Red Hat Enterprise Linux 6 Global File System 2

Red Hat Global File System 2

Steven Levine

Red Hat Global File System 2

Steven Levine
Red Hat Customer Content Services
slevine@redhat.com

Legal Notice

Abstract

This book provides information about configuring and maintaining Red Hat GFS2 (Red Hat Global File System 2) for Red Hat Enterprise Linux 6.

Table of Contents

Introduction

This book provides information about configuring and maintaining Red Hat GFS2 (Red Hat Global File System 2), which is included in the Resilient Storage Add-On.

1. Audience

This book is intended primarily for Linux system administrators who are familiar with the following activities:

- Linux system administration procedures, including kernel configuration
- Installation and configuration of shared storage networks, such as Fibre Channel SANs

2. Related Documentation

For more information about using Red Hat Enterprise Linux, see the following resources:

- *Installation Guide* — Documents relevant information regarding the installation of Red Hat Enterprise Linux 6.
- *Deployment Guide* — Documents relevant information regarding the deployment, configuration and administration of Red Hat Enterprise Linux 6.
- *Storage Administration Guide* — Provides instructions on how to effectively manage storage devices and file systems on Red Hat Enterprise Linux 6.

For more information about the High Availability Add-On and the Resilient Storage Add-On for Red Hat Enterprise Linux 6, see the following resources:

- *High Availability Add-On Overview* — Provides a high-level overview of the Red Hat High Availability Add-On.
- *Cluster Administration* — Provides information about installing, configuring and managing the High Availability Add-On.
- *Logical Volume Manager Administration* — Provides a description of the Logical Volume Manager (LVM), including information on running LVM in a clustered environment.
- *DM Multipath* — Provides information about using the Device-Mapper Multipath feature of Red Hat Enterprise Linux.
- *Load Balancer Administration* — Provides information on configuring high-performance systems and services with the Load Balancer Add-On, a set of integrated software components that provide Linux Virtual Servers (LVS) for balancing IP load across a set of real servers.
- *Release Notes* — Provides information about the current release of Red Hat products.

Red Hat documents are available in HTML, PDF, and RPM versions on the Red Hat Enterprise Linux Documentation CD and online at https://access.redhat.com/site/documentation/.

3. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: http://bugzilla.redhat.com/ against the product **Red Hat Enterprise Linux 6** and the component **doc-Global_File_System_2**. When submitting a bug report, be sure to mention the manual's identifier:

```
rh-gfs2(EN)-6 (2017-3-8T15:15)
```

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, include the section number and some of the surrounding text so we can find it easily.

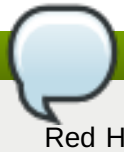
Chapter 1. GFS2 Overview

The Red Hat GFS2 file system is included in the Resilient Storage Add-On. It is a native file system that interfaces directly with the Linux kernel file system interface (VFS layer). When implemented as a cluster file system, GFS2 employs distributed metadata and multiple journals. Red Hat supports the use of GFS2 file systems only as implemented in the High Availability Add-On.

Note

Although a GFS2 file system can be implemented in a standalone system or as part of a cluster configuration, for the Red Hat Enterprise Linux 6 release Red Hat does not support the use of GFS2 as a single-node file system. Red Hat does support a number of high-performance single node file systems which are optimized for single node and thus have generally lower overhead than a cluster file system. Red Hat recommends using these file systems in preference to GFS2 in cases where only a single node needs to mount the file system.

Red Hat will continue to support single-node GFS2 file systems for mounting snapshots of cluster file systems (for example, for backup purposes).



Note

Red Hat does not support using GFS2 for cluster file system deployments greater than 16 nodes.

GFS2 is based on a 64-bit architecture, which can theoretically accommodate an 8 EB file system. However, the current supported maximum size of a GFS2 file system for 64-bit hardware is 100 TB. The current supported maximum size of a GFS2 file system for 32-bit hardware is 16 TB. If your system requires larger GFS2 file systems, contact your Red Hat service representative.

When determining the size of your file system, you should consider your recovery needs. Running the **fsck.gfs2** command on a very large file system can take a long time and consume a large amount of memory. Additionally, in the event of a disk or disk-subsystem failure, recovery time is limited by the speed of your backup media. For information on the amount of memory the **fsck.gfs2** command requires, see Section 4.11, “Repairing a File System”.

When configured in a cluster, Red Hat GFS2 nodes can be configured and managed with High Availability Add-On configuration and management tools. Red Hat GFS2 then provides data sharing among GFS2 nodes in a cluster, with a single, consistent view of the file system name space across the GFS2 nodes. This allows processes on different nodes to share GFS2 files in the same way that processes on the same node can share files on a local file system, with no discernible difference. For information about the High Availability Add-On see *Configuring and Managing a Red Hat Cluster*.

While a GFS2 file system may be used outside of LVM, Red Hat supports only GFS2 file systems that are created on a CLVM logical volume. CLVM is included in the Resilient Storage Add-On. It is a cluster-wide implementation of LVM, enabled by the CLVM daemon **clvmd**, which manages LVM logical volumes in a cluster. The daemon makes it possible to use LVM2 to manage logical volumes across a cluster, allowing all nodes in the cluster to share the logical volumes. For information on the LVM volume manager, see *Logical Volume Manager Administration*

The **gfs2.ko** kernel module implements the GFS2 file system and is loaded on GFS2 cluster nodes.

Note

When you configure a GFS2 file system as a cluster file system, you must ensure that all nodes in the cluster have access to the shared storage. Asymmetric cluster configurations in which some nodes have access to the shared storage and others do not are not supported. This does not require that all nodes actually mount the GFS2 file system itself.

This chapter provides some basic, abbreviated information as background to help you understand GFS2. It contains the following sections:

- Section 1.1, “New and Changed Features”
- Section 1.2, “Before Setting Up GFS2”
- Section 1.4, “Differences between GFS and GFS2”
- Section 1.3, “Installing GFS2”
- Section 2.9, “GFS2 Node Locking”

1.1. New and Changed Features

This section lists new and changed features of the GFS2 file system and the GFS2 documentation that are included with the initial and subsequent releases of Red Hat Enterprise Linux 6.

1.1.1. New and Changed Features for Red Hat Enterprise Linux 6.0

Red Hat Enterprise Linux 6.0 includes the following documentation and feature updates and changes.

- For the Red Hat Enterprise Linux 6 release, Red Hat does not support the use of GFS2 as a single-node file system.
- For the Red Hat Enterprise Linux 6 release, the **gfs2_convert** command to upgrade from a GFS to a GFS2 file system has been enhanced. For information on this command, see Appendix B, *Converting a File System from GFS to GFS2*.
- The Red Hat Enterprise Linux 6 release supports the **discard**, **nodiscard**, **barrier**, **nobarrier**, **quota_quantum**, **statfs_quantum**, and **statfs_percent** mount options. For information about mounting a GFS2 file system, see Section 4.2, “Mounting a File System”.
- The Red Hat Enterprise Linux 6 version of this document contains a new section, Section 2.9, “GFS2 Node Locking”. This section describes some of the internals of GFS2 file systems.

