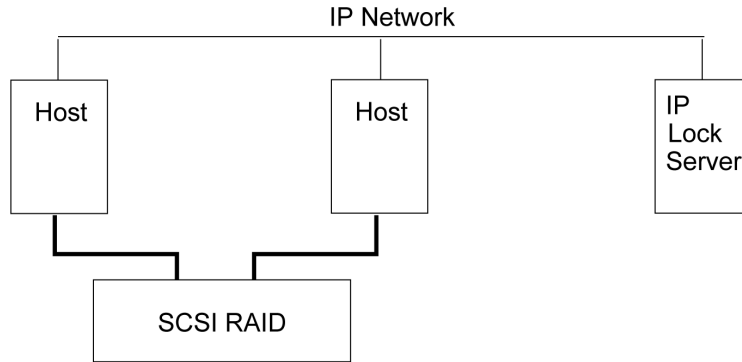


Figure 3. Parallel SCSI with IP Lock Server

Parallel SCSI with DMEP

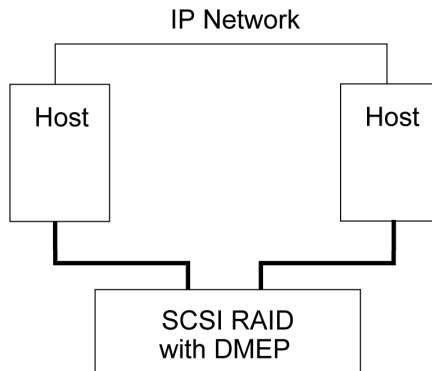


Figure 4. Parallel SCSI with DMEP Hardware

Network Block Device with IP Lock Server (lock_swdmepd or lock_gulmd)

IP-based lock servers are much slower than either Fibre Channel or parallel SCSI. On the other hand, there is more flexibility when configuring a GFS cluster since the only requirement for membership is having an IP connection.

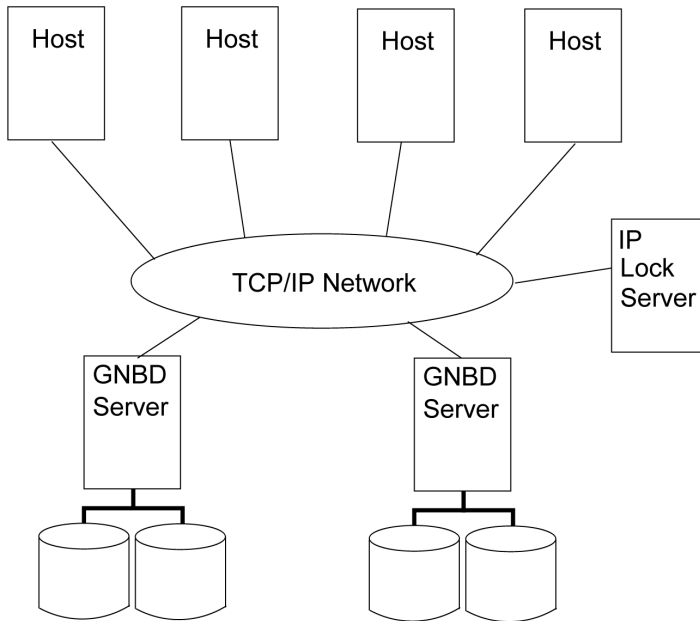


Figure 5. Network Block Device with IP Lock Server

Local File System

As a local file system, GFS provides a solid journaled file system. A local GFS file system cannot be shared. If you want to have a shared GFS file system, you must use one of the previous options.

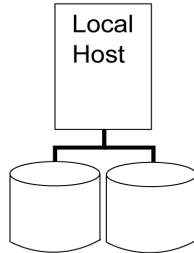


Figure 6. Local File System Configuration

Chapter 4 - Getting Started

Installing GFS

Please refer to the Installation Instructions provided during the download process for procedures concerning the installation of GFS RPMs.

***Note:** If this is an upgrade from a pre-5.1.1 GFS, you must upgrade the existing file system after the installation of the GFS 5.1.1 tools and modules. Until the file system is upgraded, it will not be usable. Refer to “Updates” on page 18 for more details.*

The installation instructions from the GFS download page are included below as a convenience:

- 1** Select the appropriate kernel.

First determine which kernel best fits your requirements. Sistina recommends the use of one of the kernels provided on the download site or a kernel built from kernel sources from **kernel.org** that have been patched for optimal GFS use. The kernel RPMs provided on the download site have had GFS performance patches applied to them and were compiled against the modified, vendor's kernel source with their config files.

Currently supported distributions/releases are:

- RedHat (7.2, 7.3, 2.1AS)
- SuSE (7.3, 8.0, SLES7).

For each of these distributions, Sistina provides RPMs that will run against a vendor kernel with GFS patches applied, an unmodified vendor kernel, and a kernel built from sources from **kernel.org** with GFS patches applied.

If you want to use your own custom kernel, you will need to download the GFS patches and apply them to a clean 2.4.18 **kernel.org** kernel. You will then need to configure your kernel to use KMOD and disable MOD-VERSIONS. Finally, you must use the

config file that was used during the RPM build. This file and the GFS kernel patches can be found on the download site.

GFS will work against a *stock* RedHat 7.2 kernel (2.4.7-31), 7.3 kernel (2.4.18-4), 2.1AS kernel (2.4.9-e.3), as well as, *stock* SuSE 7.3 kernel (2.4.10-4GB) and SLES7 (2.4.7-4GB). Since these kernels do not have GFS patches applied, certain features will be unavailable and there is a loss in performance. It is strongly encouraged that you use a prepatched kernel from the GFS download site or apply the GFS patches found at:

http://www.sistina.com/products_gfs_download.htm

Click on the **Kernel_patches for GFS 5.1** link.

Note: This location on our website contains patches for all supported kernels.

2 Select the GFS RPM.

If you chose to install one of our kernels, select the GFS RPM that best goes with the kernel RPM.

If you chose to use a custom kernel, select the distribution you are currently running and use the RPMs that were generated for a **kernel.org** kernel on that distribution.

If you chose to use the stock distribution kernel, install the GFS stock RPM that corresponds to your distribution.

3 Download and install the documentation tarball, which contains the GFS manual and sample configuration files. We strongly recommend installing this. The tarball will be extracted into the directory where it resides via:

```
host-a$ tar zxvf GFS-docs-5.1.tar.gz
```

This will create a directory called **GFS-docs** that contains the GFS manual and sample configuration files.

4 Install the kernel RPM.

The kernel RPMs will install a new kernel and associated kernel modules but do not make the appropriate modifications for either **grub** or **lilo**. It will be necessary to edit either the **grub** or **lilo** configuration file to reflect which kernel should be used at the next boot. It is recommended that you make backups of your current kernel, **initrd** and **grub/lilo** configuration file before installing the kernel RPM.

5 Install the GFS RPM.

If the Sistina kernel RPM was installed for a SuSE or RedHat system, the GFS install should be straightforward.

Install the GFS RPM in the same manner as you would any other RPM. For more information, refer to the man page for rpm.

If you decide to install a GFS RPM that was generated for a **kernel.org** kernel (that has been modified with the Sistina GFS patches), you must generate an appropriate **kernel.org** kernel. Generate this kernel by acquiring the GFS kernel patches from the download site, as well as the kernel config file that was used during RPM generation.

6 Prior to configuring and running GFS, you may need to:

- Run the following command so that the GFS modules can be recognized:

```
host -a$ depmod -a
```
- Install the Net-Telnet Perl module. See “Installing Net-Telnet Perl Module” on page 23. This module is required by most of the MultiFence agents.

If you need assistance, please email your question to our customer support department at **support@sistina.com**. For more information on contacting Sistina, see “Contacting Sistina” on page 103 and “Requesting Technical Support” on page 104.

Updates

This section addresses updates that occurred between prior versions of GFS and the GFS 5.1.1 release.

Command/Module Name Changes

For v5.1.1, GFS command names and kernel modules were renamed to better reflect functionality. If this is a new installation of GFS, you do not need the following information. If you have an earlier version, you can track the name changes in Table 1.

Table 1 GFS v5.1.1 Name Changes

Functional Area	Old Name	New Name
GFS Kernel Modules	gfs.o	gfs.o
	gnbd.o	gnbd.o
	gulm.o	lock_gulm.o
	kgnbd.o	gnbd_serv.o
	lock_harness.o	lock_harness.o
	lockstats.o	lock_stats.o
	memexp.o	lock_dmep.o
	nolock.o	lock_nolock.o
	pool.o	pool.o
GFS File System Commands	stomith.o	fence.o
	gfsconf	gfs_conf
	gfsck	gfs_fsck
	gfs_tool	gfs_tool
	gfs_jadd	gfs_jadd
	gfs_expand	gfs_grow gfs_quota
GNBD Commands	mkfs_gfs	gfs_mkfs
	gclient gserv	gnbd_import gnbd_export
GFS Lock/Servers Commands	initds	lock_swdmep.initds
	memexpd	lock_swdmepd
	Gulm_LT	lock_gulm_LT
	Gulm_Core	lock_gulm_Core

Table 1 GFS v5.1.1 Name Changes

Pool Commands	passemble	pool_assemble
	pgrow	pool_grow
	pinfo	pool_info
	ptool	pool_tool
MultiFence Commands	do_apc_ms	fence_apc_ms
	do_brcd_port	fence_brcd_port
	do_brcd_zone	REMOVED
	do_dmep	REMOVED
	do_gclient_stm	fence_gnbd
	do_wti_mps	fence_wti_mps
	stomith_d	fenced
	wait_manual	fence_manual ^a

^afence_manual is not recommended for production use.

Upgrading CIDEV Format

The on-disk format for the Cluster Information Device (CIDEV) has changed in order to support the new locking modules. To run GFS v5.1.1, you must upgrade the CIDEV.

Warning: *Once you upgrade a CIDEV to a GFS v5.1.1 format, the CIDEV cannot be used for previous GFS versions.*

- 1** Before proceeding with this upgrade, make sure all nodes have *cleanly* unmounted the GFS file system and that all nodes in the cluster have been upgraded to GFS v5.1.1.
- 2** In order to reuse information in the old CIDEV (like node names, IP addresses, Fibre Channel switch ports, etc.), store the old CIDEV to a file. The following command will read the old CIDEV from disk and store it to a file called **gfs.cf.old**:

```
host-a$ gfs_conf -p -d Cidev > gfs.cf.old
```

- 3** After storing the old CIDEV format to a file, erase the on-disk CIDEV using the following command:

```
host-a$ gfs_conf -e -d Cidev
```

- 4 Edit the file **gfs.cf.old** to conform to the GFS v5.1.1 format. See “Cluster Information Device and gfs_conf” on page 27 for the GFS v5.1.1 format of the new CIDEV. The format will depend on the lock module you choose.

Note: When converting an old CIDEV, please note that, in the `node` section the entry `sm` has changed to `fm`, and, in the `stomith` section, the keyword `stomith` has changed to `fence`:

- 5 After editing the file, write the new CIDEV to disk with the following command:

```
host-a$ gfs_conf -c -f gfs.cf.old -d Cidev
```

Upgrading GFS Ondisk Format

The ondisk format for GFS has changed in order to support cluster quotas. There have also been name changes made to locking protocols that GFS supports. The locking protocol for a GFS file system is stored in the GFS file system superblock, so these values need to be updated to match the new lock module names.

Before using a file system that was created by an older version of GFS (anything from v4.0 to v5.0.1), you must upgrade the file system. GFS v5.1.1 will not be able to use the file system until it is upgraded.

Warning: Once you've upgraded a GFS file system to a GFS v5.1.1, it cannot revert to a prior version.

Perform the following steps to upgrade a GFS file system:

- 1 Cleanly unmount the GFS file system that is being upgraded from all the nodes in the cluster. The GFS file system must not be mounted by any nodes during the upgrade process.
- 2 Since lock protocol names have changed, update the existing name in the superblock using `gfs_tool` to update the value in the GFS file system superblock.

```
host-a$ gfs_tool sb Pool_Partition proto \  
New_Lock_Protocol_Name
```

where the lock protocol names are:

v5.0.1	v5.1.1
nolock	lock_nolock
memexp	lock_dmep
gulm	lock_gulm

For example, to change a v5.0.1 nolock file system on the device **/dev/pool/pool0** to be compatible with the v5.1.1 lock protocol update, execute:

```
host-a$ gfs_tool sb /dev/pool/pool0 proto \
lock_nolock
```

- 3 Upgrade the GFS file system's super block to the new format. On a single node, mount the file system with the `upgrade` mount option:

```
host-a$ mount -t gfs Pool_Partition \
GFS_Mount_Point -o upgrade
```

- 4 If cluster quotas are going to be used for this GFS file system, they must be initialized using the following command:

```
host-a$ gfs_quota init -f GFS_Mount_Point
```

***Warning:** This may take a while on a file system that is populated with a large number of files.*

- 5 Install the GFS license as described below.

Installing the GFS License

GFS requires a license file installed in the GFS file system to allow writing of files, although you can read files and create directories and empty files. A license should have been provided with the GFS binaries. To install it:

- 1 Verify the validity of the license via:

```
host-a$ gfs_tool checklicense License_File
```

- 2 Mount the GFS file system the license is for on a single node using:

- 3 Install the license into the GFS file system via:

```
host-a$ gfs_tool setlicense Mountpoint License_File
```

You only need to run this command once (on a single node) for each GFS file system.

- 4 Mount the GFS file system on all other nodes that will access it.

Linux/GFS Limitations

Consider the following when planning and designing a GFS cluster configuration.

- Linux file systems are limited to a 2 TB maximum size due to limitations in the low level block device code. GFS is bound by this restriction as well. GFS is a 64-bit file system so, when this restriction is removed, it will be capable of addressing multi-terabyte GFS file systems.

Work is currently progressing in addressing the 2 TB block device limitation in the Gelato Project (<http://www.gelato.org>).

- Some device drivers under Linux are not 32-bit clean (they are coded using signed rather than unsigned integers). This means that some block devices may be limited to a maximum size of 1 TB.
- The Pool subsystem in GFS currently supports a maximum of 128 pools. Pools with static minor numbers are confined to minor numbers 1-64. Pools with dynamically assigned minors numbers are confined to minor numbers 65-128.
- GFS does not currently support clusters larger than 255 nodes. This is not an architectural limitation but one which is defined in the code. Once clusters large than 255 nodes have been tested, characterized, and verified, this maximum will be increased.
- GFS TCP/IP-based lock servers can be run over any compliant TCP/IP network. However, running TCP lock servers on anything slower than a 100BaseTX network is not recommended.

Collecting Configuration Information

Use the forms in Appendix B to collect the information required to configure GFS. Stepping through the configuration will be easier if you record this information in advance. Depending on your configuration, all the collected information may not be required.

The following is required for all configurations except *local/nolock*.

- IP addresses for each GFS node
- A free IP port on each GFS node for callbacks (default is 3001 for DMEP, 42040 for GULM) for each GFS file system.
- Shared Storage Device Designation

- Number and size of the required GFS file systems

***Note:** Each GFS file system using DMEP locking requires a small partition for the GFS cluster information. For GULM locking, only one CIDEV is needed for the entire cluster.*

The following may be required as well:

- IP lock server
 - IP address of the lock server.
 - A free IP port number (default is 15697 for DMEP, 42040 for GULM) for each GFS file system.

***Note:** Multiple GFS file systems require different IP port numbers.*

- Fibre Channel hardware:
 - Fibre Channel switch port used for each node and storage.
 - IP address of the FC switch (if used for MultiFence).
 - Login and password for the FC switch (if used for MultiFence).
- Network Power Switch hardware (if used for MultiFence):
 - Plug number for each GFS node
 - IP address of the Power switch
 - Login and password
- For DMEP hardware, the data pool that contains the DMEP device.

Installing Net-Telnet Perl Module

A majority of the MultiFence agents used by GFS require communicating with remote hardware to perform MultiFence operations. Examples of such hardware are a Fibre Channel or Network Power Switch. Communication with the hardware is accomplished via the Net-Telnet Perl module, which must be installed on the system.

- For DMEP, install Net-Telnet on all the cluster nodes
- For GULM, install Net-Telnet on the lock server node(s).

You can download the Perl module from the CPAN website <http://cpan.perl.org/>. The name of the module is Net-Telnet and it is located in the CPAN directory **modules/by-module/Net/**.

The following installation procedure is from the Net-Telnet.readme file. To install the module, go to the directory containing the unpacked distribution and do one of the following:

- Create a makefile by running Makefile.PL using the perl program into whose library you want to install and then run make three times:

```
perl Makefile.PL
make
make test
make install
```

- To install into a private library, for example your home directory:

```
perl Makefile.PL INSTALLSITELIB=$HOME/lib
INSTALLMAN3DIR=$HOME/man
make
make test
make pure_install
```

Alternately, you can just copy or move **Telnet.pm** from the distribution into a directory named **Net/** in the Perl library. You can then manually build the documentation using `pod2man` or `pod2html`.

Preparing Devices

It is assumed at this point that the controllers on the shared storage devices are properly configured. If in doubt, please refer to the manual that came with your shared storage devices for configuration information.