1.1.2. New and Changed Features for Red Hat Enterprise Linux 6.1

Red Hat Enterprise Linux 6.1 includes the following documentation and feature updates and changes.

- As of the Red Hat Enterprise Linux 6.1 release, GFS2 supports the standard Linux quota facilities. GFS2 quota management is documented in Section 4.5, “GFS2 Quota Management”.

  For earlier releases of Red Hat Enterprise Linux, GFS2 required the **gfs2_quota** command to manage quotas. Documentation for the **gfs2_quota** is now provided in Appendix A, *GFS2 Quota Management with the **gfs2_quota** Command*.

- This document now contains a new chapter, Chapter 5, *Diagnosing and Correcting Problems with GFS2 File Systems*.
- Small technical corrections and clarifications have been made throughout the document.

1.1.3. New and Changed Features for Red Hat Enterprise Linux 6.2

Red Hat Enterprise Linux 6.2 includes the following documentation and feature updates and changes.

- As of the Red Hat Enterprise Linux 6.2 release, GFS2 supports the **tunegfs2** command, which replaces some of the features of the **gfs2_tool** command. For further information, see the **tunegfs2** man page.

  The following sections have been updated to provide administrative procedures that do not require the use of the **gfs2_tool** command:

  - Section 4.5.4, "Synchronizing Quotas with the **quotasync** Command". and Section A.3, "Synchronizing Quotas with the **gfs2_quota** Command" now describe how to change the **quota_quantum** parameter from its default value of 60 seconds by using the **quota_quantum=** mount option.
  - Section 4.10, "Suspending Activity on a File System" now describes how to suspend write activity to a file system using the **dmsetup suspend** command.
- This document includes a new appendix, Appendix C, *GFS2 tracepoints and the debugfs glocks File*. This appendix describes the glock **debugfs** interface and the GFS2 tracepoints. It is intended for advanced users who are familiar with file system internals who would like to learn more about the design of GFS2 and how to debug GFS2-specific issues.

1.1.4. New and Changed Features for Red Hat Enterprise Linux 6.3

For the Red Hat Enterprise Linux 6.3 release, this document contains a new chapter, Chapter 2, *GFS2 Configuration and Operational Considerations*. This chapter provides recommendations for optimizing GFS2 performance, including recommendations for creating, using, and maintaining a GFS2 file system.

In addition, small clarifications and corrections have been made throughout the document.

1.1.5. New and Changed Features for Red Hat Enterprise Linux 6.4

For the Red Hat Enterprise Linux 6.4 release, Chapter 2, *GFS2 Configuration and Operational Considerations* has been updated with small clarifications.

1.1.6. New and Changed Features for Red Hat Enterprise Linux 6.6

For the Red Hat Enterprise Linux 6.6 release, this document contains a new chapter, Chapter 6, *Configuring a GFS2 File System in a Pacemaker Cluster*. This chapter provides an outline of the steps required to set up a Pacemaker cluster that includes a GFS2 file system.

In addition, small clarifications and corrections have been made throughout the document.

1.2. Before Setting Up GFS2

Before you install and set up GFS2, note the following key characteristics of your GFS2 file systems:

GFS2 nodes

Determine which nodes in the cluster will mount the GFS2 file systems.

Number of file systems

Determine how many GFS2 file systems to create initially. (More file systems can be added later.)

File system name

Determine a unique name for each file system. The name must be unique for all **lock_dlm** file systems over the cluster. Each file system name is required in the form of a parameter variable. For example, this book uses file system names **mydata1** and **mydata2** in some example procedures.

Journals

Determine the number of journals for your GFS2 file systems. One journal is required for each node that mounts a GFS2 file system. GFS2 allows you to add journals dynamically at a later point as additional servers mount a file system. For information on adding journals to a GFS2 file system, see Section 4.7, “Adding Journals to a File System”.

Storage devices and partitions

Determine the storage devices and partitions to be used for creating logical volumes (by means of CLVM) in the file systems.

Note

You may see performance problems with GFS2 when many create and delete operations are issued from more than one node in the same directory at the same time. If this causes performance problems in your system, you should localize file creation and deletions by a node to directories specific to that node as much as possible.

For further recommendations on creating, using, and maintaining a GFS2 file system. see Chapter 2, *GFS2 Configuration and Operational Considerations*.

1.3. Installing GFS2

In addition to the packages required for the Red Hat High Availability Add-On, you must install the **gfs2-utils** package for GFS2 and the **lvm2-cluster** package for the Clustered Logical Volume Manager (CLVM). The **lvm2-cluster** and **gfs2-utils** packages are part of ResilientStorage channel, which must be enabled before installing the packages.

You can use the following **yum install** command to install the Red Hat High Availability Add-On software packages:

```
# yum install rgmanager lvm2-cluster gfs2-utils
```

For general information on the Red Hat High Availability Add-On and cluster administration, see the *Cluster Administration* manual.

1.4. Differences between GFS and GFS2

This section lists the improvements and changes that GFS2 offers over GFS.

Migrating from GFS to GFS2 requires that you convert your GFS file systems to GFS2 with the **gfs2_convert** utility. For information on the **gfs2_convert** utility, see Appendix B, *Converting a File System from GFS to GFS2*.

1.4.1. GFS2 Command Names

In general, the functionality of GFS2 is identical to GFS. The names of the file system commands, however, specify GFS2 instead of GFS. Table 1.1, "GFS and GFS2 Commands" shows the equivalent GFS and GFS2 commands and functionality.

Table 1.1. GFS and GFS2 Commands

GFS Command	GFS2 Command	Description
mount	**mount**	Mount a file system. The system can determine whether the file system is a GFS or GFS2 file system type. For information on the GFS2 mount options see the gfs2_mount(8) man page.
umount	**umount**	Unmount a file system.
fsck **gfs_fsck**	**fsck** **fsck.gfs2**	Check and repair an unmounted file system.
gfs_grow	**gfs2_grow**	Grow a mounted file system.
gfs_jadd	**gfs2_jadd**	Add a journal to a mounted file system.
gfs_mkfs **mkfs -t gfs**	**mkfs.gfs2** **mkfs -t gfs2**	Create a file system on a storage device.
gfs_quota	**gfs2_quota**	Manage quotas on a mounted file system. As of the Red Hat Enterprise Linux 6.1 release, GFS2 supports the standard Linux quota facilities. For further information on quota management in GFS2, see Section 4.5, "GFS2 Quota Management".
gfs_tool	**tunegfs2** mount parameters **dmsetup suspend**	Configure, tune, or gather information about a file system. The **tunegfs2** command is supported as of the Red Hat Enterprise Linux 6.2 release. There is also a **gfs2_tool** command.
gfs_edit	**gfs2_edit**	Display, print, or edit file system internal structures. The **gfs2_edit** command can be used for GFS file systems as well as GFS2 file system.
gfs_tool setflag jdata/inherit_jdata	**chattr +j** (preferred)	Enable journaling on a file or directory.
setfacl/getfacl	**setfacl/getfacl**	Set or get file access control list for a file or directory.
setfattr/getfattr	**setfattr/getfattr**	Set or get the extended attributes of a file.

For a full listing of the supported options for the GFS2 file system commands, see the man pages for those commands.

1.4.2. Additional Differences Between GFS and GFS2

This section summarizes the additional differences in GFS and GFS2 administration that are not described in Section 1.4.1, “GFS2 Command Names”.

Context-Dependent Path Names

GFS2 file systems do not provide support for context-dependent path names, which allow you to create symbolic links that point to variable destination files or directories. For this functionality in GFS2, you can use the **bind** option of the **mount** command. For information on bind mounts and context-dependent pathnames in GFS2, see Section 4.12, “Bind Mounts and Context-Dependent Path Names”.

gfs2.ko Module

The kernel module that implements the GFS file system is **gfs.ko**. The kernel module that implements the GFS2 file system is **gfs2.ko**.

Enabling Quota Enforcement in GFS2

In GFS2 file systems, quota enforcement is disabled by default and must be explicitly enabled. For information on enabling and disabling quota enforcement, see Section 4.5, “GFS2 Quota Management”.

Data Journaling

GFS2 file systems support the use of the **chattr** command to set and clear the **j** flag on a file or directory. Setting the **+j** flag on a file enables data journaling on that file. Setting the **+j** flag on a directory means "inherit jdata", which indicates that all files and directories subsequently created in that directory are journaled. Using the **chattr** command is the preferred way to enable and disable data journaling on a file.

Adding Journals Dynamically

In GFS file systems, journals are embedded metadata that exists outside of the file system, making it necessary to extend the size of the logical volume that contains the file system before adding journals. In GFS2 file systems, journals are plain (though hidden) files. This means that for GFS2 file systems, journals can be dynamically added as additional servers mount a file system, as long as space remains on the file system for the additional journals. For information on adding journals to a GFS2 file system, see Section 4.7, “Adding Journals to a File System”.

atime_quantum parameter removed

The GFS2 file system does not support the **atime_quantum** tunable parameter, which can be used by the GFS file system to specify how often **atime** updates occur. In its place GFS2 supports the **relatime** and **noatime** mount options. The **relatime** mount option is recommended to achieve similar behavior to setting the **atime_quantum** parameter in GFS.

The data= option of the mount command

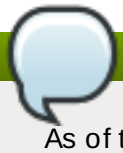
When mounting GFS2 file systems, you can specify the **data=ordered** or **data=writeback** option

of the **mount**. When **data=ordered** is set, the user data modified by a transaction is flushed to the disk before the transaction is committed to disk. This should prevent the user from seeing uninitialized blocks in a file after a crash. When **data=writeback** is set, the user data is written to the disk at any time after it is dirtied. This does not provide the same consistency guarantee as **ordered** mode, but it should be slightly faster for some workloads. The default is **ordered** mode.

The gfs2_tool command

The **gfs2_tool** command supports a different set of options for GFS2 than the **gfs_tool** command supports for GFS:

- The **gfs2_tool** command supports a **journals** parameter that prints out information about the currently configured journals, including how many journals the file system contains.
- The **gfs2_tool** command does not support the **counters** flag, which the **gfs_tool** command uses to display GFS statistics.
- The **gfs2_tool** command does not support the **inherit_jdata** flag. To flag a directory as "inherit jdata", you can set the **jdata** flag on the directory or you can use the **chattr** command to set the **+j** flag on the directory. Using the **chattr** command is the preferred way to enable and disable data journaling on a file.



Note

As of the Red Hat Enterprise Linux 6.2 release, GFS2 supports the **tunegfs2** command, which replaces some of the features of the **gfs2_tool** command. For further information, refer to the **tunegfs2**(8) man page. The **settune** and **gettune** functions of the **gfs2_tool** command have been replaced by command line options of the **mount** command, which allows them to be set by means of the **fstab** file when required.

The gfs2_edit command

The **gfs2_edit** command supports a different set of options for GFS2 than the **gfs_edit** command supports for GFS. For information on the specific options each version of the command supports, see the **gfs2_edit** and **gfs_edit** man pages.

1.4.3. GFS2 Performance Improvements

There are many features of GFS2 file systems that do not result in a difference in the user interface from GFS file systems but which improve file system performance.

A GFS2 file system provides improved file system performance in the following ways:

- Better performance for heavy usage in a single directory
- Faster synchronous I/O operations
- Faster cached reads (no locking overhead)
- Faster direct I/O with preallocated files (provided I/O size is reasonably large, such as 4M blocks)
- Faster I/O operations in general
- Faster execution of the **df** command, because of faster **statfs** calls

- Improved **atime** mode to reduce the number of write I/O operations generated by **atime** when compared with GFS

GFS2 file systems provide broader and more mainstream support in the following ways:

- GFS2 is part of the upstream kernel (integrated into 2.6.19).
- GFS2 supports the following features.
  - extended file attributes (**xattr**)
  - the **lsattr**() and **chattr**() attribute settings by means of standard **ioctl**() calls
  - nanosecond timestamps

A GFS2 file system provides the following improvements to the internal efficiency of the file system.

- GFS2 uses less kernel memory.
- GFS2 requires no metadata generation numbers.

  Allocating GFS2 metadata does not require reads. Copies of metadata blocks in multiple journals are managed by revoking blocks from the journal before lock release.
- GFS2 includes a much simpler log manager that knows nothing about unlinked inodes or quota changes.
- The **gfs2_grow** and **gfs2_jadd** commands use locking to prevent multiple instances running at the same time.
- The ACL code has been simplified for calls like **creat**() and **mkdir**().
- Unlinked inodes, quota changes, and **statfs** changes are recovered without remounting the journal.

Chapter 2. GFS2 Configuration and Operational Considerations

The Global File System 2 (GFS2) file system allows several computers ("nodes") in a cluster to cooperatively share the same storage. To achieve this cooperation and maintain data consistency among the nodes, the nodes employ a cluster-wide locking scheme for file system resources. This locking scheme uses communication protocols such as TCP/IP to exchange locking information.

You can improve performance by following the recommendations described in this chapter, including recommendations for creating, using, and maintaining a GFS2 file system.

Important

Make sure that your deployment of Red Hat High Availability Add-On meets your needs and can be supported. Consult with an authorized Red Hat representative to verify your configuration prior to deployment.

2.1. Formatting Considerations

This section provides recommendations for how to format your GFS2 file system to optimize performance.

2.1.1. File System Size: Smaller is Better

GFS2 is based on a 64-bit architecture, which can theoretically accommodate an 8 EB file system. However, the current supported maximum size of a GFS2 file system for 64-bit hardware is 100 TB and the current supported maximum size of a GFS2 file system for 32-bit hardware is 16 TB.