- 1 Determine if the shared storage devices to be used by GFS are recognized and accessible from one of the GFS nodes.

```
host-a$ cat /proc/partitions
```

If the storage device is not recognized and/or visible, you may need to load the appropriate device driver for the FC/iSCSI host bus adapter (HBA) or SCSI controller in the node.

```
host-a$ modprobe Device_Module
```

If the driver is already loaded, try unloading it.

```
host-a$ rmmmod Driver_Name
```

```
host-a$ insmod Driver_Name
```

Check the partition list in **/proc** to determine if the storage device is recognized and visible now.

```
host-a$ cat /proc/partitions
```

- 2 Create partitions on the storage devices. Keep the following in mind when creating partitions:

- If pools are to be striped across storage devices, it is best to create partitions of the same size for the stripes.
- If you will use separate journal pools, it is usually desirable to create a few extras in case you increase the number of nodes in the cluster. Mixing journal pools and data pools is not permitted – extra space in one cannot be used by another.
- In general, the CIDEV partition only needs to be about 1 MB for a cluster of 255 nodes.
- Use `fdisk`, `cfdisk` or `sfdisk` (or another partition manager of your choice) to partition the storage device as desired. Remember that a small partition is needed for the cluster information device.
- If the DMEP lock module is to be used, there needs to be one CIDEV for each GFS file system.
- If the GULM lock module is to be used, there only needs to be one CIDEV, even for several GFS file systems.

Creating Pools

This section describes how to create pool configuration files required for all GFS configurations – except local/nolock. Writing the pool configuration to the storage devices and loading pools is described in “Starting GFS” on page 33.

It is highly recommended that you allocate at least two pools for each GFS file system – a data pool for the GFS file system meta-data and data, and a small pool for the CIDEV.

`pool_tool` uses a text-based configuration file to create a pool on a device (or group of devices). `pool_tool` writes a label to the head of each storage device identifying its position in the pool. This label is independent of local storage device ID, which means that all nodes see the same pool device regardless of the order in which the shared storage devices are identified.

As with formatting, run `pool_tool` on one node to create a pool on the storage device(s).

The `pool_tool` options are documented in the man page. To see usage, run:

```
pool_tool -h
```

Below are the required fields in the `pool_tool` configuration file. Start by making a copy of the file `sample_pool_param`, which can be used as a

template. The sample configuration file has comments that describe each of the sections.

- `poolname name`

Pick a name for the pool (something like `pool0`). The block device node for the new pool will have this name (for example, `/dev/pool/pool0`).

```
poolname pool0
```

- `subpools number`

number is the total number of subpools. Come back to this if you don't know how many you will be creating. You are encouraged to start with one subpool. For a description of when you may want to use more than one subpool, see “Pool Multipathing” on page 82 and “Subpools” on page 86.

```
subpools 1
```

- `subpool id stripe_size number_of_devices type`

This is where subpools are defined.

- `id` - a sequential subpool identifier starting at zero.
- `stripe_size` - the number of sectors (512 byte blocks) in a stripe width. If you have a single device make stripe size zero.
- `type` - for the basic setup described here, all storage devices (4) will be in this single subpool and the type will be `gfs_data`.

```
subpool 0 128 4 gfs_data
```

- `pooldevice sp spid node weight`

The `pooldevice` statements specify which devices belong to the subpools defined in the previous step.

- `sp` - subpool number this device belongs to.
- `spid` - sequential counter of devices in a subpool starting with 0, 1, 2, ...
- `node` - device node (for example: `/dev/sdb1`).
- `weight` - determines the preference of hardware DMEP requests among subpool devices. In our example below, `sdb1`, `sdc1`, `sde1` will have DMEP requests mapped to them 1/5 of the time, while

sdd1 will receive 2/5 of the requests. (Journal partitions should be given a weight of 0.)

```
pooldevice 0 0 /dev/sdb1 1
pooldevice 0 1 /dev/sdc1 1
pooldevice 0 2 /dev/sdd1 2
pooldevice 0 3 /dev/sde1 1
```

For information on advanced subpool usage, see “Pool Multipathing” on page 82 and “Subpools” on page 86.

Cluster Information Device and gfs_conf

Global cluster information must be available to every node in a GFS cluster. This data is stored on a globally accessible (shared) device or partition called the Cluster Information Device (CIDEV). The CIDEV is identified in the superblock of the GFS file system. When using DMEP locking, the `-t LockTable gfs_mkfs` option is the CIDEV. For GULM locking, the `-t` option is the name of the file system as given in the GULM config file.

Information on the CIDEV includes:

- The lock server for the file system. This may be an IP address and port of an IP lock server or a pool device through which hardware DMEP commands are accessible.
- The port number used for inter-node lock callbacks.
- The timeout; or number of seconds a node may go without heartbeating before it's considered dead and subject to MultiFence procedures followed by recovery.
- The IP address of every node which may mount a particular GFS file system.
- When using DMEP locking, the client ID (CID) of every GFS node.
- Specific MultiFence methods which can be used to I/O fence any GFS node in the cluster.

The space required for the shared partition or storage device used as the CIDEV is not very large. Four Kbytes of storage is used for each GFS node, so one megabyte will suffice for clusters of up to 256 nodes.

Because physical devices frequently have different names across a SAN, it is best to make the CIDEV a small pool. If the GFS file system exists on a pool named `pool0`, a good name for the CIDEV is `pool0_cidev`.

Creating the GFS Configuration File

The program used to configure/modify the CIDEV is `gfs_conf`, which reads information from a configuration file and writes it to the CIDEV. This section describes the key components of the configuration file. DMEP- and GULM-based GFS clusters share some, but not all, of the definitions and keywords. Components specific to each are broken out below, as well as the components in common.

Hardware DMEP

- `gfsparam:`
Beginning of GFS/DMEP cluster declaration
- `lmtype = dmep`
Declares the lock manager type that is to be used by the GFS cluster
- `cmttype = dmep`
Declares the cluster manager to be used by the GFS cluster
- `datadev =`
Specifies the shared storage device to be used by the GFS cluster. This component is optional but recommended; it can make debugging and problem analysis easier.

```
gfsparam:  
lmtype = dmep  
cmttype = dmep  
datadev = Shared_Storage_Device_For_GFS
```

Software DMEP

- `lmparam:`
Beginning of GFS/DMEP lock manager declaration section
- `cbport = IP_Port_Number`
This is the UDP port used for inter-node callbacks. It needs to be different for each GFS file system that uses DMEP locks and is mounted. No other programs/daemons should be listening/communicating on this port. A good value is 3001.

- server = IP_Address_Lock_Server
 - serverport = IP_Port_Lock_Listening_On
- lmparam:
cbport = IP_Port_Number
server = IP_Address_Lock_Server
serverport = IP_Port_Lock_Listening_On

Hardware DMEP

- lmparam: = Beginning of GFS DMEP lock manager declaration section.
 - cbport = IP_Port_Number. This is the UDP port used for inter-node callbacks. This number needs to be unique for each GFS file system that uses DMEP locks and is mounted. No other programs/daemons should be listening/communicating on this port. A good value is 3001.
 - dmepdev = DMEP_Capable_Device or Pool_Device
 - segment = Segment_Number
- lmparam:
cbport = IP_Port_Number
dmepdev = DMEP_Capable_Device or Pool_Device
segment = Segment_Number

GULM

- gulmcluster = Specifies start of the GULM cluster definition. The following subcomponents are required:
 - node = Node where the lock server is running
 - port = IP Port it is listening for requests on
 - heartbeat_rate = Heart beat every X seconds
 - allowed_misses = How many heartbeats can be missed before MultiFence occurs
- gulmcluster:
node = Node where the lock server is running
port = IP Port it is listening for requests on
heartbeat_rate = Heart beat every X seconds
allowed_misses = How many heartbeats can be missed before MultiFence occurs

- `locktable`: Start of a lock table interface definition. There may be multiple lock table interfaces. The following subcomponents are required:
 - `name` = lock table name
 - `node` = where the lock table server is running
 - `filesystem` = GFS file system this lock table is associated with
 - `port` = IP port the lock table server is listening on
 - `range` = Range of locks being provided by this lock table
 - `verbose` = logging control
- `locktable`:
- `name` = lock table name
 - `node` = where the lock table server is running
 - `filesystem` = GFS file system this lock table is associated with
 - `port` = IP port the lock table server is listening on
 - `range` = Range of locks being provided by this lock table
 - `verbose` = logging control

Defining MultiFence Method(s)

The `fence` command declares a MultiFence method for the cluster. It is followed by a set of name-value pairs that describe options for that MultiFence action. Each MultiFence method has its own set of options, described below. Some methods require that the hardware be correctly configured for the MultiFence action to work. It is permitted to declare multiple MultiFence actions for a specific node.

WTI Network Power Switch models 115, 230 (`fence_wti_nps`)

The parameters are:

- `name` = unique name of this method
- `agent` = Agent binary
- `ipaddr` = IP address of the Network Power Switch
- `passwd` = Password for login

The node-specific parameter is the plug number which the specific node is plugged into.

```
fence:  
name = Nps_One  
agent = fence_wti_nps  
ipaddr = 10.0.1.9  
passwd = Password
```

APC MasterSwitch (fence_apc_ms)

The parameters are:

- name = Unique name of this method
- agent = Agent Binary
- ipaddr = IP address of the MasterSwitch
- login = Login Name
- passwd = Password for Login

The node-specific parameter is the outlet number which the specific node is plugged into.

```
fence:  
name = Apc1  
agent = fence_apc_ms  
ipaddr = 10.0.1.21  
login = Apc  
passwd = Apc
```

Brocade FC switch port (fence_brcd_port)

The parameters are:

- name = Unique name this method
- agent = Agent Binary
- ipaddr = IP address of the FC switch
- login = Login Name to the switch
- passwd = Password for login

The node-specific parameter is the port number where the node is connected on the switch.

```
fence:  
name = Brocade  
agent = fence_brcd_port  
ipaddr = 10.0.1.3  
login = Root  
passwd = Password
```

GNBD Network Block Device (fence_gnbd)

This has one required parameter:

- `name` = Unique handle to refer to this particular method
- `agent` = Agent Binary

The node-specific parameter is the IP address of the node.

```
fence:  
name = Gnbd  
agent = fence_gnbd
```

Manual (fence_manual)

This has one required parameter:

- `name` = Unique name of this method
- `agent` = Agent Binary

The node-specific parameter is the IP address of the node.

```
fence:  
name = manual IP_Address_of_Node  
agent = fence_manual
```

Note: fence_manual is not recommended for production use.

Defining Member Nodes

- `node` = Keyword to signify start of node definition. This is the description of each node which may mount the file system. An entry is required for each node in the cluster.
- `name` = Unique name to refer to this node
- `cid` = Cluster ID of the node; for DMEP only. This is a numeric identity of the nodes; each node must have a different number. This is best done by just starting at 0 and incrementing by 1 for each node.
- `ipaddr` = IP Address of the node. This should be the same IP address used as the `hostdata` mount option.
- `fm` = MultiFence method node-specific parameters. Each node can use an arbitrary number of MultiFence methods. These are started with the `fm:` tag, followed by a name descriptor from the MultiFence definition section. Depending on the MultiFence method, node specific parameters may be required after the name.

For example, here are two nodes with a single MultiFence method using the Network Power Switch described above.

```
node:  
name = node1  
cid = 0  
ipaddr = 10.0.0.3  
fm = nps_one 1  
fm = manual
```

Here is an example of using more than one MultiFence method. In this scenario, the methods are executed in the order they are defined in the node definition. The first method is tried, and if that fails, the next is executed.

```
node:  
name = node1  
cid = 0  
ipaddr = 10.0.0.3  
fm = nps_one 1  
fm = manual
```

```
node:  
name = node2  
cid = 1  
ipaddr = 10.0.0.4  
fm = nps_one 2  
fm = manual
```

```
node:  
name = node3  
cid = 2  
ipaddr = 10.0.0.5  
fm = nps_one 3  
fm = manual
```

Starting GFS

This section describes what to do after gathering all required information and creating the files required to activate GFS on a cluster/node, including:

- 1 Loading GFS modules
- 2 Writing the pool configuration to the storage devices
- 3 Loading the pool(s)
- 4 Writing the configuration information device (CIDEV)
- 5 Creating the GFS file system
- 6 Starting required daemons

- 7 Configuring the hardware DMEP device if this is being used
- 8 Mounting the GFS file system

Note: Steps 1, 3, 6 and 8 must be done on all the GFS nodes. Steps 2, 4, 5 and 7 must be done on one of the GFS nodes only. GNBD, GFS local and GFS root are special cases and will be covered in separate sections.

- 1 Load the GFS modules on each GFS node. The modules to load depend on your configuration. It may be necessary to build the module dependency tree after installing the GFS RPM via:

```
host-a$ depmod -a
```

All configurations require:

```
host-a$ modprobe gfs
```

DMEP configurations require:

```
host-a$ modprobe lock_dmeop
```

GULM configurations require:

```
host-a$ modprobe lock_gulm
```

```
host-a$ modprobe pool
```

- 2 Write the pool files to disk (from one GFS node only):

```
host-a$ pool_tool Parameter_File_Name
```

This writes the pool labels to the storage devices and checks to see if labels are about to be written over previously written labels or if they are about to be written onto an **ext2** partition. This is a precautionary measure. If you see one of these warnings, carefully think about the partitions you've chosen to become part of your pool. The warning will say something like:

```
You are attempting to write labels onto a  
device with an EXT2 FS. This may be a partition  
you are using, are you sure you want to do  
this? (Y)es, (Q)uit
```

It is up to you to make the correct choice on whether to proceed in the event of a warning.

The reverse of this operation is to erase the labels on the devices. To do so, use the **-e** option as follows:

```
host-a$ pool_tool -e Pool_Name
```

- 3 Load the pools (on all GFS nodes). The `pool_assemble` command scans all partitions and loads the pools it finds. You must do this at least once before mounting the file system. Also you must load the `pool.o` kernel module before running `pool_assemble`.

```
host-a$ pool_assemble
```

When `pool_assemble` identifies a pool, it dynamically creates a new device node in the `/dev/pool/` directory. The name of the device node is the poolname as specified in the pool configuration file.

```
host-a$ pool_assemble
```

You can unload pools by running `pool_assemble` with the `-r` option. To unload all pools that are currently active: use:

```
host-a$ pool_assemble -r
```

A single pool can be unloaded via:

```
host-a$ pool_assemble -r Pool_Name
```

- 4 Write the configuration information (from one GFS node only). To store the cluster information from the configuration file onto the CIDEV, run the following create command:

```
host-a$ gfs_conf -c -f Configuration_File \
-d Cidev
```

If successful, this will echo back the configuration which has been written. If `gfs_conf` finds a cluster config already on the CIDEV, the create command will fail with a warning message. The previous config can be erased with the command:

```
host-a$ gfs_conf -e -d Cidev
```

To print the current cluster configuration stored on a specific CIDEV:

```
host-a$ gfs_conf -p -d Cidev
```

- 5 Make the file system (from one GFS node only). The standard options will be explained here. For more information, please see the man pages for `gfs_mkfs`.

DMEP configuration:

- `-j number` - Number of journals to create. You need at least one for each GFS node.
- `-p Locking_Protocol` - Name of the locking protocol to use (in this case `lock_dmeop`).

- `-t Lock_Table_Name` - Name of the CIDEV pool (in this case `/dev/pool/pool0_cidev`).

```
host-a$ gfs_mkfs -j 8 -p lock_dmeq \
-t /dev/pool/pool0_cidev /dev/pool/pool0
```

GULM configuration:

- `-j number` - The number of journals to create. You need at least one for each GFS node.
- `-p Locking_Protocol` - The name of the locking protocol to use (in this case `lock_gulm`).
- `-t Lock_Table_Name` - This will be the name of the file system (in this case `myfs`).

```
host-a$ gfs_mkfs -p lock_gulm -t myfs -j 8 \
/dev/pool/pool0
```

- 6 Start the daemons. On each GFS node using DMEP, you need to start `fenced`.

```
host-a$ /sbin/fenced
```

- DMEP IP lock server - If you're using a DMEP IP lock server, you need to start `lock_swdmepd` on the lock server node.

```
host-locksrv$ /sbin/lock_swdmepd -p Port_Num
```

- GULM IP lock server - If you're using a GULM IP lock server, you need to start `lock_gulmd` on the lock server node.

```
host-locksrv$ /sbin/lock_gulmd Cidev
```

- 7 If you're using DMEP hardware, you need to run the `lock_hwdmep_conf` command from one node. The device node is the data pool. Please see the man page for information on additional options.

```
host-a$ lock_hwdmep_conf Device_Node
```

- 8 Mount the GFS file system on all GFS nodes.

- Cluster using DMEP locking. This is the standard Linux mount command. You need to include the `-t gfs` and the option `-o hostdata=Host_IP_Addr`. Please see the man page for information on additional options.

```
host-a$ mount -t gfs /dev/pool/Data_Pool/Mount_Point \
-o hostdata=Host_IP
```

- Cluster using GULM locking. This is the standard Linux mount command. You need to include the `-t gfs` and the option `-o`

hostdata=hostname:cidev. Please see the man page for information on additional options.

```
host-a$ mount -t gfs \  
/dev/pool/Data_Pool/Mount_Point -o \  
hostdata=host-a:/dev/pool/Cidev_Pool
```

Congratulations

Congratulations! GFS should now be running. If you experience problems, please check the troubleshooting information in Chapter 10.