Note that even though GFS2 large file systems are possible, that does not mean they are recommended. The rule of thumb with GFS2 is that smaller is better: it is better to have 10 1TB file systems than one 10TB file system.

There are several reasons why you should keep your GFS2 file systems small:

- Less time is required to back up each file system.
- Less time is required if you need to check the file system with the **`fsck.gfs2`** command.
- Less memory is required if need to check the file system with the **`fsck.gfs2`** command.

In addition, fewer resource groups to maintain mean better performance.

Of course, if you make your GFS2 file system too small, you might run out of space, and that has its own consequences. You should consider your own use cases before deciding on a size.

2.1.2. Block Size: Default (4K) Blocks Are Preferred

As of the Red Hat Enterprise Linux 6 release, the **`mkfs.gfs2`** command attempts to estimate an optimal block size based on device topology. In general, 4K blocks are the preferred block size because 4K is the default page size (memory) for Linux. Unlike some other file systems, GFS2 does most of its operations using 4K kernel buffers. If your block size is 4K, the kernel has to do less work to manipulate the buffers.

It is recommended that you use the default block size, which should yield the highest performance. You may need to use a different block size only if you require efficient storage of many very small files.

2.1.3. Number of Journals: One for Each Node that Mounts

GFS2 requires one journal for each node in the cluster that needs to mount the file system. For example, if you have a 16-node cluster but need to mount only the file system from two nodes, you need only two journals. If you need to mount from a third node, you can always add a journal with the **gfs2_jadd** command. With GFS2, you can add journals on the fly.

2.1.4. Journal Size: Default (128MB) Is Usually Optimal

When you run the **mkfs.gfs2** command to create a GFS2 file system, you may specify the size of the journals. If you do not specify a size, it will default to 128MB, which should be optimal for most applications.

Some system administrators might think that 128MB is excessive and be tempted to reduce the size of the journal to the minimum of 8MB or a more conservative 32MB. While that might work, it can severely impact performance. Like many journaling file systems, every time GFS2 writes metadata, the metadata is committed to the journal before it is put into place. This ensures that if the system crashes or loses power, you will recover all of the metadata when the journal is automatically replayed at mount time. However, it does not take much file system activity to fill an 8MB journal, and when the journal is full, performance slows because GFS2 has to wait for writes to the storage.

It is generally recommended to use the default journal size of 128MB. If your file system is very small (for example, 5GB), having a 128MB journal might be impractical. If you have a larger file system and can afford the space, using 256MB journals might improve performance.

2.1.5. Size and Number of Resource Groups

When a GFS2 file system is created with the **mkfs.gfs2** command, it divides the storage into uniform slices known as resource groups. It attempts to estimate an optimal resource group size (ranging from 32MB to 2GB). You can override the default with the **-r** option of the **mkfs.gfs2** command.

Your optimal resource group size depends on how you will use the file system. Consider how full it will be and whether or not it will be severely fragmented.

You should experiment with different resource group sizes to see which results in optimal performance. It is a best practice to experiment with a test cluster before deploying GFS2 into full production.

If your file system has too many resource groups (each of which is too small), block allocations can waste too much time searching tens of thousands (or hundreds of thousands) of resource groups for a free block. The more full your file system, the more resource groups that will be searched, and every one of them requires a cluster-wide lock. This leads to slow performance.

If, however, your file system has too few resource groups (each of which is too big), block allocations might contend more often for the same resource group lock, which also impacts performance. For example, if you have a 10GB file system that is carved up into five resource groups of 2GB, the nodes in your cluster will fight over those five resource groups more often than if the same file system were carved into 320 resource groups of 32MB. The problem is exacerbated if your file system is nearly full because every block allocation might have to look through several resource groups before it finds one with a free block. GFS2 tries to mitigate this problem in two ways:

- First, when a resource group is completely full, it remembers that and tries to avoid checking it for

future allocations (until a block is freed from it). If you never delete files, contention will be less severe. However, if your application is constantly deleting blocks and allocating new blocks on a file system that is mostly full, contention will be very high and this will severely impact performance.

- Second, when new blocks are added to an existing file (for example, appending) GFS2 will attempt to group the new blocks together in the same resource group as the file. This is done to increase performance: on a spinning disk, seeks take less time when they are physically close together.

The worst-case scenario is when there is a central directory in which all the nodes create files because all of the nodes will constantly fight to lock the same resource group.

2.2. File System Fragmentation

Red Hat Enterprise Linux 6.4 introduces improvements to file fragmentation management in GFS2. With Red Hat Enterprise Linux 6.4, simultaneous writes result in less file fragmentation and therefore better performance for these workloads.

While there is no defragmentation tool for GFS2 on Red Hat Enterprise Linux, you can defragment individual files by identifying them with the filefrag tool, copying them to temporary files, and renaming the temporary files to replace the originals. (This procedure can also be done in versions prior to Red Hat Enterprise Linux 6.4 as long as the writing is done sequentially.)

2.3. Block Allocation Issues

This section provides a summary of issues related to block allocation in GFS2 file systems. Even though applications that only write data typically do not care how or where a block is allocated, a little knowledge about how block allocation works can help you optimize performance.

2.3.1. Leave Free Space in the File System

When a GFS2 file system is nearly full, the block allocator starts to have a difficult time finding space for new blocks to be allocated. As a result, blocks given out by the allocator tend to be squeezed into the end of a resource group or in tiny slices where file fragmentation is much more likely. This file fragmentation can cause performance problems. In addition, when a GFS2 is nearly full, the GFS2 block allocator spends more time searching through multiple resource groups, and that adds lock contention that would not necessarily be there on a file system that has ample free space. This also can cause performance problems.

For these reasons, it is recommended that you not run a file system that is more than 85 percent full, although this figure may vary depending on workload.

2.3.2. Have Each Node Allocate its Own Files, If Possible

Due to the way the distributed lock manager (DLM) works, there will be more lock contention if all files are allocated by one node and other nodes need to add blocks to those files.

In GFS (version 1), all locks were managed by a central lock manager whose job was to control locking throughout the cluster. This grand unified lock manager (GULM) was problematic because it was a single point of failure. GFS2's replacement locking scheme, DLM, spreads the locks throughout the cluster. If any node in the cluster goes down, its locks are recovered by the other nodes.

With DLM, the first node to lock a resource (like a file) becomes the "lock master" for that lock. Other

nodes may lock that resource, but they have to ask permission from the lock master first. Each node knows which locks for which it is the lock master, and each node knows which node it has lent a lock to. Locking a lock on the master node is much faster than locking one on another node that has to stop and ask permission from the lock's master.

As in many file systems, the GFS2 allocator tries to keep blocks in the same file close to one another to reduce the movement of disk heads and boost performance. A node that allocates blocks to a file will likely need to use and lock the same resource groups for the new blocks (unless all the blocks in that resource group are in use). The file system will run faster if the lock master for the resource group containing the file allocates its data blocks (that is, it is faster to have the node that first opened the file do all the writing of new blocks).