Ready After Boot

There are a few things to be aware of to make mounting GFS partitions at boot time work properly.

- The major concern is that most setups of GFS require a working network. So it is necessary to ensure that networking is initialized before trying to mount GFS. To deal with this correctly, there is a collection of initrc scripts.
- Make sure that all the GFS binaries are installed in **/sbin** and that all the modules are installed correctly.
- Since GFS is journaled, bringing up a system after a crash requires no user intervention. `fsck` is not required so there are no long delays. There will be a short delay as GFS replays the journals.

Use the following basic procedure for mounting GFS:

1 Load the modules

First you will need the modules to be loaded. This can be done by adding `modprobe` lines to one of your `initrc` scripts. However, most distributions should have some mechanism in place to load modules at boot time.

***Note:** If you see a taint message such as the example below, this is not a problem. These messages are for debugging purposes from the kernel to inform you that a non-GPL'ed kernel module has been loaded. The Linux kernel developers use this warning because they do not support/debug a*

kernel that has a non-GPL'ed module loaded (since they don't have access to the source for the module).

```
host-a$ modprobe gfs
Warning: loading /lib/modules/2.4.18/sistina/gfs.o
will taint the kernel: no license.
```

2 Assemble pools after after loading the modules.

3 Mount GFS

If you are using hardware locks or IP locks, ensure that the network is up and working. It is generally a good idea to put your GFS mount points in **/etc/fstab**.

Locate the generic mount script and add the `nogfs` option to the mount command. so that it will not try to mount the GFS file systems. Since the generic mount script usually runs before the networking is set up, and GFS requires active networking, you need to delay things a bit (unless you're using `nolock`).

Below is an example when using DMEP locking:

```
#blockdevice mountpoint fstype options dump pass
/dev/pool/isobel /gfs gfs hostdata=10.1.1.1 0 0
```

In the above example, `10.1.1.1` is the IP address of the node you are editing the `fstab` on. If you do not wish to mount GFS at boot time, add `noauto` to the options field.

Below is an example when using GULM locking:

```
#blockdevice mountpoint fstype options dump pass
/dev/pool/isobel /gfs gfs noauto,hostdata=10.1.1.1 0 0
```

Note: *There is no mention of starting the lock server. It is assumed that the server is always running, and this usually means it is not on a computer running GFS. If it is, make sure to start it before you try to mount any GFS partitions.*

Chapter 5 - Configuration Procedures

Two GFS File Systems with Fibre Channel (FC) and lock_gulmd Lock Server

This procedure describes setting up GFS on a cluster of nodes with FC storage using the GULM lock server. This setup will provide two GFS file systems using one GULM lock server. This procedure also uses a Brocade Fibre Channel switch for its MultiFence method. It is assumed that all nodes are running a GFS-enabled kernel and that GFS (and all its tools) has been installed. The MultiFence agent used in this procedure requires the Perl Telnet module installed on the lock server node.

Hardware Requirement

- One (1) node to be the lock server. Node does not need a Fibre Channel HBA. Hostname and IP address are:
`lockserver 10.0.4.10`
- Four (4) nodes to mount the two GFS file systems. Each node must be equipped with a Fibre Channel HBA. Hostnames and IP addresses are:
`host-a 10.0.4.1`
`host-b 10.0.4.2`
`host-c 10.0.4.3`
`host-d 10.0.4.4`
- Two (2) FC RAID devices, each with two (2) partitions. Each RAID has one very small partition – 4 MB. The second partition on each RAID is the remainder of the device – 99 GB.

- One (1) Brocade Fibre Channel switch with IP address 10.0.4.20.

```
Port 0: host-a
Port 1: host-b
Port 2: host-c
Port 3: host-d
Port 4: FC RAID 1
Port 5: FC RAID 2
```

Setup Procedure

- 1 Load the Fibre Channel driver on nodes a-d.
- 2 Verify that the partition from the RAID devices has been recognized and is accessible.

```
host-a% cat /proc/partitions
..
 8    17        4000         sdb1
 8    18       99000000        sdb2
... \
```

/dev/sdb1 and **/dev/sdb2** are the two partitions from the first RAID device. **/dev/sdc1** and **/dev/sdc2** are the two partitions from the second RAID device. Repeat this verification step on the other hosts. The device nodes assigned to the RAID may differ among nodes.

- 3 Load the kernel modules on nodes a-d.

```
host-a$ modprobe pool
host-a$ modprobe gfs
host-a$ modprobe lock_gulm
```

```
host-b$ modprobe pool
host-b$ modprobe gfs
host-b$ modprobe lock_gulm
```

```
host-c$ modprobe pool
host-c$ modprobe gfs
host-c$ modprobe lock_gulm
```

```
host-d$ modprobe pool
host-d$ modprobe gfs
host-d$ modprobe lock_gulm
```

- 4 Create a pool for the GFS configuration information. On host-a edit a new file named **poolcidev.cf**.

```
host-a$ cat > poolcidev.cf
poolname pool_cidev
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdb1 0
^D
```

- 5 Create pool_cidev.

```
host-a$ pool_tool poolcidev.cf
Pool labels successfully written.
```

- 6 Create a pool for the first file system. On host-a create a file named **pool0.cf**.

```
host-a$ cat > pool0.cf
poolname pool0
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdb2 0
^D
```

- 7 Create pool0.

```
host-a$ pool_tool pool0.cf
Pool labels successfully written.
```

- 8 Create a pool for the second file system. On host-a create a file named **pool1.cf**.

```
host-a$ cat > pool1.cf
poolname pool1
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdc2 0
^D
```

- 9 Create pool0.

```
host-a$ pool_tool pool1.cf
Pool labels successfully written.
```

10 Load the two new pools on all the nodes.

```
host-a$ pool_assemble  
Added pool_cidev.  
Added pool0.  
Added pool1.
```

```
host-b$ pool_assemble  
Added pool_cidev.  
Added pool0.  
Added pool1.
```

```
host-c$ pool_assemble  
Added pool_cidev.  
Added pool0.  
Added pool1.
```

```
host-d$ pool_assemble  
Added pool_cidev.  
Added pool0.  
Added pool1.
```

11 Create the first GFS file system.

```
host-a$ gfs_mkfs -p lock_gulm -t myfs0 -j 4 \  
/dev/pool/pool0  
Device: /dev/pool/pool0  
Blocksize: 4096  
Filesystem Size: ...  
Journals: 4  
Resource Groups: ...  
Locking Protocol: lock_gulm  
Lock Table: myfs0  
Syncing...
```

12 Create the second GFS file system.

```
host-a$ gfs_mkfs -p lock_gulm -t myfs1 -j 4 \  
/dev/pool/pool0  
Device: /dev/pool/pool0  
Blocksize: 4096  
Filesystem Size: ...  
Journals: 4  
Resource Groups: ...  
Locking Protocol: lock_gulm  
Lock Table: myfs1  
Syncing...
```

13 Set up the GULM configuration file. On host-a, edit a new file named **gulm.cf**.

```
host-a$ cat > gulm.cf  
gulmcluster:  
node = lockserver  
port = 40040  
heartbeat_rate = 30  
allowed_misses = 2  
  
locktable:  
name = mylt0  
node = lockserver  
filesystem = myfs0  
port = 42040  
  
locktable:  
name = mylt1  
node = lockserver  
filesystem = myfs1  
port = 42041  
  
node:  
name = lockserver  
ipaddr = 10.0.4.10  
fm = manual 10.0.4.10  
  
node:  
name = host-a  
ipaddr = 10.0.4.1  
fm = brocade 0
```

```
node:
name = host-b
ipaddr = 10.0.4.2
fm = brocade 1

node:
name = host-c
ipaddr = 10.0.4.3
fm = brocade 2

node:
name = host-d
ipaddr = 10.0.4.4
fm = brocade 3

fence:
name = brocade
agent = fence_brcd_port
ipaddr = 10.0.4.20
login = username
passwd = password

fence:
name = manual
agent = fence_manual
^D
```

14 Write the config file to the CIDEV.

```
host-a$ gfs_conf -c -d /dev/pool/pool_cidev \
-f gulm.cf
gulmcluster:
node = lockserver
port = 40040
heartbeat_rate = 30
allowed_misses = 2

locktable:
name = mylt0
node = lockserver
filesystem = myfs0
port = 42040

locktable:
name = mylt1
node = lockserver
```

```
filesystem = myfs1  
port = 42041
```

```
node:  
name = lockserver  
ipaddr = 10.0.4.10  
fm = manual 10.0.4.10
```

```
node:  
name = host-a  
ipaddr = 10.0.4.1  
fm = brocade 0
```

```
node:  
name = host-b  
ipaddr = 10.0.4.2  
fm = brocade 1
```

```
node:  
name = host-c  
ipaddr = 10.0.4.3  
fm = brocade 2
```

```
node:  
name = host-d  
ipaddr = 10.0.4.4  
fm = brocade 3
```

```
fence:  
name = brocade  
agent = fence_brcd_port  
ipaddr = 10.0.4.20  
login = username  
passwd = password
```

```
fence:  
name = manual  
agent = fence_manual
```

- 15** Start the GULM lock server on the node designated for this purpose. Since the GULM lock server is not connected to any of the RAID5 with a FC HBA, you must copy the **gulm.cf** file to the lock server.

***Note:** Make sure that the **gulm.cf** file you copy to the lock server is identical to the **gulm.cf** file written to the CIDEV.*

***Note:** If you should ever make a change to the cluster, be sure to identically update both **gulum.cf** files – the one on the CIDEV and the one on the lock server.*

```
lockserver$ lock_gulmd GULM_Config_File
```

Be sure the **/sbin** directory is in the PATH environment variable.

- 16** Install the GFS file system license. This only needs to be done on a single node in the cluster but it must be done in both file systems. To install the license, both GFS file systems must be mounted.

```
host-a$ mount -t gfs -o \
hostdata=host-a:/dev/pool/pool_cidev \
/dev/pool/pool0 /gfs
host-a$ gfs_tool setlicense \
/gfs0 License_File
```

```
host-a$ mount -t gfs -o \
hostdata=host-a:/dev/pool/pool_cidev \
/dev/pool/pool1 /gfs
host-a$ gfs_tool setlicense \
/gfs1 License_File
```

- 17** Mount the first GFS file system on the remaining nodes (b-d) using **/gfs0** as the mount point.

```
host-b$ mount -t gfs -o hostdata=host-b:/dev/pool/pool_cidev \
/dev/pool/pool0 /gfs0
host-c$ mount -t gfs -o hostdata=host-c:/dev/pool/pool_cidev \
/dev/pool/pool0 /gfs0
host-d$ mount -t gfs -o hostdata=host-d:/dev/pool/pool_cidev \
/dev/pool/pool0 /gfs0
```

- 18** Mount the second GFS file system on the remaining nodes (b-d) using **/gfs1** as the mount point.

```
host-b$ mount -t gfs -o hostdata=host-b:/dev/pool/pool_cidev \
/dev/pool/pool1 /gfs1
host-c$ mount -t gfs -o hostdata=host-c:/dev/pool/pool_cidev \
/dev/pool/pool1 /gfs1
host-d$ mount -t gfs -o hostdata=host-d:/dev/pool/pool_cidev \
/dev/pool/pool1 /gfs1
```

GFS with Fibre Channel and lock_swdmepd Lock Server

This procedure describes setting up GFS on a cluster of nodes with FC storage using the `lock_swdmepd` lock server. This setup also uses a Network Power Switch for its MultiFence method. It is assumed that all nodes are running a GFS-enabled kernel and that GFS and all its tools have been installed. The MultiFence agent used in this example requires the Perl Telnet module installed on all nodes.

Hardware Requirement

- One (1) node to be the lock server. Hostname and IP address are:
`lockserver 10.0.4.10`
- Four (4) nodes to mount the GFS file system. Each has a FC card. Host names and IP addresses are:
`host-a 10.0.4.1`
`host-b 10.0.4.2`
`host-c 10.0.4.3`
`host-d 10.0.4.4`
- One (1) FC RAID device with two (2) partitions. The first partition is very small – 4 MB. The second partition is the remainder of the device – 99 GB.
- One (1) FC hub or switch.
- One (1) Network Power Switch from WTI. IP address is 10.0.4.20. The nodes are plugged into the NPS as follows:
`Plug 1: host-a`
`Plug 2: host-b`
`Plug 3: host-c`
`Plug 4: host-d`

Setup Procedure

- 1 Load the FC driver on nodes a - d.
- 2 Verify that the two partitions from the RAID device have been recognized and are accessible.

```
host-a% cat /proc/partitions
```

```
..  
8 17 4000 sdb1  
8 18 99000000 sdb2  
..
```

/dev/sdb1 and **/dev/sdb2** are the two partitions from the RAID device. Repeat this verification step on the other hosts. The device nodes assigned to the RAID may differ among nodes.

- 3** Load the kernel modules on nodes a - d.

```
host-a$ modprobe pool
host-a$ modprobe lock_dmeq
host-a$ modprobe gfs
```

```
host-b$ modprobe pool
host-b$ modprobe lock_dmeq
host-b$ modprobe gfs
```

```
host-c$ modprobe pool
host-c$ modprobe lock_dmeq
host-c$ modprobe gfs
```

```
host-c$ modprobe pool
host-c$ modprobe lock_dmeq
host-c$ modprobe gfs
```

- 4** Create a pool for the file system. On host-a, edit a new file named **pool0.cf**.

```
host-a$ cat > pool0.cf
poolname pool0
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdb2 0
^D
```

- 5** Create **pool0**.

```
host-a$ pool_tool pool0.cf
Pool labels successfully written.
```

- 6** Create a second pool for the GFS configuration information. On host-a, edit a new file named **pool0.cidev.cf**.

```
host-a$ cat > pool0.cidev.cf
poolname pool0_cidev
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdb1 0
```

- 7** Create **pool0_cidev**.

```
host-a$ pool_tool pool0cidev.cf
Pool labels successfully written.
```

8 Load the two new pools on all the nodes.

```
host-a$ pool_assemble  
Added pool0.  
Added pool0_cidev.
```

```
host-b$ pool_assemble  
Added pool0.  
Added pool0_cidev.
```

```
host-c$ pool_assemble  
Added pool0.  
Added pool0_cidev.
```

```
host-d$ pool_assemble  
Added pool0.  
Added pool0_cidev.
```

9 Create the file system.

```
host-a$ gfs_mkfs -p lock_dmep -t \  
/dev/pool/pool0_cidev -j 8 /dev/pool/pool0  
Device: /dev/pool/pool0  
Blocksize: 4096  
Filesystem Size: ...  
Journals: 8  
Resource Groups: ...  
Locking Protocol: lock_gulm  
Lock Table: /dev/pool/pool0_cidev  
Syncing...
```

10 Set up the GFS CIDEV. On host-a, create a file named **gfscf.cf**.

```
host-a$ cat > gfscf.cf
gfscparam:
lmtype = dmep
cmtype = dmep
datadev = /dev/pool/pool0

lmparam:
cbport = 3001
server = 10.0.4.10
serverport = 15697

cmparam:
node timeout = 30

fence:
name = nps
agent = fence_wti_nps
ipaddr = 10.0.4.20
passwd = Password

node:
name = host-a
ipaddr = 10.0.4.1
cid = 0
fm = nps 1

node:
name = host-b
ipaddr = 10.0.4.2
cid = 1
fm = nps 2

node:
name = host-a
ipaddr = 10.0.4.3
cid = 2
fm = nps 3

node:
name = host-a
ipaddr = 10.0.4.4
cid = 3
fm = nps 4
^D
```

11 Write the GFS configuration to the CIDEV.

```
host-a$ gfs_conf -c -d /dev/pool/pool0_cidev \  
-f gfscf.cf  
gfsparam:  
lmtype = dmep  
cmtype = dmep  
datadev = /dev/pool/pool0  
  
lmparam:  
cbport = 3001  
server = 10.0.4.10  
serverport = 15697  
  
cmparam:  
node timeout = 30  
  
fence:  
name = nps  
agent = fence_wti_nps  
ipaddr = 10.0.4.20  
passwd = Password  
  
node:  
name = host-a  
ipaddr = 10.0.4.1  
cid = 0  
fm = nps 1  
  
node:  
name = host-b  
ipaddr = 10.0.4.2  
cid = 1  
fm = nps 2  
  
node:  
name = host-a  
ipaddr = 10.0.4.3  
cid = 2  
fm = nps 3  
  
node:  
name = host-a  
ipaddr = 10.0.4.4  
cid = 3  
fm = nps 4
```

12 Start the fence daemon on each host.

Note: The MultiFence agent `fence_wti_nps` must be in root's path on each host.

```
host-a$ fenced
host-b$ fenced
host-c$ fenced
host-d$ fenced
```

- 13** Start the `lock_swdmepd` lock server on the node designated for this purpose.

```
lockserver$ lock_swdmepd
```

- 14** Install the GFS file system license. This only needs to be done on a single node in the cluster. The GFS file system must be mounted to install the license.

```
host-a$ mount -t gfs /dev/pool/pool0 /gfs -o \
hostdata=10.0.4.1
```

```
host-a$ gfs_tool setlicense /gfs License_File
```

- 15** Mount the GFS file system on nodes (b-d) using `/gfs` as the mount point.

```
host-b$ mount -t gfs /dev/pool/pool0 /gfs -o \
hostdata=10.0.4.2
```

```
host-c$ mount -t gfs /dev/pool/pool0 /gfs -o \
hostdata=10.0.4.3
```

```
host-d$ mount -t gfs /dev/pool/pool0 /gfs -o \
hostdata=10.0.4.4
```

GFS with Fibre Channel and gulm Lock Server

This procedure describes setting up GFS on a cluster of nodes with FC storage using the `gulm` lock server. This setup also uses a Network Power Switch for its MultiFence method. It is assumed that all nodes are running a GFS-enabled kernel and that GFS along with all its tools have been installed. The MultiFence agent used in this example requires the Perl Telnet module to be installed on the lock server node.

Hardware Requirement

- One (1) node to be the lock server. This node should be equipped with a Fibre Channel HBA. Hostname and IP address are:
- Four nodes to mount the GFS file system, each node equipped with a Fibre Channel HBA. Host names and IP addresses are:

```
lockserver 10.0.4.10
```

```
host-a 10.0.4.1  
host-b 10.0.4.2  
host-c 10.0.4.3  
host-d 10.0.4.4
```

- One (1) FC RAID device with two (2) partitions. The first partition is very small – 4 MB. The second partition is the remainder of the device – 99 GB.
- One (1) Fibre Channel hub or switch.
- One (1) Network Power Switch (NPS) from WTI. IP address is 10.0.4.20. The nodes are plugged into the NPS as follows:

```
Plug 1: hostA  
Plug 2: hostB  
Plug 3: hostC  
Plug 4: hostD  
Plug 5: lockserver
```

Setup Procedure

- 1 Load the Fibre Channel driver on the lockserver and nodes a-d.
- 2 Verify that the partition from the RAID device has been recognized and is accessible.

```

host-a% cat /proc/partitions
..
 8      17      4000      sdb1
 8      18     99000000   sdb2
... \

```

/dev/sdb1 and **/dev/sdb2** are the two partitions from the RAID device. Repeat this verification step on the other hosts. The device numbers assigned to the RAID may differ among nodes.

- 3 Load the kernel modules on nodes a-d.

```

host-a$ modprobe pool
host-a$ modprobe gfs
host-a$ modprobe lock_gulm

```

```

host-b$ modprobe pool
host-b$ modprobe gfs
host-b$ modprobe lock_gulm

```

```

host-c$ modprobe pool
host-c$ modprobe gfs
host-c$ modprobe lock_gulm

```

```

host-d$ modprobe pool
host-d$ modprobe gfs
host-d$ modprobe lock_gulm

```

- 4 Create a pool for the file system. On host-a create a file named **pool0.cf**.

```

host-a$ cat > pool0.cf
poolname pool0
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdb2 0
^D

```

- 5 Create pool0.

```

host-a$ pool_tool pool0.cf
Pool labels successfully written.

```

- 6 Create a second pool for the GFS configuration information. On host-a edit a new file named **pool0cidev.cf**.

```
host-a$ cat > pool0cidev.cf
poolname pool0_cidev
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdb1 0
^D
```

- 7 Create pool0_cidev.

```
host-a$ pool_tool pool0cidev.cf
Pool labels successfully written.
```

- 8 Load the two new pools on all the nodes.

```
host-a$ pool_assemble
Added pool0.
Added pool0_cidev.
```

```
host-b$ pool_assemble
Added pool0.
Added pool0_cidev.
```

```
host-c$ pool_assemble
Added pool0.
Added pool0_cidev.
```

```
host-d$ pool_assemble
Added pool0.
Added pool0_cidev.
```

- 9 Create the file system.

```
host-a$ gfs_mkfs -p lock_gulm -t myfs -j 4 \
/dev/pool/pool0
Device: /dev/pool/pool0
Blocksize: 4096
Filesystem Size: ...
Journals: 4
Resource Groups: ...
Locking Protocol: lock_gulm
Lock Table: myfs
Syncing...
```

10 Set up the gulm configuration file.

```
host-a$ cat > gulm.cf
gulmcluster:
node = lockserver
port = 40040
heartbeat_rate = 30
allowed_misses = 2

locktable:
name = mylt
node = lockserver
filesystem = myfs
port = 42040

node:
name = lockserver
ipaddr = 10.0.4.10
fm = nps 5

node:
name = host-a
ipaddr = 10.0.4.1
fm = nps 1

node:
name = host-b
ipaddr = 10.0.4.2
fm = nps 2

node:
name = host-c
ipaddr = 10.0.4.3
fm = nps 3

node:
name = host-d
ipaddr = 10.0.4.4
fm = nps 4

fence:
name = nps
agent = fence_wti_nps
ipaddr = 10.0.4.20
passwd = Password
^D
```

11 Write the config file to the CIDEV.

```
host-a$ gfs_conf -c -d /dev/pool/pool0_cidev \  
-f gulm.cf  
gulmcluster:  
node = lockserver  
port = 40040  
heartbeat_rate = 30  
allowed_misses = 2  
  
locktable:  
name = mylt  
node = lockserver  
filesystem = myfs  
port = 42040  
  
node:  
name = lockserver  
ipaddr = 10.0.4.10  
fm = nps 5  
  
node:  
name = host-a  
ipaddr = 10.0.4.1  
fm = nps 1  
  
node:  
name = host-b  
ipaddr = 10.0.4.2  
fm = nps 2  
  
node:  
name = host-c  
ipaddr = 10.0.4.3  
fm = nps 3  
  
node:  
name = host-d  
ipaddr = 10.0.4.4  
fm = nps 4  
  
fence:  
name = nps  
agent = fence_wti_nps  
ipaddr = 10.0.4.20  
passwd = Password
```

- 12** Start the gulm lock server on the node designated for this purpose.

```
lockserver$ lock_gulmd /dev/pool/pool0_cidev
```

Be sure the `/sbin` directory is in the `PATH` variable.

- 13** Install the GFS file system license. This only needs to be done on a single node in the cluster. In order to install the license the GFS file system needs to be mounted.

```
host-a$ mount -t gfs -o \  
hostdata=host-a:/dev/pool/pool0_cidev \  
/dev/pool/pool0 /gfs  
host-a$ gfs_tool setlicense /gfs License_File
```

- 14** Mount the GFS file system on the remaining nodes (b-d) using `/gfs` as the mount point.

```
host-b$ mount -t gfs -o \  
hostdata=host-b:/dev/pool/pool0_cidev \  
/dev/pool/pool0 /gfs  
  
host-c$ mount -t gfs -o \  
hostdata=host-c:/dev/pool/pool0_cidev \  
/dev/pool/pool0 /gfs  
  
host-d$ mount -t gfs \  
-o hostdata=host-d:/dev/pool/pool0_cidev \  
/dev/pool/pool0 /gfs
```

GFS with Parallel SCSI and lock_gulmd Lock Server

This procedure describes setting up GFS on a cluster of nodes with shared parallel SCSI storage using the lock_gulmd lock server. The setup uses a Network Power Switch for its MultiFence method. It is assumed that all nodes are running a GFS-enabled kernel and that GFS (with all the tools) has been installed.

Note: Multi-port SCSI RAIDs are probably your best bet, but if you are going to share a bus, make sure that there are no conflicting LUNs.

The MultiFence agent used in this example requires the Perl Telnet module installed on the lock server node.

Hardware Requirement

- One (1) node to be the lock server. Hostname and IP address are:

```
lockserver 10.0.4.10
```
- Two (2) nodes to mount the GFS file system. Each has a SCSI HBA. Host names and IP addresses are:

```
host-a 10.0.4.1  
host-b 10.0.4.2
```
- One (1) SCSI RAID device with two partitions. The first partition is very small – 4 MB. The second partition is the rest of the device – 99 GB.
- One (1) Network Power Switch from WTI with the IP address 10.0.4.20. The host nodes are plugged into the NPS as follows:

```
Plug 1: lockserver  
Plug 2: host-a  
Plug 3: host-b
```

Setup Procedure

- 1 Verify that the two partitions from the RAID device have been found.

```
host-a% cat /proc/partitions
...
 8    17    4000    sdb1
 8    18 99000000    sdb2
...
```

/dev/sdb1 and **/dev/sdb2** are the two partitions from the RAID device. Repeat this verification step on the other nodes. The device nodes assigned to the RAID may differ among nodes.

- 2** Load the kernel modules on hosts a and b.

```

host-a$ modprobe pool
host-a$ modprobe lock_gulm
host-a$ modprobe gfs

host-b$ modprobe pool
host-b$ modprobe lock_gulm
host-b$ modprobe gfs

```

- 3** Create a pool for the file system. On host-a create a file named **pool0.cf**.

```

host-a$ cat > pool0.cf
poolname pool0
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdb2 0
^D

```

- 4** Create pool0.

```

host-a$ pool_tool pool0.cf
Pool labels successfully written.

```

- 5** Create a second pool for the GFS configuration information. On host-a create a file named **pool0cidev.cf**.

```

host-a$ cat > pool0cidev.cf
poolname pool0_cidev
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sdb1 0
^D

```

- 6** Create **pool0_cidev**.

```

host-a$ pool_tool pool0cidev.cf
Pool labels successfully written.

```

- 7** Load the two new pools on all the nodes.

```

host-a$ pool_assemble
Added pool0.
Added pool0_cidev.

host-b$ pool_assemble
Added pool0.
Added pool0_cidev.

```

8 Create the file system.

```
host-a$ gfs_mkfs -j 2 -p lock_gulm -t gfsscsi \  
/dev/pool/pool0  
Device: /dev/pool/pool0  
Blocksize: 4096  
Filesystem Size: ...  
Journals: 2  
Resource Groups: ...  
Locking Protocol: lock_gulm  
Lock Table: gfsscsi  
Syncing...
```

- 9 Set up the GFS configuration device. On host-a create a file named **gfscf.cf**.

```
host-a$ cat > gfscf.cf
gulumcluster:
node = lockserver
port = 40040
heartbeat_rate = 30
allowed_misses = 2
verbose = 227

locktable:
name = scsilt
node = lockserver
filesystem = gfsscsi
port = 42040
verbose = 227

node:
name = lockserver
ipaddr = 10.0.4.10
fm = nps 1

node:
name = host-a
ipaddr = 10.0.4.1
fm = nps 2

node:
name = host-b
ipaddr = 10.0.4.2
fm = nps 3

fence:
name = nps
agent = fence_wti_nps
ipaddr = 10.0.4.20
passwd = Password
^D
```

10 Write the GFS configuration to the CIDEV.

```

host-a$ gfs_conf -c gfscf.cf
gulmcluster:
node = lockserver
port = 40040
heartbeat_rate = 30
allowed_misses = 2

locktable:
name = scsilt
node = lockserver
filesystem = gfsscsi
port = 42040
verbose = 227

node:
name = lockserver
ipaddr = 10.0.4.10
fm = nps 1

node:
name = host-a
ipaddr = 10.0.4.1
fm = nps 2

node:
name = host-b
ipaddr = 10.0.4.2
fm = nps 3

fence:
name = nps
agent = fence_wti_nps
ipaddr = 10.0.4.20
passwd = Password

```

11 Copy the **gfscf.cf** file to the lock server node **/etc/gfs/gfscf.cf**.

```

host-a$ gfs_conf -c -d /dev/pool/pool0_cidev -f \
gfscf.cf

```

***Note:** The MultiFence agent `fence_wti_nps` must be in root's path on the lockserver node.*

12 Start the `lock_gulmd` lock server on the node designated for this purpose.

```

lockserver$ lock_gulmd /etc/gfs/gfscf.cf

```

- 13** Install the GFS file system license. This only needs to be done on a single node in the cluster. To install the license, the GFS file system must be mounted.

```
host-a$ mount -t gfs /dev/pool/pool0 /gfs \  
-o hostdata=host-a:/dev/pool/pool0_cidev  
host-a$ gfs_tool setlicense /gfs License_File
```

- 14** Mount the GFS file system on host-b using `gfs` as the mount point.

```
host-b$ mount -t gfs /dev/pool/pool0 /gfs \  
-o hostdata=host-b:/dev/pool/pool0_cidev
```

GFS with GNBD

This procedure is for getting a basic GNBD GFS configuration running with three nodes – one as the server and two for clients. Before using this procedure, please make sure that you have successfully built and installed GFS on all of the hosts which are to be members of the GFS cluster.

Note: You can obtain significant performance benefits by partitioning your block devices and running pool on top of GNBD, so this configuration procedure assumes that you want to use pool on top of GNBD.

Hardware Requirement

- One (1) node which will be the GNBD server *and* the GFS IP-based lock server (running `lock_gulmd`). This node has a 32GB SCSI device labeled **/dev/sdb** which will be used for GFS data.
- Normally you would want to run `lock_gulmd` on a node that is neither a GNBD server or one of the GFS clients. The main reasons for this are better performance and increased reliability. Unfortunately, since only three (3) hosts are used in this example, it is necessary to double up functionality on one of the nodes.

`host-a 192.168.1.2`

- Two (2) GNBD client nodes which will mount the GFS file system.

`host-b 192.168.1.3`

`host-c 192.168.1.4`

- One (1) 32GB SCSI device attached to host-a and recognized as **sdb**. This drive has five partitions; one is about four megabytes and the other four are equally sized and use the rest of the drive space.
 - `sdb1` = 4MB
 - `sdb2`, `sdb3`, `sdb4` and `sdb5` = 7.9 GB

The small partition will be used for the CIDEV. It can be larger, but 4 MB is all that is needed. The other four partitions must be of equal size, so that pool can stripe across them.

Setup Procedure

- 1 On host-a (the GNBD server), export the five `sdb` partitions as five GNBD's. The global names of the GNBD's are **gfscfdata** and **gfsdata[1-4]**. The first will be used for the CIDEV and the rest will be used for the GFS file system.

```
host-a$ modprobe gnbd_serv
host-a$ gnbd_export -e gfscfdata -d /dev/sdb1
host-a$ gnbd_export -e gfsdata1 -d /dev/sdb2
host-a$ gnbd_export -e gfsdata2 -d /dev/sdb3
host-a$ gnbd_export -e gfsdata3 -d /dev/sdb4
host-a$ gnbd_export -e gfsdata4 -d /dev/sdb5
```

- 2 On host-b and host-c (the GNBD clients), import the five GNBDs. The `gnbd_import` program will import all GNBDs being exported from host-a.

```
host-b$ modprobe gnbd
host-b$ gnbd_import -i host-a
```

```
host-c$ modprobe gnbd
host-c$ gnbd_import -i host-a
```

The five GNBDs should now be accessible as:

```
/dev/gnbd/gfscfdata /dev/gnbd/gfsdata1
/dev/gnbd/gfsdata2 /dev/gnbd/gfsdata3
/dev/gnbd/gfsdata4
```

At this point, setting up Pool and GFS on these shared devices is no different than the other configuration procedures.