2.3.3. Preallocate, If Possible

If files are preallocated, block allocations can be avoided altogether and the file system can run more efficiently. Newer versions of GFS2 include the **`fallocate`**(1) system call, which you can use to preallocate blocks of data.

2.4. Cluster Considerations

When determining the number of nodes that your system will contain, note that there is a trade-off between high availability and performance. With a larger number of nodes, it becomes increasingly difficult to make workloads scale. For that reason, Red Hat does not support using GFS2 for cluster file system deployments greater than 16 nodes.

Deploying a cluster file system is not a "drop in" replacement for a single node deployment. We recommend that you allow a period of around 8-12 weeks of testing on new installations in order to test the system and ensure that it is working at the required performance level. During this period any performance or functional issues can be worked out and any queries should be directed to the Red Hat support team.

We recommend that customers considering deploying clusters have their configurations reviewed by Red Hat support before deployment to avoid any possible support issues later on.

2.5. Usage Considerations

This section provides general recommendations about GFS2 usage.

2.5.1. Mount Options: noatime and nodiratime

It is generally recommended to mount GFS2 file systems with the **`noatime`** and **`nodiratime`** arguments. This allows GFS2 to spend less time updating disk inodes for every access.

2.5.2. DLM Tuning Options: Increase DLM Table Sizes

DLM uses several tables to manage, coordinate, and pass lock information between nodes in the cluster. Increasing the size of the DLM tables might increase performance. In Red Hat Enterprise Linux 6.1 and later, the default sizes of these tables have been increased, but you can manually increase them with the following commands:

```
echo 1024 > /sys/kernel/config/dlm/cluster/lkbtbl_size
echo 1024 > /sys/kernel/config/dlm/cluster/rsbtbl_size
echo 1024 > /sys/kernel/config/dlm/cluster/dirtbl_size
```

These commands are not persistent and will not survive a reboot, so you must add them to one of the startup scripts and you must execute them before mounting any GFS2 file systems, or the changes will be silently ignored.

For more detailed information on GFS2 node locking, see Section 2.9, “GFS2 Node Locking”.

2.5.3. VFS Tuning Options: Research and Experiment

Like all Linux file systems, GFS2 sits on top of a layer called the virtual file system (VFS). You can tune the VFS layer to improve underlying GFS2 performance by using the **sysctl**(8) command. For example, the values for **dirty_background_ratio** and **vfs_cache_pressure** may be adjusted depending on your situation. To fetch the current values, use the following commands:

```
sysctl -n vm.dirty_background_ratio
sysctl -n vm.vfs_cache_pressure
```

The following commands adjust the values:

```
sysctl -w vm.dirty_background_ratio=20
sysctl -w vm.vfs_cache_pressure=500
```

You can permanently change the values of these parameters by editing the **/etc/sysctl.conf** file.

To find the optimal values for your use cases, research the various VFS options and experiment on a test cluster before deploying into full production.

2.5.4. SELinux: Avoid SELinux on GFS2

Security Enhanced Linux (SELinux) is highly recommended for security reasons in most situations, but it is not supported for use with GFS2. SELinux stores information using extended attributes about every file system object, and SELinux labels on GFS2 file systems can get out of sync between cluster nodes because of how they are cached in memory.

When mounting a GFS2 file system, you must ensure that SELinux will not attempt to read the **seclabel** element on each file system object by using one of the **context** options as described on the **mount**(8) man page; SELinux will assume that all content in the file system is labeled with the **seclabel** element provided in the **context** mount options. This will also speed up processing as it avoids another disk read of the extended attribute block that could contain **seclabel** elements.

For example, on a system with SELinux in enforcing mode, you can use the following **mount** command to mount the GFS2 file system if the file system is going to contain Apache content. This label will apply to the entire file system; it remains in memory and is not written to disk.

```
# mount -t gfs2 -o context=system_u:object_r:httpd_sys_content_t:s0
/dev/mapper/xyz/mnt/gfs2
```

```
# mount -t gfs2 -o context=system_u:object_r:httpd_sys_content_t:s0
/dev/mapper/xyz/mnt/gfs2
```

If you are not sure whether the file system will contain Apache content, you can use the labels **public_content_rw_t** or **public_content_t**, or you could define a new label altogether and define a policy around it.

Note that in a Pacemaker cluster you should always use Pacemaker to manage a GFS2 file system. You can specify the mount options when you create a GFS2 file system resource, as described in

Chapter 6, *Configuring a GFS2 File System in a Pacemaker Cluster*.

2.5.5. Setting Up NFS Over GFS2

Due to the added complexity of the GFS2 locking subsystem and its clustered nature, setting up NFS over GFS2 requires taking many precautions and careful configuration. This section describes the caveats you should take into account when configuring an NFS service over a GFS2 file system.

Warning

If the GFS2 file system is NFS exported, and NFS client applications use POSIX locks, then you must mount the file system with the **`localflocks`** option. The intended effect of this is to force POSIX locks from each server to be local: that is, non-clustered, independent of each other. (A number of problems exist if GFS2 attempts to implement POSIX locks from NFS across the nodes of a cluster.) For applications running on NFS clients, localized POSIX locks means that two clients can hold the same lock concurrently if the two clients are mounting from different servers. If all clients mount NFS from one server, then the problem of separate servers granting the same locks independently goes away. If you are not sure whether to mount your file system with the **`localflocks`** option, you should not use the option; it is always safer to have the locks working on a clustered basis.

In addition to the locking considerations, you should take the following into account when configuring an NFS service over a GFS2 file system.

- Red Hat supports only Red Hat High Availability Add-On configurations using NFSv3 with locking in an active/passive configuration with the following characteristics:
  - The back-end file system is a GFS2 file system running on a 2 to 16 node cluster.
  - An NFSv3 server is defined as a service exporting the entire GFS2 file system from a single cluster node at a time.
  - The NFS server can fail over from one cluster node to another (active/passive configuration).
  - No access to the GFS2 file system is allowed *except* through the NFS server. This includes both local GFS2 file system access as well as access through Samba or Clustered Samba.
  - There is no NFS quota support on the system.

  This configuration provides HA for the file system and reduces system downtime since a failed node does not result in the requirement to execute the **`fsck`** command when failing the NFS server from one node to another.
- The **`fsid=`** NFS option is mandatory for NFS exports of GFS2.
- If problems arise with your cluster (for example, the cluster becomes inquorate and fencing is not successful), the clustered logical volumes and the GFS2 file system will be frozen and no access is possible until the cluster is quorate. You should consider this possibility when determining whether a simple failover solution such as the one defined in this procedure is the most appropriate for your system.

2.5.6. Samba (SMB or Windows) File Serving over GFS2

As of the Red Hat Enterprise Linux 6.2 release, you can use Samba (SMB or Windows) file serving from a GFS2 file system with CTDB, which allows active/active configurations. For information on Clustered Samba configuration, see the *Cluster Administration* document.