- 3 Load kernel modules on GFS gnbd client nodes host-b and host-c.

```
host-b$ modprobe pool
host-b$ modprobe lock_gulm
host-b$ modprobe gfs
```

```
host-c$ modprobe pool
host-c$ modprobe lock_gulm
host-c$ modprobe gfs
```

- 4 Create a pool for the file system. On host-b create a file named **pool0.cf**.

```
host-b$ cat > pool0.cf
poolname pool0
subpools 1
subpool 0 128 4 gfs_data
pooldevice 0 0 /dev/gnbd/gfsdata1 0
pooldevice 0 1 /dev/gnbd/gfsdata2 0
pooldevice 0 2 /dev/gnbd/gfsdata3 0
pooldevice 0 3 /dev/gnbd/gfsdata4 0
^D
```

- 5 Create pool0.

```
host-b$ pool_tool pool0.cf
Pool labels successfully written.
```

- 6 Load the pool on all nodes.

```
host-b$ pool_assemble
Added pool0.
host-c$ pool_assemble
Added pool0.
```

At this point, a pool device exists that is striped across the four gnbd partitions. This allows gnbd to deal with multiple requests at once, significantly increasing performance

- 7 Create the file system.

```
host-b$ gfs_mkfs -p lock_gulm -t gfsgnbd \
-j 2 /dev/pool/pool0
Device: /dev/pool/pool0
Blocksize: 4096
Filesystem Size: ...
Journals: 2
Resource Groups: ...
Locking Protocol: lock_gulm
Lock Table: gfsgnbd
Syncing...
```

8 Set up the GFS CIDEV. On host-b edit a new file named **gfscf.cf**.

```
host-b$ cat > gfscf.cf
gumcluster:
node      = host-a
port      = 40040
heartbeat_rate = 30
allowed_misses = 2
verbose   = 227

locktable:
name      = gnbdlr
node      = host-a
filesystem = gfsgnbd
port      = 42040
verbose   = 227

node:
name = host-a
ipaddr = 192.168.1.2
fm = gnbd 192.168.1.2

node:
name = host-b
ipaddr = 192.168.1.3
fm = gnbd 192.168.1.3

node:
name = host-c
ipaddr = 192.168.1.4
fm = gnbd 192.168.1.4

fence:
name = gnbd
agent = fence_gnbd
^D
```

9 Write the GFS configuration to the CIDEV.

```

host-b$ gfs_conf -c -d /dev/pool/gfscfdata \
-f gfscf.cf
gulmcluster:
node = host-a
port = 40040
heartbeat_rate = 30
allowed_misses = 2
verbose = 227

locktable:
name = gnbdlr
node = host-a
filesystem = gfsgnbd
port = 42040
verbose = 227

node:
name = host-a
ipaddr = 192.168.1.2
fm = gnbd 192.168.1.2

node:
name = host-b
ipaddr = 192.168.1.3
fm = gnbd 192.168.1.3

node:
name = host-c
ipaddr = 192.168.1.4
fm = gnbd 192.168.1.4

fence:
name = gnbd
agent = fence_gnbd

```

10 Place a copy of the **gfscf.cf** configuration file on host-a in **/etc/gfs/gfscf.cf**.

```

host-a$ gfs_conf -c -d /dev/gnbd/gfscfdata -f \
gfscf.cf

```

11 Start the **lock_gulmd** lock server on the node designated for this purpose.

```

host-a$ lock_gulmd /etc/gfs/gfscf.cf

```

- 12** Install the GFS file system license. This only needs to be done on a single node in the cluster. To install the license, the GFS file system must be mounted.

```
host-b$ mount -t gfs /dev/pool/pool0 /gfs \  
-o hostdata=host-b:/dev/gnbd/gfscfdata  
host-b$ gfs_tool setlicense /gfs License_File
```

- 13** Mount the GFS file system on host-c using `gfs` as the mount point.

```
host-c$ mount -t gfs /dev/pool/pool0 /gfs \  
-o hostdata=host-c:/dev/gnbd/gfscfdata
```

GFS as a Local File System

This procedure is for single client use of GFS. Doing this lets you have a journaled file system on a local node. You only need a single node with either an unused disk drive or a free partition (`/dev/hda4` in this example). Installing GFS in this manner restricts you to using it on a single node.

- 1 Load the kernel modules on the node.

```
host-a$ modprobe pool
host-a$ modprobe lock_nolock
host-a$ modprobe gfs
```

- 2 Make the file system.

```
host-a$ gfs_mkfs /dev/hda4 -p lock_nolock -j 1
```

The `-j` option puts a single journal onto this file system. Do not use any other value here, as this setup cannot do more than one host. If needed, you can use the `-J` option to adjust the size of the journal. See the man page for details.

- 3 Mount GFS. This is the standard mount command:

```
host-a$ mount -t gfs /dev/hda4 /Mount_Point
```

- 4 Install the GFS file system license.

```
host-a$ gfs_tool setlicense /Mount_Point \
License_File
```

GFS on a Server's Root File System

Customers who wish to use GFS on a server's root file system should contact Sistina support for further information.

Chapter 6 - Advanced Configuration Procedures

Adding New Storage Hardware

The first step is to install the new storage hardware. Once installed, the hardware must be accessible and recognized by all nodes in the GFS cluster. For more information on this process, see “Preparing Devices” on page 24. You may be using the new devices to expand (grow) an existing GFS file system or to create an entirely new GFS file system. This section describes the more difficult operation – expanding the file system.

Online GFS Resizing (with Pool)

***Caution:** We strongly advise you to create a backup of the target file system before using the following procedure. If this procedure is done incorrectly, file system corruption could occur.*

The `gfs_grow` utility performs online expansion of a GFS file system. To do online expansion when GFS is built on pool:

- 1 Determine which pool GFS is built on using `df`:

```
host-a% df
Filesystem      1k-blocks      Used    Available   Use%  Mounted on
/dev/hda1        7076124    1460872    5255800    22%   /
/dev/pool/pool0 71925076         0    71925076    0%   /gfs
```

In this example, GFS is mounted on `/dev/pool/pool0`. Therefore, the pool in this case is `pool0`.

- 2 Create a new pool configuration file that includes the new storage. It must contain all the prior configuration information of the pool that is being targeted for expansion. The only additions/differences that are permitted are the number of subpools and extra entries for subpool and pool device additions.

Use `pool_info` to list the current configuration and write it to a new configuration file. Then you can make appropriate modifications to the file.

```

host-a$ pool_info -p pool0
poolname    pool0
subpools    1
subpool     0      0      1      gfs_data
pooldevice  0      0      /dev/sde2  0
    
```

You can redirect the output from this command to a file and use it as the basis of the new pool configuration file.

- 3** As an example, you wish to add four storage devices to the GFS file system: `/dev/sda`, `/dev/sdb`, `/dev/sdc` and `/dev/sdd`. You also want to place the devices in a single, striped subpool.

You would do this by editing the file obtained in Step 2 with the following changes.

***Note:** Line numbers in the following example equate to explanations below. Do not actually add line numbers in your pool configuration files.*

```

1. poolname    pool0
2. subpools    2
3. subpool     0      0      1      gfs_data
4. subpool     1     128    4      gfs_data
5. pooldevice  0      0      /dev/sde2  0
6. pooldevice  1      0      /dev/sda   0
7. pooldevice  1      1      /dev/sdb   0
8. pooldevice  1      2      /dev/sdc   0
9. pooldevice  1      3      /dev/sdd   0
    
```

- Line 2 is changed to reflect two subpools.
 - Line 4 was added to describe the characteristics of the new subpool.
 - Lines 6-9 were added to describe the characteristics of the new storage devices.

The updated file is then saved as **big_pool**.

- 4** Use `pool_grow` to write the expanded labels.

```

host-a$ pool_grow big_pool
Warning! You are about to write new labels for
pool0.
Are you certain you want to continue? [yN] y
    
```

- 5 The device on which the GFS file system exists has now been expanded to include the new storage. Execute `dmesg` to see that the kernel has been notified of the change.

```
host-a% dmesg
...
Target pool found :: attempting to grow
Capacity before: 257038336
Capacity after: 401430528
```

- 6 The final step is to expand (grow) the GFS file system itself. This is accomplished by executing `gfs_grow /gfs`.

```
host-a$ gfs_grow /gfs
FS: Mount Point: /gfs
FS: Device: /dev/pool/pool0
FS: Options: rw
FS: Size: 18049024
DEV: Size: 53744128
Preparing to write new FS information...
Done.
```

- 7 Run `df` to see the results of the newly resized file system.

```
host-a% df
Filesystem      1k-blocks    Used    Available  Use %  Mounted on
/dev/hda1        7076124    1460880    5255792    22%    /
/dev/pool/pool0 214696452      0    214696452    0%    /gfs
```

Caveats:

- The maximum size of a Linux file system is 2 terabytes.
- This utility can only resize mounted file systems. If the file system is not mounted, it cannot access required internal file system information.
- If *every* block on the file system is used and the block containing the resource index is full (so that no more index entries can be added), then it is not possible to expand the file system. This is because the resource index requires a free block in the existing file system in order to store information about the location of the new resource groups.

Caution: *Running multiple `gfs_grow` processes simultaneously, either on different nodes or on the same node, is not supported and may result in file system corruption. Although this result is unlikely, do not do this.*

Adding a GFS File System to an Existing Configuration Using DMEP

To add a new GFS file system:

- 1 Create a data pool. See “Creating Pools” on page 25.
- 2 Create a CIDEV pool. See “Cluster Information Device and gfs_conf” on page 27.

- 3 Write the data and CIDEV pools.

```
host-a$ pool_tool Data_Pool_File  
host-a$ pool_tool Cidev_Pool_File
```

- 4 Load the new pools on all the nodes.

```
host-a$ pool_assemble
```

- 5 Create a new GFS configuration file. You can obtain a copy of the current file and modify it as needed.

```
host-a$ gfs_conf -p -d Cidev_Partition > newconfig.cf
```

- Change `datadev:` to match the new `data_pool` name.
- If the `lock_dmep` lock server is being used, modify the port for the `lockdev:` unless you're using a different DMEP lock server.
- If a hardware DMEP lock server is being used, it is most likely the same as the GFS data device.
- `cbport:` needs to be different for each GFS file system.

- 6 Write the new GFS configuration file to the CIDEV.

```
host-a$ gfs_conf -c -d Cidev -f newconfig.cf
```

- 7 Create the GFS file system (see the man page for additional information).

- 8 If using `lock_dmep` server, start `lock_swdmepd` with the `-p` option to specify a different port.

```
host-a$ lock_swdmepd -p Port
```

- 9 Mount the new file system on each node.

Setting up Failover for the lock_swdmepd Server

Customers who wish to implement failover for the lock_swdmepd lock server should contact Sistina support for further information.

Adding a Node to a Cluster

To add a new node to a GFS configuration, it must be added to the GFS cluster configuration file. From one of the current GFS nodes:

- 1 Obtain a copy of the GFS configuration file.

```
host-a$ gfs_conf -p -d Cidev_Partition > newconfig.cf
```

- 2 Add the required lines for the new node(s) to newconfig.cf.

Note: This example presumes DMEP locking. When using DMEP locking, each node must be identified by a unique CID number. Remember that increasing CID numbers is required; do not go back and reuse CID's of a node which have been removed.

- 3 Append the new node(s) to the active configuration.

```
host-a$ gfs_conf -a -d Cidev_Partition -f newconfig.cf
```

- 4 Verify that the GFS file system(s) to be mounted by the additional node(s) have an appropriate number of journals.

```
host-a$ gfs_tool df GFS_Mount_Point
```

- 5 To add more journals, see “Adding Journals to a Cluster” on page 78.

On the new node(s):

- 1 Prepare the node. Follow the steps in “Installing GFS” on page 15.
- 2 Load all necessary drivers and GFS modules. See “Preparing Devices” on page 24.
- 3 Assemble required pool(s) using `pool_assemble`.
- 4 Start the MultiFence daemon `fenced` (if required).
- 5 Mount the GFS file system(s).
- 6 Perform per node customizing as required.

Removing a Node from a Cluster

Warning: Only remove node(s) from a cluster configuration that has been unmounted cleanly. Changing the CIDEV contents when the cluster is not in a consistent state can cause problems.

- 1 Before removing a node from the cluster configuration, unmount the GFS file system from the node being removed.
- 2 Verify the IP address of the node to be removed from the cluster.
`host-a$ gfs_conf -p -d Cidev_Partition`
- 3 Remove the node with the given IP address.
`host-a$ gfs_conf -r IP_Address -d Cidev_Partition`
- 4 Remove the GFS pools on the node.
`host-a$ pool_assemble -r Pool_Partition`
- 5 Remove any node/local customizations that were made for GFS as required.

Adding Journals to a Cluster

- 1 If separate GFS journal subpools are not being used, skip to Step 4.
- 2 If GFS journal subpools are being used, determine if there are unallocated journals. If additional journals were not allocated when the GFS file system was created, it may not be possible to add additional nodes to the file system. To determine if GFS journals are being used with a particular GFS file system/pool combination, run the following command and look for `gfs_journal` in the output.
`host-a$ pool_info -p Pool_Partition`
- 3 If there are no spare GFS journal subpools but an allocated partition exists, it can be added to the pool as a GFS journal subpool. To do so, run the following commands with the GFS file system unmounted and with pool deactivated.

Caution: As in the case of any operation that changes on-disk structures, it is highly advisable to have good backups of the file system before proceeding.

- Unmount the GFS file system that will be affected.

```
host-a$ umount GFS_Mount_Point
```

- Obtain a copy of the current pool configuration.

```
host-a$ pool_info -p Pool_Partition > newpool.cf
```

- Edit **newpool.cf** adding the new GFS journal. Be sure to make appropriate modifications to the subpools, subpool and pooldevice sections.
- Write the pool file to disk.

```
host-a$ pool_tool newpool.cf
```

- Assemble the new pool(s) on all nodes.

```
host-a$ pool_assemble -r Pool_Partition
```

```
host-a$ pool_assemble Pool_Partition
```

- Mount the GFS file system(s) on all nodes.

4 Add additional journals.

```
host-a$ gfs_jadd -j Number_of_New_Journals \  
GFS_Mount_Point
```

5 Verify that the journals were added.

```
host-a$ gfs_tool -tv GFS_Mount_Point
```

Updating the GFS License

GFS now requires you to install a license file to allow reading and writing of files. A license should have been provided with the GFS binaries. To install your license file:

- 1 Verify the validity of the license via:

```
host-a$ gfs_tool checklicense License_File
```

- 2 Mount the GFS file system the license is for on a single node using the appropriate mount options for that file system. This step is only required if you are upgrading an existing GFS file system. When creating a new GFS file system, see “Installing the GFS License” on page 21.

- 3 Install the license into the GFS file system using:

```
host-a$ gfs_tool setlicense GFS_Mount_Point License_File
```

This command only needs to be run once (on a single node) for each GFS file system.

- 4 Mount the GFS file system on all remaining nodes that will be accessing it. A valid GFS evaluation license looks like:

```
# GFS license file
LicenseVersion = "1"
GFSVersion = "5.1"
LicenseNumber = "1234-123-131"
Name = "Very Very Big ISP, Inc."
StartDate = "Wed Oct 21 12:19:30 2002 CDT"
StopDate = "Fri Oct 23 12:19:30 2002 CDT"
StartSec = "1023297570"
StopSec = "1025889570"
NodeCount = "4"
Hash = "DEADFAD1-FEEDBEEF"
```

A fully-enabled GFS license looks like:

```
# GFS license file
LicenseVersion = "1"
GFSVersion = "5.1"
LicenseNumber = "1234-123-131"
Name = "Very Very Big ISP, Inc."
NodeCount = "8"
Hash = "BADCAD5-BADADD8"
```

- 5 Validate the license using the following command:

```
host-a$ gfs_tool checklicense License_File
```

Using Context Dependent Path Names

Context Dependent Path Names (CDPN) make sharing GFS between different nodes, node types and even operating systems much easier. It allows you to create links that point to different locations depending on the context. This is especially nice when using GFS for a shared root file system.

CDPN Expansion Strings

GFS does CDPN expansion for the following strings:

- @hostname** The value substituted for the link corresponds to the output of `uname -n`.
- @mach** The value substituted for the link corresponds to the output of `uname -m`.
- @os** The value substituted for the link corresponds to the output of `uname -s`.

Hostname Differentiation

In this example, there are three computers – larry, moe and curly. Each computer needs specific configuration files in the `/etc` directory. To enable this with CDPN:

- 1 Create three directories corresponding to the host names of the nodes:

```
host-a$ mkdir larry
host-a$ mkdir moe
host-a$ mkdir curly
```

- 2 Create the `/etc` directory in each node directory.

```
host-a$ mkdir larry/etc
host-a$ mkdir moe/etc
host-a$ mkdir curly/etc
```

- 3 Populate the `etc` directories.

```
host-a$ cp -r /etc/* larry/etc/
host-a$ cp -r /etc/* moe/etc/
host-a$ cp -r /etc/* curly/etc/
```

- 4 Edit the various files with your favorite editor (such as `vim`) as appropriate.

- 5 Create the CDPN.

```
host-a$ ln -s @hostname/etc etc
```

Now when you are logged in on larry, the `etc` symlink points to `larry/etc`¹. When you are on curly, the `etc` symlink points to `curly/etc`. When you are on moe, the `etc` symlink points to `moe/etc`.

Node Type Differentiation

In this example there are two nodes: jim and bob. `uname -m` prints out alpha for bob. The goal is to provide separation of the `/bin` and `/sbin` directories for each architecture type. To enable this with CDPN:

- 1 Create two directories corresponding to the output of `uname -m` for each box.

```
host-a$ mkdir i386
host-a$ mkdir alpha
```

¹An `ls -l` of the `etc` symlink will show `@hostname/etc` if it is set up correctly.

- 2 Create bin and sbin directories in each.

```
host-a$ mkdir i386/bin; mkdir i386/sbin  
host-a$ mkdir alpha/bin; mkdir alpha/sbin
```

- 3 Populate the directories with the appropriate binaries

- 4 Create the CDPN.

```
host-a$ ln -s @mach/bin bin  
host-a$ ln -s @mach/sbin sbin
```

Now when you are logged in on jim, the bin and sbin symlinks point to **i386/bin** and **i386/sbin**, respectively. When you are logged in on bob, the bin and sbin symlinks point to **alpha/bin** and **alpha/sbin**, respectively.

Pool Multipathing

Dynamic Multipathing is not enabled by default in the Pool module when multiple I/O paths are detected. You can enable this capability with two classes of service: failover and round-robin load sharing.

Failover Load Sharing

```
host-a% insmod pool mp_type=1
```

Round-Robin Load Sharing

```
host-a% insmod pool mp_type=2 mp_stripe=64
```

Valid values for mp_type are:

- 0 The default. Dynamic multipathing is not enabled.
- 1 Failover load sharing.
- 2 Round-robin load sharing.

Specify mp_stripe in kilobytes. Values less than 64 are not permitted. For more details, see “Dynamic Multipathing” on page 87.

Enabling Large File Support

GFS supports files larger than 2GB by supporting the Large File Summit (LFS) specification. By default, code compiled on a system with LFS support is compiled with 32-bit file offsets and 32-bit file system calls – so that existing code will continue to work as it did before LFS support.

To take advantage of GFS large file support, your applications need to enable 64-bit file offsets. There are two ways to do this:

- `_LARGEFILE64_SOURCE`

Applications which define this before the inclusion of any headers will get additional syscalls.

If this symbol is defined, the standard UNIX headers will define an additional set of 64-bit data types and syscalls.

The new types and functions include the following:

```
off64_t, struct stat64, struct flock64
```

```
open64() mmap64() setrlimit64() getrlimit64() truncate64()  
ftruncate64() stat64() lstat64() fstat64() statfs64() fstatfs64()  
getdents64()
```

These interfaces function identically to their 64-bit counterparts except that file offset arguments are expanded to 64 bits.

The normal 32-bit versions of these functions will still be available unless they are mapped to the 64-bit versions with `_LARGEFILE_SOURCE` or `_FILE_OFFSET_BITS`.

- `_FILE_OFFSET_BITS=64`

If this preprocessor symbol is defined, then the `off_t` data type will be 64 bits wide and the file related syscalls (and their related structures) will be the 64-bit versions.

***Note:** Applications built with these flags against non-LFS versions of glibc WILL use the 64 bit interfaces when dynamically linked against the new glibc.*

For more information see the Large File Summit specification at <http://www.sas.com/standards/large.file/>

Also see Scyld's FTP site <ftp://ftp.scyld.com/pub/lfs/>

Chapter 7 - Performance

This chapter describes some fine tuning you can do to mount options, pools and the GFS file system to improve performance.

Mount Options

All mount options are invoked via the `mount` command or the options field in `/etc/fstab`:

```
host-a$ mount /dev/sda? /mnt -o foo,bar,foobar=XX
```

```
# Device          Mount Point  FS Type          Options  Freq  PassNo
/dev/pool/pool_gfs /scratch    gfs              foo,bar,foobar=XX  1      2
```

noatime

Do not update the file access time (`atime`) when reading from a file. This option is useful on file systems where performance is more critical than updating the file access time (which is rarely important). Applications that do make use of `atime` are HSMs (Hierarchical Storage Managers) and mail user agents.

nodiratime

Do not update the file access time for directories when reading/traversing them. This option is useful on file systems where performance is more critical than updating the directory access time.

direct

All I/O to files in the GFS file system mounted with this option shall be accomplished via Direct I/O, even if the `O_DIRECT` flag is not specified on `open`.

***Caution:** Take care when using this option to ensure I/O operations are FS-aligned. If non-FS-aligned I/O is requested, it will fail with -EINVAL.*

This option can provide significant speed improvement for applications that are capable of Direct I/O such as databases and high performance scientific codes.

ignore_local_fs

By default, when `no_lock` `lock` module is used, it automatically turns on the **localcaching** and **localflocks** flags. This option forces GFS to treat the file system as a multihost file system.

localcaching

This flag notifies GFS that it is running as a local file system so that it can enable optimizations that may not be turned on when it is running as a cluster file system. These optimizations are automatically turned on when the `lock_nolock` module is loaded.

localflocks

This flag tells GFS to let the VFS layer do all **flock** and **fcntl** file locking. This is turned on automatically by the `lock_nolock` module.

Pools

Layout

Subpools

Internally, a pool is divided into subpools. Subpools serve two purposes:

- Subpools divide disks into separate stripe groups. Overall bandwidth and parallelism of the pool may be enhanced by using subpools spread across a number of storage devices.
- Subpools can also be used for GFS journals.

Journals

Subpools can be used to isolate, on different devices, file system journals from regular file system data. This is done by assigning each subpool a type of **gfs_journal** or **gfs_data**.

At least one **gfs_data** subpool must exist within a pool, and you must create one **gfs_journal** subpool for *each* GFS host which will have the GFS volume mounted.

***Note:** It is recommended that you allocate extra **gfs_journal** subpools to allow for the easy addition of new GFS nodes in the future. `gfs_grow` allows journals to be dynamically added if there is available space in the file system.*

When using **gfs_journal** subpools, you must use `gfs_mkfs` without the `-j` option, which specifies the number of journals and instructs `gfs_mkfs` to ignore subpool types.

Journal subpools are entirely optional but can provide additional performance in cases where there is a high number of metadata updates in the GFS file system and/or when they are placed on high-performance storage (for example, persistent SSD storage).

If you don't assign journals to separate subpools, they will be allocated from the single data subpool where the GFS file system resides.

Dynamic Multipathing

Pool supports two forms of dynamic multipathing – failover and round-robin.

Failover

Failover provides the ability to use multiple I/O paths to access storage where only a single I/O path is used until a failure occurs.

A common scenario would be a multi-port storage device – two controllers – where both ports are plugged into a storage switch. Pool would detect both paths to the shared storage when pools are assembled. It would then choose a single path to use to access the device. This path would stay active until Pool detects it is no longer valid. At that point, it would *failover* from the original path to the other valid path. I/O operations would continue on the new path and the old path would be marked as unavailable.

Pool does not currently support ordering of I/O paths for failover.

Round-Robin

Round-robin allows load sharing between multiple I/O paths to the shared storage. As in the case of failover, Pool detects the multiple I/O paths when pools are assembled. Unlike failover, it will mark all paths as active and attempt to load-share across them.

An example of round-robin load sharing is a multi-port storage device – two controllers – where both ports are active and plugged into a storage switch.

- Pool detects the two paths and attempts to load-share I/O requests over both active paths in a round-robin manner. In this case I/O requests alternate between the two active paths.
- Since the round-robin is performed on an I/O request basis, large I/O requests end up using a single path for the request unless you tell Pool otherwise. You do this by providing Pool with a request stripe size to be used as a basis of determining when large I/O requests should be shared across multiple paths.
- The default stripe size is 64KB. This means that until an I/O request exceeds 64KB, it will be transferred on a single I/O path. When the request exceeds the stripe size, it will then be split across the available I/O paths in stripe size chunks.
- An example of this would be an I/O write request of 256KB. 256 is larger than 64 so the I/O request will be spread across the available I/O paths. Since in this scenario there are only two active I/O paths, 2 x 64 KB will be handled via one path and 2 x 64 KB via the other.

Round-robin load-sharing is limited to four active paths currently. This is not an architectural limit. It can be raised by special request.

Currently Pool Dynamic Multipathing does not *failback* an I/O path. Failback is when an I/O path fails and is marked as unavailable. At a later point, the I/O path returns as a valid I/O path that could be used for I/O requests.

At this point, Pool could mark it as active and utilize it for either failover or round-robin load-sharing. Unfortunately, in order to determine that the I/O path is active and valid once again, it must be probed. Probing can take considerable time and result in severe (crippling) degradation of the SAN. The only way to address this issue is to have HBA vendors reduce their timeout values. If and when progress is made in this effort, failback capabilities may be added to Pool.

To enable multipathing please see “Pool Multipathing” on page 82

GFS File System Tunables

You can tune the following parameters on a per GFS file system basis using `gfs_tool settune`:

<code>atime_quantum</code>	Sets the quantum period that will expire before atime updates are guaranteed to be performed. You can use this option to change the quantum that atime updates take to occur rather than completely disabling them via the noatime option. The default value is 60 seconds.
<code>quota_quantum</code>	Sets the maximum time a quota update can remain cached on a node before being synced to the quota file. The default is 60 seconds.
<code>quota_scale</code>	Controls the frequency of quota file sync increases as the user approaches their limit. The more frequent the syncs, the more accurate the quota enforcement. Unfortunately, that increases the contention between nodes for the quota file. Set the value for <code>quota_scale</code> using <code>gfs_tool</code> . <ul style="list-style-type: none">• The default value is one (1). This sets the maximum theoretical quota overrun (with infinite node with infinite bandwidth) to twice the user's limit. In practice, the maximum overrun should be much less.• A <code>quota_scale</code> number greater than one (1) makes quota syncs more frequent and reduces the maximum overrun.• Numbers less than one (but greater than zero) make quota syncs less frequent.
<code>quota_enforce</code>	Provides enforcement of quotas for a GFS file system. You can turn off a quota by setting its value to zero (0). GFS will continue to keep track of each ID's usage, but it will not enforce quota limits. Set the <code>quota_account</code> parameter to zero to completely disable quotas.

Chapter 8 - Fibre Channel Hardware

This chapter describes configuring hardware that might be used with GFS. Most of the discussion focuses on Fibre Channel (FC) since that topic may cause some confusion. For definitions of terms specific to Fibre Channel, see the Glossary.

Fibre Channel Background Information

Merging Channel and Network Features

Traditionally, accessing data has been divided into two categories, the channel and the network.

- *Channels* usually deal with well-structured, closed and fixed environments. Usually there is one host system and several attached devices, like a master-slave setup. Communication is usually restricted to be between the master and one of the attached slaves. The design of the channel often put priority in moving large sets of data quickly using the entire channel.
- In the *network* model, the environment is unstructured, open and behaves much more unpredictably. All devices can talk to any other device. The focus when designing is often on flexibility – being able to deliver smaller packets of data to any destination, even on a faulty network. One consequence of this is that software must ensure correct behavior regarding, for example, connections, access permissions and route information.

Channels often are characterized by high throughput and low overhead, while networks show low throughput and high overhead. However, channels are much less flexible than networks and run for shorter distances. One objective of Fibre Channel is to combine the best features of both channels and networks.

Sharing Devices

The main reason for using Fibre Channel is to be able to share storage devices among several computers. Fibre Channel may also be used as a regular IP network or as a fast SCSI-bus for a single computer, but the main reason for investing in FC equipment is the desire to share storage devices.

With a Fibre Channel setup, a new topology emerges unlike the traditional computer-storage device topology. There is no computer that *owns* the storage device – it is equally shared by all the computers connected to the FC network.

One obvious requirement is to keep the file system(s) on these devices synchronized. There are two ways of doing this.

- One method is for ALL the nodes to mount the file system(s) read-only. This means that the file system will never change, hence it is never out of sync with any of the computers. With this method, any regular file system can be used on the devices. It is important to remember that *all* the computers must mount read-only. Even with just one writer, there will be problems. Since regular file systems assume that the disk never changes under them, they use caching mechanisms intensively in order to increase performance. In a regular single-node setup this is not a problem, but it does not work in a multicomputer setup.
- A better option is to use a shared disk file system on the shared devices. This is a file system with built-in mechanisms for ensuring synchronization between multiple computers. One such file system is Sistina's Global File System.

Using Existing Protocols for Mapping Data

One of the features of Fibre Channel is to use existing, well-established protocols for transportation of data. Fibre Channel does not impose any new format for data, hence all existing applications can continue to use the standard protocols. Supported protocols include SCSI, IP and HIPPI.

Cabling

Fibre Channel can use both optical and copper cabling. Optical cables can be either multimode or singlemode fiber. Singlemode allows much greater distances than multimode. Copper cables are less expensive than optical cables but they do not allow the distances provided by optical cables.

Fibre Channel Topologies

Fibre Channel offers three different topologies that can be combined to fit any needed configuration.

- The simplest topology is a point-to-point connection between two nodes.
- A more advanced topology uses an Arbitrated Loop. This configuration allows up to 126 devices to share one loop.
- Finally, Fibre Channel switches are available that can form fabrics with thousands of nodes.

Point-to-Point

This was the first and most simple Fibre Channel topology. It connects two nodes, be it two computers or one computer and a storage device.

- No need for arbitration.
- No addressing problems – the package you receive must come from the other side.
- The devices have access to the full bandwidth of the link. However, constant access to the full FC bandwidth is usually not needed. This kind of topology may be useful for connecting two nodes that are physically separated by some distance.

Fabrics

Switches were introduced to be able to connect several devices together. The idea was to have several point-to-point connections, each point-to-point connection connecting one node to the switch. This way several nodes would be able to connect together in a larger topology. The topology that is made when using one or more switches is called a fabric.

Arbitrated Loops

Arbitrated Loops were added on to the original FC-specifications. It was realized that there would be a need for a topology between the point-to-point and the switched topologies. The Arbitrated Loop combines some of the features from the Point-To-Point with some of the features from the fabric.

One of the main arguments against the pure fabric is the price per fabric port. This can be too high if each fabric port is connected to just one node.

Although this configuration offers each device full connection to the fabric, most nodes do not constantly need the full bandwidth of Fibre Channel. The

Arbitrated Loop offers (as the name suggests) a loop topology. The loop can contain up to 126 NL-ports and one fabric-port with all the nodes arbitrating for usage of the loop.

The arbitrated loop is made up by having all nodes connect their outgoing connector to the incoming connector of the downstream node. This way, all nodes on the loop act as repeaters for all the frames running around the loop.

At any given time, only one port can be sending frames. (One exception is, if both ports want full-duplex support, they may agree to set up such a connection and hence fully utilize the connection in both directions.) An arbitration procedure selects the Loop Master, who thereafter owns the loop. Since any the nodes on the loop must win an arbitration before being allowed to use the loop, performance declines as more nodes are added. A loop with 127 active ports is likely to not operate with the performance one would like.

Private Loops

A loop that consists of up to 126 NL ports is considered to be a *private loop*. These nodes can access all other nodes on this loop, but cannot (by using Fibre Channel) access any other nodes.

Public Loops

When the FL-port of a switch is connected to the loop, the loop becomes a *public loop*. Assuming there are more nodes connected to the switch, the nodes on the loop may access and be accessed by these nodes. By connecting several public loops to a fabric a huge number of nodes may be interconnected. Each of the loop nodes arbitrate for the full use of the bandwidth of that loop. When a FL-port is part of a loop, arbitration is no longer totally fair, the FL-port will have higher priority than other ports on the loop.

To communicate with nodes outside the local loop, a node must support public loops. However, some switches provide this functionality for nodes that do not support public loops. See “Switches” on page 95.

Fibre Channel Switches and Hubs

Multiple nodes and storage are connected together using Fibre Channel switches or hubs. Switches allow a larger and more efficient topology than hubs. Hubs are cheaper and can be used in smaller topologies. Switches and hubs can also be combined.

Hubs

Hubs are used to connect nodes and storage devices into an Arbitrated Loop. Although the cable layout will look like a star topology, the actual Fibre Channel layout is a loop. A drawback of loops is that all devices on the loop have to share the bandwidth. With a hub, no more than 126 devices can be connected.

Switches

Fibre Channel switches allow large Fibre Channel topologies. Switches have different multiple ports (starting with 8 or 16). Switch ports may be connected to another switch, directly to a node or storage device, or to a hub. By combining several switches, large topologies with thousands of devices can be configured.

On an Arbitrated Loop, every new device must send inquiry requests around the loop in order to determine the size and topology of the loop. When using a switch, this is handled by the switch. Any new device must only log into the switch. The device can then query the switch for information regarding the topology and other devices in the topology.

Storage Devices

This section examines some of the problems one might experience with storage devices when using them as shared block devices for GFS.

Disks and JBODs

The smallest physical unit is the hard drive itself. A Fibre Channel hard drive is usually a SCSI disk with a Fibre Channel interface.

Instead of an independent single hard drive, the hard drive may be assembled in an enclosure. In the case of JBODs (Just a Bunch of Disks), all the hard drives are accessible as individual drives to the GFS nodes.

Pool may be used to stripe data across the disks in order to increase bandwidth.

A potential problem with using JBODs is overloading the hard drives. Many hard drives are not intended to be accessed by multiple nodes at the same time. This may lead to lower performance or SCSI errors. In order to prevent problems, the hardware should be stress tested before installing GFS.

RAID Controllers

RAID controllers form an interface to the individual hard drives. Instead of having the individual hard drive directly accessible by the GFS nodes, all I/O traffic goes through the controller.

Note: GFS only works with RAID controllers that sit on the storage. Using one RAID controller in each of the GFS nodes will not work. The reason is that the individual RAID controllers will not be synchronized when writing parity blocks.

Many RAID controllers can export one or more groups consisting of one or more disks. Each of these disk aggregations would have a different Logical Unit Number (LUN). Some RAID controllers are also able to set up access tables where certain nodes have access to certain groups of hard drives and not others. There may also be limits to the number of nodes that can access the controller.

When using a RAID controller with these capabilities, you must configure the controller to allow all the GFS nodes to access the hard drive group(s) you intend to use as storage for the GFS file system.

Configuring a RAID controller is usually done with some form of serial cable connected to the RAID. Consult the manual for your controller for its capabilities, configuration options and configuration method.

In the case of a RAID controller that is set to export multiple groups of hard drives, the different groups will be seen as multiple LUNs by the hosts. The node may have to be configured to be able to recognize multiple LUNs.

Some RAID controllers make the controller itself available on one of the LUNs. This allows configuring the controller through the SCSI/FC interface. However, this sometimes causes problems when Linux nodes scan the device for LUNs. In case of problems consult the manual for how to disable this feature, or how to map the controller to the highest LUN.

Some RAID controllers may be overloaded when accessed by multiple nodes. This may lead to lower performance or access problems for the nodes. In order to prevent problems the hardware should be stress tested before installing GFS.

FC Under Linux

This section focuses on FC under Linux.