Simultaneous access to the data in the Samba share from outside of Samba is not supported. There is currently no support for GFS2 cluster leases, which slows Samba file serving.

2.6. File System Backups

It is important to make regular backups of your GFS2 file system in case of emergency, regardless of the size of your file system. Many system administrators feel safe because they are protected by RAID, multipath, mirroring, snapshots, and other forms of redundancy, but there is no such thing as safe enough.

It can be a problem to create a backup since the process of backing up a node or set of nodes usually involves reading the entire file system in sequence. If this is done from a single node, that node will retain all the information in cache until other nodes in the cluster start requesting locks. Running this type of backup program while the cluster is in operation will negatively impact performance.

Dropping the caches once the backup is complete reduces the time required by other nodes to regain ownership of their cluster locks/caches. This is still not ideal, however, because the other nodes will have stopped caching the data that they were caching before the backup process began. You can drop caches using the following command after the backup is complete:

```
echo -n 3 > /proc/sys/vm/drop_caches
```

It is faster if each node in the cluster backs up its own files so that the task is split between the nodes. You might be able to accomplish this with a script that uses the **rsync** command on node-specific directories.

The best way to make a GFS2 backup is to create a hardware snapshot on the SAN, present the snapshot to another system, and back it up there. The backup system should mount the snapshot with **-o lockproto=lock_nolock** since it will not be in a cluster.

2.7. Hardware Considerations

You should take the following hardware considerations into account when deploying a GFS2 file system.

- Use Higher-Quality Storage Options

  GFS2 can operate on cheaper shared-storage options, such as iSCSI or Fibre Channel over Ethernet (FCoE), but you will get better performance if you buy higher-quality storage with larger caching capacity. Red Hat performs most quality, sanity, and performance tests on SAN storage with Fibre Channel interconnect. As a general rule, it is always better to deploy something that has been tested first.

- Test Network Equipment Before Deploying

  Higher-quality, faster-network equipment makes cluster communications and GFS2 run faster with better reliability. However, you do not have to purchase the most expensive hardware. Some of the most expensive network switches have problems passing multicast packets, which are used for passing **fcntl** locks (flocks), whereas cheaper commodity network switches are sometimes faster and more reliable. It is a general best practice to try equipment before deploying it into full production.

2.8. Performance Issues: Check the Red Hat Customer Portal

For information on best practices for deploying and upgrading Red Hat Enterprise Linux clusters using the High Availability Add-On and Red Hat Global File System 2 (GFS2) see the article "Red Hat Enterprise Linux Cluster, High Availability, and GFS Deployment Best Practices" on Red Hat Customer Portal at https://access.redhat.com/site/articles/40051.

2.9. GFS2 Node Locking

In order to get the best performance from a GFS2 file system, it is very important to understand some of the basic theory of its operation. A single node file system is implemented alongside a cache, the purpose of which is to eliminate latency of disk accesses when using frequently requested data. In Linux the page cache (and historically the buffer cache) provide this caching function.

With GFS2, each node has its own page cache which may contain some portion of the on-disk data. GFS2 uses a locking mechanism called *glocks* (pronounced gee-locks) to maintain the integrity of the cache between nodes. The glock subsystem provides a cache management function which is implemented using the *distributed lock manager* (DLM) as the underlying communication layer.

The glocks provide protection for the cache on a per-inode basis, so there is one lock per inode which is used for controlling the caching layer. If that glock is granted in shared mode (DLM lock mode: PR) then the data under that glock may be cached upon one or more nodes at the same time, so that all the nodes may have local access to the data.

If the glock is granted in exclusive mode (DLM lock mode: EX) then only a single node may cache the data under that glock. This mode is used by all operations which modify the data (such as the **`write`** system call).

If another node requests a glock which cannot be granted immediately, then the DLM sends a message to the node or nodes which currently hold the glocks blocking the new request to ask them to drop their locks. Dropping glocks can be (by the standards of most file system operations) a long process. Dropping a shared glock requires only that the cache be invalidated, which is relatively quick and proportional to the amount of cached data.

Dropping an exclusive glock requires a log flush, and writing back any changed data to disk, followed by the invalidation as per the shared glock.

The difference between a single node file system and GFS2, then, is that a single node file system has a single cache and GFS2 has a separate cache on each node. In both cases, latency to access cached data is of a similar order of magnitude, but the latency to access uncached data is much greater in GFS2 if another node has previously cached that same data.

Note

Due to the way in which GFS2's caching is implemented the best performance is obtained when either of the following takes place:

- An inode is used in a read only fashion across all nodes.
- An inode is written or modified from a single node only.

Note that inserting and removing entries from a directory during file creation and deletion counts as writing to the directory inode.

It is possible to break this rule provided that it is broken relatively infrequently. Ignoring this rule too often will result in a severe performance penalty.

If you **mmap**() a file on GFS2 with a read/write mapping, but only read from it, this only counts as a read. On GFS though, it counts as a write, so GFS2 is much more scalable with **mmap**() I/O.

If you do not set the **noatime mount** parameter, then reads will also result in writes to update the file timestamps. We recommend that all GFS2 users should mount with **noatime** unless they have a specific requirement for **atime**.

2.9.1. Issues with Posix Locking

When using Posix locking, you should take the following into account:

- Use of Flocks will yield faster processing than use of Posix locks.
- Programs using Posix locks in GFS2 should avoid using the **GETLK** function since, in a clustered environment, the process ID may be for a different node in the cluster.

2.9.2. Performance Tuning With GFS2

It is usually possible to alter the way in which a troublesome application stores its data in order to gain a considerable performance advantage.

A typical example of a troublesome application is an email server. These are often laid out with a spool directory containing files for each user (**mbox**), or with a directory for each user containing a file for each message (**maildir**). When requests arrive over IMAP, the ideal arrangement is to give each user an affinity to a particular node. That way their requests to view and delete email messages will tend to be served from the cache on that one node. Obviously if that node fails, then the session can be restarted on a different node.

When mail arrives by means of SMTP, then again the individual nodes can be set up so as to pass a certain user's mail to a particular node by default. If the default node is not up, then the message can be saved directly into the user's mail spool by the receiving node. Again this design is intended to keep particular sets of files cached on just one node in the normal case, but to allow direct access in the case of node failure.

This setup allows the best use of GFS2's page cache and also makes failures transparent to the application, whether **imap** or **smtp**.

Backup is often another tricky area. Again, if it is possible it is greatly preferable to back up the working set of each node directly from the node which is caching that particular set of inodes. If you have a backup script which runs at a regular point in time, and that seems to coincide with a spike in

the response time of an application running on GFS2, then there is a good chance that the cluster may not be making the most efficient use of the page cache.

Obviously, if you are in the (enviable) position of being able to stop the application in order to perform a backup, then this will not be a problem. On the other hand, if a backup is run from just one node, then after it has completed a large portion of the file system will be cached on that node, with a performance penalty for subsequent accesses from other nodes. This can be mitigated to a certain extent by dropping the VFS page cache on the backup node after the backup has completed with following command:

```
echo -n 3 >/proc/sys/vm/drop_caches
```

However this is not as good a solution as taking care to ensure the working set on each node is either shared, mostly read only across the cluster, or accessed largely from a single node.