Installing FC Driver as a Module

Unless there is a need to boot from or to have a FC-device accessible at boot time, there are some advantages in having the Fibre Channel driver installed as a module and not compiled into the kernel.

- Some drivers occasionally will not see all (or any) of the FC devices. Reloading the driver may fix this. When loading a driver, inspect **/proc/scsi/scsi** or **/proc/partitions** to verify that all the storage units/partitions you expected are listed. (This assumes knowledge about how many storage units/partitions should be there.)
- Most drivers do not allow a static mapping between a FC device and a **/dev/sd*** entry. By using the driver as a module, there is more control over what **/dev/sd*** device corresponds to what FC drive.
- When installing additional FC devices, not all drivers notice the new devices when they are connected. Even if the driver notices the new device, the SCSI sublayer does not get updated about new devices. Reloading the driver is a convenient way of forcing the SCSI subsystem to notice new devices. The same is to a certain degree true when removing a device – reloading the driver ensures that all layers are aware that a device is gone.

Increasing the Device Number Limit

Linux enforces a limit to how many devices can be connected to a node at run-time. The limit is the **SD_EXTRA_DEVS** value, located in the file **drivers/scsi/hosts**. If there is a need to connect more devices at run-time, increase the default and recompile the kernel.

Hot Adding Storage Units

With FC, it is possible to add more storage units to the FC network without having to power anything down. The requirement is a spare port on either a hub or a switch for connecting the new storage unit. Connecting a new storage device can lead to certain problems, particularly errors reported as SCSI errors. In order to reduce the likelihood of problems, you should try to avoid having traffic on the FC-network when inserting new storage units. The most convenient way of making new storage units available is to reload the FC driver.

- 1 Stop all I/O to the FC subsystem.
- 2 Unmount any file system mounted on the FC subsystem.
- 3 Execute:
`rmmod Name_of_FC_Driver_Module`
- 4 Connect any new storage units.
- 5 Execute:
`modprobe Name_of_FC_Driver_Module`

If reloading the FC driver is not an option, a forced rescan of the SCSI bus is needed.

- 1 Connect the new device to the FC network.
The driver should give a message about receiving a LIP, a Notification Change or something like that.
- 2 Check `/proc/scsi/scsi` to make sure the new drives are seen by the SCSI layer.
- 3 If the drives do NOT show up in `/proc/scsi/scsi`, you need to manually add them to the SCSI layer.

- 4 Execute:
`echo "scsi add-single-device A B C D" > \
/proc/scsi/scsi`

where A = Host ID, B = Channel ID, C = Device ID and D = LUN ID.

For example:

```
echo "scsi add-single-device 1 0 15 0" > \  
/proc/scsi/scsi
```

for LUN 0 on device 15 on channel 0 on the second adapter.

- 5 The device should now show up in `proc/scsi/scsi`.

Hot Removing Storage Units

With FC, it is also possible to remove an attached storage device without affecting other parts of the setup.

Warning: *You must logically remove the SCSI subsystem before you physically disconnect it. Otherwise applications will still expect to be able to access the device and this risks confusing the SCSI layer and potentially bringing down the computer. Be sure to do the following steps in the sequence shown.*

- 1 Stop all applications accessing the device to be removed, close all files on the device, and so on.
- 2 Unmount any file system(s) on the device.
- 3 Unload the pool on the device, using:

```
pool_assemble -r Name_of_Pool
```
- 4 In `/proc/scsi/scsi`, find the device-ID, hostbus-ID and channel-ID of the device.
- 5 Execute this command

```
echo "scsi remove-single-device A B C D" > \  
/proc/scsi/scsi
```

where A = Host ID, B =Channel ID, C = Device ID and D = LUN ID.
This removes the device from the SCSI layers and ensures that the device will not be referenced again.
- 6 Inspect `/proc/scsi/scsi` to make sure the device is no longer listed.
- 7 Now you may physically disconnect the device.

HBAs Used by Sistina Under Linux

Emulex 8000

Emulex has several Fibre Channel HBAs. Currently the LightPulse family is supported under Linux.

Firmware

Make sure that you have the latest firmware for your cards. Firmware is available from Emulex FTP server. In order to download the firmware to the HBA there is a DOS tool as well as a Linux-tool.

BIOS Settings

Drivers

Emulex provides the Linux driver for the Emulex HBAs. This driver is available at the Emulex FTP server <ftp://ftp.emulex.com/>. This FTP location also includes drivers for both IP and SCSI as well as a diagnostic utility. This driver is only supported on x386 platform nodes.

QLogic

The QLogic 2100, 2200 and 2300 families are all supported under Linux. However, the 2100 family is retired and no longer supported by QLogic. Newer drivers may not support this HBA.

BIOS Settings

By entering the BIOS of the HBA during boot up, it is possible to:

- Set the Loop ID of the HBA. (In accordance with FC Specifications, this is the address the HBA now will attempt to get. In the case of address conflicts, the HBA may get another Loop ID.
- Specify the frame size (packet size) to be used.
- Scan the Fibre Channel network. This allows you to check that the HBA actually can see all the devices it is supposed to see. Testing this saves time and helps you discover if, for example, you forgot a terminator or improperly connected a cable.

Drivers

There are currently three sources of drivers for the QLogic HBAs:

- QLogic has written its own driver. The source is available as beta-drivers at the QLogic homepage <http://www.QLogic.com> under **Drivers / Software**. The standard driver from the QLogic webpage runs only on Intel boxes. QLogic does not support the Alpha platform. This driver supports Arbitrated Loops as well as Fabric networks. The different topologies are automatically detected; no configuration is needed.
- Matthew Jacob (mjacob@feral.com) of Feral Software (<http://www.feral.com>) has written a driver for the QLogic HBAs. This driver supports both FreeBSD and Linux.
- The University of New Hampshire's driver is included in a standard Linux kernel.

Compaq

No information available at this time.

Other FC HBAs

Interphase

There are Linux drivers available for the Interphase FC HBA. Sistina has not tested these HBAs. Interphase has written and released their own driver. The source for this driver is available at <http://www.iphase.com>.

There is also a driver from the University of New Hampshire This Interphase driver has been included in the Linux kernel as of v2.2.11. The most recent version of the driver can be obtained at <http://www.iol.unh.edu/consortiums/index.html>. This driver provides both IP and SCSI services.

JNI

There are Linux drivers available for JNI FC HBA. Sistina has not tested these HBAs. JNI has written their own driver for this card. The source as well as other information is available at <http://www.jni.com>.

FC Under FreeBSD

Currently, a QLogic HBA is the only HBA with support under FreeBSD. There is one driver for this HBA, written by Matthew Jacob at Feral Software. Matthew's email address is mjacob@feral.com. Feral Software is at <http://www.feral.com>. This driver is written to support both FreeBSD and Linux. Limited testing by Sistina shows that this driver works for both FC-AL as well as with fabric topologies built around Brocade FC switches.



Chapter 9 - Troubleshooting

This chapter presents:

- Sistina contact information
- How to request technical support
- Solutions to common problems.

Contacting Sistina

For Support

If you already have a support contract, please contact Sistina support directly:

- **Email** - Please send a question or a description of your problem to support@sistina.com. For the speediest service, include the information detailed in “Requesting Technical Support” on page 104.
- **Phone** - Support personnel can be reached at 1-866-SISTINA (1-866-747-8462) during the hours of 07:00-19:00 CT.
- **Web** - Online support is available at Sistina Support http://www.sistina.com/products_cs.htm

If you are a new customer, information on support services can be found at:

- **Email** info@sistina.com
- **Phone** 612-638-0500
- **Web** Support for Sistina's Customers http://www.sistina.com/products_cs.htm

For General Information

- **Snailmail**
Sistina Software
1313 5th Street SE, Suite 111
Minneapolis, MN 55414
- **Phone** 612-638-0500
- **Fax** 612-379-3952
- **Email** info@sistina.com

White Papers on Our Website

- Infinitely Smarter Storage
<https://www.sistina.com>
- Sistina's Global File System
http://www.sistina.com/products_gfs.htm
- Sistina's GFS Publications
http://www.sistina.com/products_GFS_publications.htm
- Large File Summit (LFS) Specification
<http://www.sas.com/standards/large.file/>
- Fibre Channel Overview
<http://www.fibrechannel.com/technology/>
- SCSI Quick Start Guide
http://www.scsifaq.org/scsi_quick_start.html
- The Official SCSI FAQ
<http://www.scsifaq.org/scsifaq.html>

Requesting Technical Support

In order to best assist you, Sistina needs information about your configuration and the contents of certain files. Please collect as much of the following information as possible.

For GFS errors, please provide:

- 1 Attach a short description of your hardware, including types of nodes and disks, disk interface (SCSI, FC, NBD), and any other information about your hardware you feel is important.

- 2 Include the config file you used to define your pools. You can find out exactly what the system thinks you have by running `pool_info` with the `-p` option:

```
pool_info -p Name_of_Pool
```

- 3 If you are not using pool, describe the kind of block device you are putting GFS onto.
- 4 Include the config file being used by GFS. Obtain this by running:

```
gfs_conf -p -d Cidev
```
- 5 The command line used to make the file system.
- 6 The command line used to mount the file system.

When GFS trips a panic trap, please provide:

- 1 All of the information requested in two sections above.
- 2 The debug dumps for each node that had GFS mounted from the same set of disks. If a node is still running, this can be accomplished by running the following command on that host:

```
gfs_tool dbdump GFS_Mount_Point
```

Save the output of this command to a file. If the node crashed, try to get as much as you can. The best way to do this is to have messages sent to a serial port with another node collecting data from that serial port.

Include the node's name in the filename of these dumps (for example, **somenode.gfsdbdump**).

This can be a lot of information. If you end up with more than a couple of files, tar and gzip them into a single archive. Submit this compressed archive file by email to **support@sistina.com** with a short description of the error.

Solutions to Common Problems

Recovering a Node from MultiFence

WTI Network Power Switch

The WTI Network Power Switch should require no intervention. The node gets rebooted while another node plays the journal to ensure no data corruption or loss occurs. When it comes back up, the GFS should be in the normal state after a reboot.

APC MasterSwitch

The APC MasterSwitch, like WTI NPS, requires no intervention.

Brocade FC Switch

The Brocade Fibre Channel switch method, `fence_brcd_port`, uses the `portdisable` command. To bring a node back into the cluster after MultiFence by this method, you'll need to:

- 1 Put the node into a *good* state. The easiest way to do this is to shut down the node.
- 2 Reenable the port on the Brocade switch. If you aren't sure which port the node is on, you can run `gfs_conf -p -d Cidev` to see which port is assigned to that node.

```
telnet Brocadeiname_or_IP
login: Admin
password: Password
portshow Port_Number # to verify this is the
disabled portport
enable Port_Number # to enable the port
portshow Port_Number # to verify the port has been
enabled
exit
```

- 3 Bring the node up.

GNBD Network Block Device

Manual fencing requires human intervention both to MultiFence and to recover. To recover:

- 1 The node which begins the MultiFence procedure will print a message to syslog of the form: **fence_manual: Node 10.0.4.108 requires hard reset.**
Run `fence_ack_manual` after power cycling the node.
- 2 The operator must first manually reset the specified node or manually disconnect it from the storage devices.
- 3 On the node where the syslog message appeared, run:

```
fence_ack_manual -s 10.0.4.108
```

Warning: *If you run `fence_ack_manual` before or without the manual MultiFence, unrecoverable file system corruption may occur.*

lock_swdmepd server goes down

If the lock_swdmepd server goes down for any reason, all I/O to the GFS file system(s) stops. This is to ensure data integrity since none of the nodes can get locks. To recover, you need to reboot all the nodes.

The order of the boot is lock_swdmepd server first. Make sure it comes up cleanly and that the lock_swdmepd is running. Then bring up all the nodes. The first node up should read all the journals. Once that has occurred GFS should mount cleanly on all nodes.

File system won't mount

```
mount: wrong fs type, bad option, bad superblock on  
Pool_Name, or too many mounted file systems
```

This usually means there is an error in the GFS configuration file or the mount command. Check the order of the mount command and the IP addresses in the GFS configuration file. One of the node IP addresses listed in the GFS configuration file must match the IP address used for `hostdata=IP addr` in the mount line.

Cannot see disk devices

There could be a number of causes:

- Make sure the HBA driver is loaded. You can use `lsmod` to check. For more information, see “Preparing Devices” on page 24.
- If the driver is loaded and you still can't see the devices, check the connections between the node and the device. Check to make sure the GBIC is seated completely both in the Fibre Channel switch and in the HBA (if the HBA has one).
- Make sure the Fibre Channel switch is on and configured correctly.
- If you are using zones on the Fibre Channel switch, make sure the disk and node are in a zone.
- Make sure the disk device is on.
- If you can't see the disks from any node and you've tried all the above, make sure the driver is built correctly.

Pools not loading

Make sure you can see the disk devices with `cat /proc/partitions`. If you cannot, see the previous troubleshooting entry. Try loading them

manually and check the error. There may be a mistake in the pool configuration files.

GFS doesn't always mount on startup

For GFS to mount, several things must occur. Review the **dmesg** and look for problems with the HBA loading, loading the pools or loading the GFS modules. Try to determine where the problem is and why it is occurring.

Node is MultiFenced even when there is no problem on the node

MultiFence occurs when a node isn't communicating with the locking device within a certain timeout period. If you are using a lock_dmeop server and you have a large amount of lock traffic, you may want to increase the timeout period in the GFS configuration file. For more information, see “Cluster Information Device and gfs_conf” on page 27. Another option is to move the lock traffic to a different network to reduce IP traffic.

Hardware

HBA driver

If you only see some of the disks when the HBA driver loads, it may be that with certain HBA drivers, the LIPs on the switch cause unrecoverable interrupts. The easiest solution for this is to use Fibre Channel switch zoning to keep the LIPs from other nodes from affecting each other.

Compiling driver

See “HBAs Used by Sistina Under Linux” on page 99.

Loading driver

See “Preparing Devices” on page 24.

Finding storage devices

- Is the storage device powered on?
- Are the FC cables correctly connected? From storage to switch/hub and from switch/hub to node?
- Is the storage correctly terminated? Check the manual for the device. Some devices requires an external terminator, other have internal jumpers that need to be set correctly.

- If you are using a switch or managed hub, is there some setting that prevents the port with the node to see the port with the storage? See “Switches” on page 95 for more details.
- Does the storage require some configuration before a new node can connect to it?
- If there are no obvious problems with the setup, step back and make the setup as simple as possible. This means connecting the storage directly to the node. No switch, no managed hubs.

Finding a storage device from multiple nodes

- Make sure that at least one node can find the storage device. See “Finding storage devices” above.
- Make sure that each of the nodes can find the storage device if this is done one node at a time.
- If this fails, you may need to configure the storage device for each node that will access it. Check to see if Sistina has previous experience with this storage device. If not, look in the documentation for the device. This is usually referred to as *host mapping*, and the configuration process will likely ask you for the WWN of the node that is to be connected.

Appendix A - Command/Module Summary

Functional Area	Name	Synopsis
GFS Kernel Modules	gfs.o	Kernel module that implements the GFS file system.
	gnbd.o	Kernel module that implements GNBD server system.
	lock_gulm.o	Kernel module to communicate with the GULM lock server.
	gnbd_serv.o	Kernel module to implement the GNBD client.
	lock_harness.o	Kernel module that interfaces between the GFS kernel module and locking subsystems.
	lock_stats.o	Kernel module that implements lock statistic gathering.
	lock_dmep.o	Kernel module to communicate with DMEP lock servers.
	lock_nolock.o	Kernel module that implements the <i>no lock</i> protocol for GFS.
	pool.o	Kernel module that implements the GFS cluster volume manager.
fence.o	Kernel module that provides communication between the GFS and MultiFence agents.	

GFS File System Commands	<code>gfs_conf</code>	Configure/modify/delete information on the CIDEV.
	<code>gfs_fsck</code>	Perform GFS file system consistency checks and repairs.
	<code>gfs_tool</code>	Enable/tune advanced features for GFS file systems.
	<code>gfs_jadd</code>	Add journals to an existing GFS file system.
	<code>gfs_grow</code>	Grow (expand) the capacity of a GFS file system.
	<code>gfs_quota</code>	Administrative interface for GFS disk quotas.
	<code>gfs_mkfs</code>	Create a GFS file system.
GNBD Commands	<code>gnbd_import</code>	Administrative interface for a GNBD client.
	<code>gnbd_export</code>	Administrative interface for a GNBD server.
GFS Lock/Server Commands	<code>lock_swdmep_initds</code>	Disk store initializer for lock_swdmepd (HA config).
	<code>lock_swdmepd</code>	Software-based DMEP lock server daemon.
	<code>lock_gulmd</code>	Grand Unified Lock Manager daemon.
	<code>lock_gulm_Core</code>	Subcomponent of the lock_gulmd server.
	<code>lock_gulm_LT</code>	Subcomponent of the lock_gulmd server.
Port Commands	<code>pool_assemble</code>	Create/delete pools.
	<code>pool_grow</code>	Grow (expand) existing pool(s).
	<code>pool_info</code>	Provide information about existing pool(s).
	<code>pool_tool</code>	Initialize on-disk pool structures.
MultiFence Commands	<code>fence_apc_ms</code>	MultiFence agent for the APD MasterSwitch NPS.
	<code>fence_brcd_port</code>	MultiFence agent for the Brocade Silkworm series.
	<code>fence_gnbd</code>	MultiFence agent for use with GNBD.
	<code>fence_wti_nps</code>	MultiFence agent for the Western Telematic NPS.
	<code>fenced</code>	MultiFence daemon.
	<code>fence_manual</code>	MultiFence agent that is based on operator intervention.