2.9.3. Troubleshooting GFS2 Performance with the GFS2 Lock Dump

If your cluster performance is suffering because of inefficient use of GFS2 caching, you may see large and increasing I/O wait times. You can make use of GFS2's lock dump information to determine the cause of the problem.

This section provides an overview of the GFS2 lock dump. For a more complete description of the GFS2 lock dump, see Appendix C, *GFS2 tracepoints and the debugfs glocks File*.

The GFS2 lock dump information can be gathered from the **debugfs** file which can be found at the following path name, assuming that **debugfs** is mounted on **/sys/kernel/debug/**:

```
/sys/kernel/debug/gfs2/fsname/glocks
```

The content of the file is a series of lines. Each line starting with G: represents one glock, and the following lines, indented by a single space, represent an item of information relating to the glock immediately before them in the file.

The best way to use the **debugfs** file is to use the **cat** command to take a copy of the complete content of the file (it might take a long time if you have a large amount of RAM and a lot of cached inodes) while the application is experiencing problems, and then looking through the resulting data at a later date.

Note

It can be useful to make two copies of the **debugfs** file, one a few seconds or even a minute or two after the other. By comparing the holder information in the two traces relating to the same glock number, you can tell whether the workload is making progress (that is, it is just slow) or whether it has become stuck (which is always a bug and should be reported to Red Hat support immediately).

Lines in the **debugfs** file starting with H: (holders) represent lock requests either granted or waiting to be granted. The flags field on the holders line f: shows which: The 'W' flag refers to a waiting request, the 'H' flag refers to a granted request. The glocks which have large numbers of waiting requests are likely to be those which are experiencing particular contention.

Table 2.1, “Glock flags” shows the meanings of the different glock flags and Table 2.2, “Glock holder flags” shows the meanings of the different glock holder flags in the order that they appear in the glock dumps.

Table 2.1. Glock flags

Flag	Name	Meaning
b	Blocking	Valid when the locked flag is set, and indicates that the operation that has been requested from the DLM may block. This flag is cleared for demotion operations and for "try" locks. The purpose of this flag is to allow gathering of stats of the DLM response time independent from the time taken by other nodes to demote locks.
d	Pending demote	A deferred (remote) demote request
D	Demote	A demote request (local or remote)
f	Log flush	The log needs to be committed before releasing this glock
F	Frozen	Replies from remote nodes ignored - recovery is in progress. This flag is not related to file system freeze, which uses a different mechanism, but is used only in recovery.
i	Invalidate in progress	In the process of invalidating pages under this glock
I	Initial	Set when DLM lock is associated with this glock
l	Locked	The glock is in the process of changing state
L	LRU	Set when the glock is on the LRU list
o	Object	Set when the glock is associated with an object (that is, an inode for type 2 glocks, and a resource group for type 3 glocks)
p	Demote in progress	The glock is in the process of responding to a demote request
q	Queued	Set when a holder is queued to a glock, and cleared when the glock is held, but there are no remaining holders. Used as part of the algorithm the calculates the minimum hold time for a glock.
r	Reply pending	Reply received from remote node is awaiting processing
y	Dirty	Data needs flushing to disk before releasing this glock

Table 2.2. Glock holder flags

Flag	Name	Meaning
a	Async	Do not wait for glock result (will poll for result later)
A	Any	Any compatible lock mode is acceptable
c	No cache	When unlocked, demote DLM lock immediately
e	No expire	Ignore subsequent lock cancel requests
E	exact	Must have exact lock mode
F	First	Set when holder is the first to be granted for this lock
H	Holder	Indicates that requested lock is granted
p	Priority	Enqueue holder at the head of the queue
t	Try	A "try" lock
T	Try 1CB	A "try" lock that sends a callback
W	Wait	Set while waiting for request to complete

Having identified a glock which is causing a problem, the next step is to find out which inode it relates to. The glock number (n: on the G: line) indicates this. It is of the form *type*/*number* and if *type* is 2, then the glock is an inode glock and the *number* is an inode number. To track down the inode, you can then run **`find -inum` *`number`*** where *number* is the inode number converted from the hex format in the glocks file into decimal.

Note

If you run the **find** on a file system when it is experiencing lock contention, you are likely to make the problem worse. It is a good idea to stop the application before running the **find** when you are looking for contended inodes.

Table 2.3, "Glock types" shows the meanings of the different glock types.

Table 2.3. Glock types

Type number	Lock type	Use
1	Trans	Transaction lock
2	Inode	Inode metadata and data
3	Rgrp	Resource group metadata
4	Meta	The superblock
5	Iopen	Inode last closer detection
6	Flock	**flock**(2) syscall
8	Quota	Quota operations
9	Journal	Journal mutex

If the glock that was identified was of a different type, then it is most likely to be of type 3: (resource group). If you see significant numbers of processes waiting for other types of glock under normal loads, then report this to Red Hat support.

If you do see a number of waiting requests queued on a resource group lock there may be a number of reason for this. One is that there are a large number of nodes compared to the number of resource groups in the file system. Another is that the file system may be very nearly full (requiring, on average, longer searches for free blocks). The situation in both cases can be improved by adding more storage and using the **gfs2_grow** command to expand the file system.

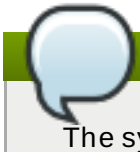
Chapter 3. Getting Started

This chapter describes procedures for initial setup of GFS2 and contains the following sections:

3.1. Prerequisite Tasks

You should complete the following tasks before setting up Red Hat GFS2:

- Make sure that you have noted the key characteristics of the GFS2 nodes (see Section 1.2, “Before Setting Up GFS2”).
- Make sure that the clocks on the GFS2 nodes are synchronized. It is recommended that you use the Network Time Protocol (NTP) software provided with your Red Hat Enterprise Linux distribution.



Note

The system clocks in GFS2 nodes must be within a few minutes of each other to prevent unnecessary inode time-stamp updating. Unnecessary inode time-stamp updating severely impacts cluster performance.

- In order to use GFS2 in a clustered environment, you must configure your system to use the Clustered Logical Volume Manager (CLVM), a set of clustering extensions to the LVM Logical Volume Manager. In order to use CLVM, the Red Hat Cluster Suite software, including the **clvmd** daemon, must be running. For information on using CLVM, see *Logical Volume Manager Administration*. For information on installing and administering Red Hat Cluster Suite, see *Cluster Administration*.

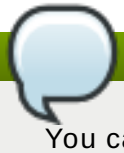
3.2. Initial Setup Tasks

Initial GFS2 setup consists of the following tasks:

1. Setting up logical volumes.
2. Making a GFS2 files system.
3. Mounting file systems.

Follow these steps to set up GFS2 initially.