Glossary

General Storage Network Terms

Term	Meaning
1 MHz	A frequency of one million Hertz (cycles per second).
1 Mflop/s	A computational rate of one million floating-point operations per second.
1 Gflop/s	A computational rate of one billion floating-point operations per second.
1 Tflop/s	A computational rate of one trillion floating-point operations per second.
1 Kbyte	2^{10} bytes of data.
1 Mbyte	2^{20} bytes of data.
1 Gbyte	2^{30} bytes of data.
1 Tbyte	2^{40} bytes of data.
1 Mbyte/s	A transfer rate of 2^{20} bytes of data per second.
1 Gbyte/s	A transfer rate of 2^{30} bytes of data per second.
1 Tbyte/s	A transfer rate of 2^{40} bytes of data per second.
arbitrate	Process of selecting one L Port from a collection of several ports that concurrently request use of the arbitrated loop.
arbitrated loop	A loop type topology where two or more ports can be interconnected, but only two ports at a time can communicate.
CDSL	Context Dependent Symbolic Links

Term	Meaning
CIDEV	Cluster Information Device. Each GFS file system requires a companion CIDEV which defines parameters associated with the file system. The CIDEV must be a globally accessible shared device just as the file system device is shared.
DMEP	Device Memory Export Protocol
F Port	A port in a fabric where an N Port or NL Port may attach.
fabric	A group of interconnections between ports that includes a fabric element.
FCP	Fibre Channel Protocol
FL Port	A port in a fabric where an N Port or an NL Port may attach.
GBIC	Gigabit Interface Converter. The physical hardware used with Fibre Channel switches and some HBAs to allow different connectors for the cables.
GNBD	GNBD Network Block Device. A method of sharing a disk on one node to many other nodes.
HBA	Host Bus Adapter. The physical hardware installed in a node that allows the node to access a shared network medium.
JBOD	Just a Bunch of Disks. Acronym for miscellaneous storage devices.
L Port	An arbitrated loop port: either an NL Port, an FL Port, or a GL Port
MultiFence	Disables a device so it cannot access the file system. This is the first step in the recovery process for failed nodes.
LUN	Logical Unit Number
N Port	A port attached to a node for use with point-to-point or fabric topologies.
NL Port	A port attached to a node for use in all three topologies.
node	A device that has at least one N Port or NL Port (Fibre Channel only).
NPS	Network Power Switch
point-to-point	A topology where exactly two ports communicate.
RAID	Redundant Arrays of Independent Disks

Term	Meaning
switch	A particular implementation of a fabric topology. Almost exclusively a hardware device.
storage cluster	A group of networked computers that have equal, concurrent access to a shared storage space.
topology	The arrangement in which the nodes of a LAN are connected to each other.

Fibre Channel Specific Terms

Term	Meaning
Alias Server	A fabric software facility that supports multicast group management.
Arbitrated Loop	The FC Arbitrated Loop (FC-AL) is a standard defined on top of the FC-PH standard. It defines the arbitration on a loop where several FC nodes share a common medium. A FC-AL may contain up to 126 nodes and a fabric port.
Bandwidth	Maximum effective transfer rate for a given set of physical variants such as communication model, payload size, fibre speed and overhead specified by FC-PH.
Classes of Service	Fibre Channel offers the possibility to use different classes of service for the data.
Class 1, Acknowledged Connection Service	Class 1 provides true connection service. The result is circuit switched, dedicated bandwidth connections.
Class 2, Acknowledged Connectionless Service	Class 2 is a connectionless service, independently switching each frame and providing guaranteed delivery with an acknowledgment of receipt.
Class 3, Unacknowledged Connectionless Service	Class 3 is a connectionless service, similar to Class 2, but no confirmation of receipt is given.
Class 4, Fractional Bandwidth Connection Oriented Service	Class 4 is fractional bandwidth, connection-oriented service.
Class 6, Simplex Connection Service	Class 6 is similar to Class 1, providing simplex connection services. However, Class 6 also provides multicast and preemption.

Term	Meaning
Intermix	FC has an optional mode called Intermix. Intermix allows the reservation of full FC bandwidth for a dedicated (Class 1) connection but also allows connectionless traffic within the fabric to share the link during idle Class 1 transmission.
Coaxial Cable	An electrical transmission medium consisting of concentric conductors separated by a dielectric material with the spacings and material arranged to give a specified electrical impedance.
Credit	The maximum number of receive buffers allocated to a transmitting port. It represents the maximum number of outstanding frames which can be transmitted by that port without causing a buffer overrun condition at the receiver.
Domain ID	The domain number uniquely identifies the switch in a fabric.
D ID	Destination Identifier, the address used to indicate the targeted destination of the transmitted frame.
Destination Port	Port to which a frame is targeted.
E Ports	E Ports (Expansion Ports) are ports used for switch-to-switch communication. This allows several switches to be connected, forming larger fabrics.
EDTOV	Error Detect Timeout value.
Exchange	The basic mechanism which transfers information consisting of one or more related non-concurrent sequences which may flow in the same or opposite directions. An exchange may span multiple Class 1 dedicated connections. The exchange is identified by an Originator Exchange Identifier (OX ID) and a Responder Exchange Identifier (RX ID).
Exchange Identifier (X ID)	A generic reference to OX ID and RX ID (See Exchange).
F BSY	Fabric Port Busy.
F BSY(DF)	F BSY response to a data frame.
F BSY(LC)	F BSY response to any link control except P-BSY.
F CTL	Frame Control.
F Port	F Ports (Fabric Ports) are the ports used on a switch. A F Port can only participate in a point-to-point connection with a N Port.

Term	Meaning
Fabric	The entity which interconnects various N Ports / NL Ports attached to it and is capable of routing frames by only using the D ID information in the frame header.
FCP	Fibre Channel Protocol. (SCSI over FC)
FC-PA	ANSI X3.230-1994, Fibre Channel Physical and Signaling Interface.
Fibre	A general term used to cover all transmission media specified in FC-PH. (Not to be confused with Fiber)
FL Port	FabricLoop Ports used on a Switch. This port is a F Port that contains the extra functionality to participate on Fibre Channel Arbitrated Loops. When connected to an Arbitrated Loop on a switch, all devices on the loop will have access to all other devices connected to the switch.
Frame	An indivisible unit of information used by FC-2.
Frame Content	The information contained in a frame between the Start-of-Frame and End-of-Frame delimiters, excluding the delimiters.
GBIC	GigaBit Interface Converter. A removable serial transmitter module designed to provide gigabaud capability for Fibre Channel and other protocols that use the same physical layer.
G Ports	Generic ports on a switch that will behave as a E Port, FL Port or F Port, depending on what it is connected to them.
Hard Address	The AL PA which an NL Port attempts to acquire in the LIHA loop initialization Sequence.
HSSDC	High Speed Serial Data Connector. Used for 1 Gigabit connectors.
HSSDC2	High Speed Serial Data Connector 2. Used for 2 Gigabit connectors.
Initiator	A SCSI device containing application clients that originate device requests and task management functions to be processed by a target SCSI device.
Interface connector	An optical or electrical connector which connects the media to the Fibre Channel transmitter or receiver.
Interswitch Link (ISL)	ISL is a fibre link between two switches.

Term	Meaning
L Port	A term used for describing a N Port or F Port that contains Arbitrated Loop functions associated with Arbitrated Loop topology.
LAN	Local Area Network.
LED	Light Emitting Diode.
LIP	Loop Initialization Primitive. One primitive sequence defined in FC. The LIP is used in Arbitrated Loop configurations to reset all attached ports to a known state.
LOGI	Login.
L Ports	L Port (Loop Port) is not a term in Fibre Channel, but is often used as a generic term for NL Ports and FL Ports when one can determine from the context what port is meant.
L Ports Login BB Credits	On FC-AL, equal to the number of receive buffer that a receiving NL Port must have available when a loop circuit is established. Login BB Credit is discovered in the PDISC or PLOGI protocol.
Loop ID	7-bit values numbered contiguously from 0 to 126 to represent the 127 legal hard addresses on a loop (not all of the 256 possible PA ALs are used in FC-AL for reasons related to running disparity). Loop IDs correspond to the 7-bit SEL word in SFF-8045 used for specifying hard addresses. Decimal 127 (7F hex) is not a valid Loop ID, but is used to signify that no hard address is being assigned to an NL Port.
Meaningful	A control field or bit shall be applicable and shall be interpreted by the receiver whenever it is specified as meaningful. Whenever it is specified as not meaningful, it shall be ignored.
Multicast	Multicast is used when multiple copies of data are to be sent to designated multiple destinations.
N Port	Node Port. A hardware entity which includes a Link Control Facility. It may act as an originator, a responder, or both. N Ports are the simplest ports. A N Port can only participate in a point-to-point connection. A node with a N Port only connects to another node with a N Port or to a switch. Using a switch means that more nodes can be reached, even with just a N Port.

Term	Meaning
N Port Identifier	A fabric-unique address identifier by which a N Port is uniquely known. The identifier may be assigned by the fabric during the initialization procedure. The identifier may also be assigned by other procedures not defined in FC-PH. The identifier is used in the S ID and D ID fields of a frame.
NA	Not applicable.
Name Identifier	A 64-bit identifier, with a 60-bit value preceded with a 4-bit Network Address Authority Identifier, used to identify entities in Fibre Channel such as N Port, Node, F Port or Fabric.
NL Port	A N Port that contains the extra functionality to participate on Fibre Channel Arbitrated Loops.
Node	A collection of one or more N Ports or NL Ports controlled by a level above FC-2.
Optical Fibre	Any filament or fibre, made of dielectric material, that guides light.
Payload	Contents of the data field of a frame, excluding optional headers and fill bytes, if present.
PDISC	Discover N Port Service parameters.
Ports	All equipment connected to Fibre Channel have one or more FC Ports. These ports are entities capable of sending and receiving data under the Fibre Channel Protocol. For a computer, the Host Bus Adapter is the port. Fibre Channel specifies different ports with different purposes. Although all the different ports physically can be connected together, only logical valid connections will provide a Fibre Channel connection. Port types are: N Port, NL Port, F Port, FL Port, G Port.
Plug	The cable half of the interface connector which terminates an optical or electrical signal transmission cable.
Preferred Address	On FC-AL, the AL PA which a NL Port attempts to acquire first during loop initialization. Following power-on reset, the preferred address of a private NL Port is its hard address (if any). Following receipt of a LIP other than LIP(AL PD,AL PS), the preferred address of a private NL Port is its previously acquired address. Fabric-assigned or soft addresses are not considered to be preferred addresses.

Term	Meaning
Responder Exchange Identifier (RX ID)	An identifier assigned by a Responder to identify an Exchange and meaningful only to the Responder.
Running Disparity	A binary parameter indicating the cumulative disparity (positive or negative) of all previously issued transmission characters.
SOF	Start of Frame
Source Identifier (S ID)	The address identifier used to indicate the source port of the transmitted frame.
STP	Shielded twisted pair.
Target	A SCSI device that receives SCSI commands and directs such commands to one or more logical units for execution.
Well-known addresses	A set of address identifiers defined in FC-PH to access global server functions such as a name server.
Worldwide Name	A name identifier which is worldwide unique. For FC, 64-bit unsigned binary values are used.

Index

A

- agents (MultiFence) 4
- Alpha 2, 4
- APC MasterSwitch
 - fence method 31
 - node recovery 106
- Arbitrated Loop 93, 95
- archive 105

B

- backup 73, 78
- bandwidth 5, 86, 95
- binaries 37
- BIOS 100
- boot 37
- bootable file system 3
- Brocade FC switch 31, 39, 101, 106
- BSD style quotas (UID/GID) 3

C

- cache 3
- CDPN 3, 81
 - expansion strings 80
- channels 91
- cid 32, 77
- CIDEV 27, 35
- client ID 27
- cluster
 - adding journals 78
 - application 8
 - removing a node 78
 - volume manager (pool) 4
- Context Dependent Path Names (CDPN) 80
- copper cabling 92

D

- device number limit 97

- direct I/O 86
- DMEP 7, 27, 28, 36
- drivers 97
- dynamic multipathing
 - failover 87
 - round-robin 87

E

- Emulex HBAs 99, 100
- etc symlink 81

F

- fabrics 93
- failback 88
- failover 7, 82, 87
- fctl 86
- Feral Software 100, 101
- Fibre Channel
 - configuration example 39, 47, 53
 - hardware 91–101
- file access time 85
- finding a storage device 109
- flock 86
- FreeBSD 101
- FS-aligned I/O 86
- fuzzy quotas 3

G

- GFS
 - adding a node to a cluster 77
 - adding storage hardware 73
 - clusters 2, 28
 - command summary 111–112
 - configuration file 77
 - configuring hardware 91
 - erasing device labels 34
 - evaluation license 80

- Fibre Channel with DMEP topology 11
- Fibre Channel with IP Lock Server topology 11
- file system tunables 89
- forms for configuration 113–115
- hardware configuration 9
- introduction 1
- large files 82
- loading modules 34
- loading pools 35
- local file system topology 14
- lock harness 6
- lock module 6
- making the file system 35
- mounting the file system 36
- network block device topology 13
- partitions 36, 37
- removing a node from a cluster 78
- starting 33
- starting the daemons 36
- topology illustrations 10–14
- updating the license 79
- writing config information 35
- writing pool files to disk 34
- glock layer 6
- GNBD 4, 5, 106
 - configuration example 65
 - fence method 32
- GULM
 - configuration 28, 29, 36
 - introduction 7
 - IP lock server 36
 - lock server 53
- gzip 105
- H**
- HBA drivers 108
- HBAs 5, 99
- Hierarchical Storage Managers (HSMs) 85
- HIPPI 92
- hostname 81
- hot adding storage units 97
- hot removing storage units 99
- hubs 94
- I**
- IA-32 2, 4
- IA-64 4
- IDE boot disk 9
- initrc scripts 37
- Inodes 2
- Interphase FC HBA 101
- IP
 - networks 5
 - supported protocol 92
- J**
- JBOD 5, 95
- JNI FC HBA 101
- journal
 - boot advantage 37
 - isolating in subpools 86
 - journaled file system 2
 - journaled file system on local node 71
 - subpools 78
- L**
- Large File Summit (LFS) specification 82
- license 46, 58, 79
- Linux
 - FC 97
 - file size maximum 75
 - FreeBSD driver support 101
 - mount command 36
- load sharing 1, 82, 87
- local file system
 - configuration example 71
 - optimizations 86
- local node 71
- lock
 - DMEP 7
 - GULM 7
 - introduction 4, 6
 - layers 6
 - lock modules 6
 - lock server 27, 38

- lock table interfaces 30
- lock table server 30
- lock_swdmepd lock server 47, 107
- Multifence introduction 8
- NoLock module 7
- Loop ID 100
- LUNs 96
- M**
- machine type 81
- mail user agents 85
- Manual fence method 32
- member nodes 32
- memory mapping 3
- meta data 2, 87
- modular locking 4
- mount
 - failure 107, 108
 - invoke options 85
 - script 38
- MultiFence 2, 4, 6, 8, 27, 30, 32, 39, 47, 108
- multihost file system 86
- multi-port storage 87
- N**
- NBD server 65
- Network Power Switch 8, 47, 53, 105
- networks 91
- NFS 2
- node multifence timeout period 108
- node recovery 105
- O**
- optical cabling 92
- P**
- panic trap 105
- partitioning 65
- Perl Telnet module 47, 53
- Point-to-Point FC topology 93
- Pool 1, 4, 5, 27, 65, 82, 95
- POSIX 2
- private loop 94
- probing 88
- protocols for mapping data 92
- public loop 94
- Q**
- QLogic 101
- QLogic HBAs 100
- quantum period 89
- quiesce 3
- quotas 3, 89
- R**
- RAID 5, 54, 96
- read-only file system 92
- recovery 8
- resource index 75
- round-robin 82, 87
- S**
- SAN 5
- SCSI
 - configuration example 59
 - disconnecting subsystem 99
 - errors 97
 - FC interface 96
 - protocol 92
 - rescan bus 98
- Scyld 83
- serial cable 96
- shared storage devices 5, 80, 92, 95
- Sistina
 - contacting for support 103
 - email 103
 - land phone 103
 - technical support 104
 - website 103
 - white papers 103
- snapshotting 3
- split mirrors 3
- SSD storage 87
- star topology 95
- stripe 65

- stripe groups 86
- stripe size 88
- striped pools 5
- subpools 73, 86, 87
- syslog 106

T

- tar 105
- TCP/IP 4
- timeout 27, 88
- troubleshooting 105–109

U

- UDP port 28, 29
- UNIX 83

V

- VFS layer 86

W

- warnings 34
- website 4
- WTI Network Power Switch 30, 105, 106