1. Using LVM, create a logical volume for each Red Hat GFS2 file system.



Note

You can use **init.d** scripts included with Red Hat Cluster Suite to automate activating and deactivating logical volumes. For more information about **init.d** scripts, see *Configuring and Managing a Red Hat Cluster*.

2. Create GFS2 file systems on logical volumes created in Step 1. Choose a unique name for each file system. For more information about creating a GFS2 file system, see Section 4.1, “Making a File System”.

   You can use either of the following formats to create a clustered GFS2 file system:

   ```
   mkfs.gfs2 -p lock_dlm -t ClusterName:FSName -j NumberJournals BlockDevice
   ```

   ```
   mkfs -t gfs2 -p lock_dlm -t LockTableName -j NumberJournals BlockDevice
   ```

   For more information on creating a GFS2 file system, see Section 4.1, “Making a File System”.

3. At each node, mount the GFS2 file systems. For more information about mounting a GFS2 file system, see Section 4.2, “Mounting a File System”.

   Command usage:

   mount *BlockDevice MountPoint*

   mount -o acl *BlockDevice MountPoint*

   The **-o acl** mount option allows manipulating file ACLs. If a file system is mounted without the **-o acl** mount option, users are allowed to view ACLs (with **getfacl**), but are not allowed to set them (with **setfacl**).

   Note

   You can use **init.d** scripts included with the Red Hat High Availability Add-On to automate mounting and unmounting GFS2 file systems.

Chapter 4. Managing GFS2

This chapter describes the tasks and commands for managing GFS2 and consists of the following sections:

4.1. Making a File System

You create a GFS2 file system with the **mkfs.gfs2** command. You can also use the **mkfs** command with the **-t gfs2** option specified. A file system is created on an activated LVM volume. The following information is required to run the **mkfs.gfs2** command:

- Lock protocol/module name (the lock protocol for a cluster is **lock_dlm**)
- Cluster name (when running as part of a cluster configuration)
- Number of journals (one journal required for each node that may be mounting the file system)

When creating a GFS2 file system, you can use the **mkfs.gfs2** command directly, or you can use the **mkfs** command with the **-t** parameter specifying a file system of type **gfs2**, followed by the gfs2 file system options.

Note

Once you have created a GFS2 file system with the **mkfs.gfs2** command, you cannot decrease the size of the file system. You can, however, increase the size of an existing file system with the **gfs2_grow** command, as described in Section 4.6, “Growing a File System”.

Usage

When creating a clustered GFS2 file system, you can use either of the following formats:

```
mkfs.gfs2 -p LockProtoName -t LockTableName -j NumberJournals BlockDevice
```

```
mkfs -t gfs2 -p LockProtoName -t LockTableName -j NumberJournals
BlockDevice
```

When creating a local GFS2 file system, you can use either of the following formats:

Note

For the Red Hat Enterprise Linux 6 release, Red Hat does not support the use of GFS2 as a single-node file system.

```
mkfs.gfs2 -p LockProtoName -j NumberJournals BlockDevice
```

```
mkfs -t gfs2 -p LockProtoName -j NumberJournals BlockDevice
```

Warning

Make sure that you are very familiar with using the ***LockProtoName*** and ***LockTableName*** parameters. Improper use of the ***LockProtoName*** and ***LockTableName*** parameters may cause file system or lock space corruption.

LockProtoName

Specifies the name of the locking protocol to use. The lock protocol for a cluster is **lock_dlm**.

LockTableName

This parameter is specified for GFS2 file system in a cluster configuration. It has two parts separated by a colon (no spaces) as follows: ***ClusterName:FSName***

- ***ClusterName***, the name of the cluster for which the GFS2 file system is being created.
- ***FSName***, the file system name, can be 1 to 16 characters long. The name must be unique for all **lock_dlm** file systems over the cluster, and for all file systems (**lock_dlm** and **lock_nolock**) on each local node.

Number

Specifies the number of journals to be created by the **mkfs.gfs2** command. One journal is required for each node that mounts the file system. For GFS2 file systems, more journals can be added later without growing the file system, as described in Section 4.7, “Adding Journals to a File System”.

BlockDevice

Specifies a logical or physical volume.

Examples

In these example, **lock_dlm** is the locking protocol that the file system uses, since this is a clustered file system. The cluster name is **alpha**, and the file system name is **mydata1**. The file system contains eight journals and is created on **/dev/vg01/lvol0**.

```
mkfs.gfs2 -p lock_dlm -t alpha:mydata1 -j 8 /dev/vg01/lvol0
```

```
mkfs -t gfs2 -p lock_dlm -t alpha:mydata1 -j 8 /dev/vg01/lvol0
```

In these examples, a second **lock_dlm** file system is made, which can be used in cluster **alpha**. The file system name is **mydata2**. The file system contains eight journals and is created on **/dev/vg01/lvol1**.

```
mkfs.gfs2 -p lock_dlm -t alpha:mydata2 -j 8 /dev/vg01/lvol1
```

```
mkfs -t gfs2 -p lock_dlm -t alpha:mydata2 -j 8 /dev/vg01/lvol1
```

Complete Options

Table 4.1, "Command Options: **mkfs.gfs2**" describes the **mkfs.gfs2** command options (flags and parameters).

Table 4.1. Command Options: mkfs.gfs2

Flag	Parameter	Description
-c	***Megabytes***	Sets the initial size of each journal's quota change file to ***Megabytes***.
-D		Enables debugging output.
-h		Help. Displays available options.
-J	***MegaBytes***	Specifies the size of the journal in megabytes. Default journal size is 128 megabytes. The minimum size is 8 megabytes. Larger journals improve performance, although they use more memory than smaller journals.
-j	***Number***	Specifies the number of journals to be created by the **mkfs.gfs2** command. One journal is required for each node that mounts the file system. If this option is not specified, one journal will be created. For GFS2 file systems, you can add additional journals at a later time without growing the file system.
-O		Prevents the **mkfs.gfs2** command from asking for confirmation before writing the file system.
-p	***LockProtoName***	Specifies the name of the locking protocol to use. Recognized locking protocols include: **lock_dlm** — The standard locking module, required for a clustered file system. **lock_nolock** — Used when GFS2 is acting as a local file system (one node only).
-q		Quiet. Do not display anything.

Flag	Parameter	Description
-r	***MegaBytes***	Specifies the size of the resource groups in megabytes. The minimum resource group size is 32 MB. The maximum resource group size is 2048 MB. A large resource group size may increase performance on very large file systems. If this is not specified, mkfs.gfs2 chooses the resource group size based on the size of the file system: average size file systems will have 256 MB resource groups, and bigger file systems will have bigger RGs for better performance.
-t	***LockTableName***	A unique identifier that specifies the lock table field when you use the **lock_dlm** protocol; the **lock_nolock** protocol does not use this parameter. This parameter has two parts separated by a colon (no spaces) as follows: ***ClusterName:FSName***. ***ClusterName*** is the name of the cluster for which the GFS2 file system is being created; only members of this cluster are permitted to use this file system. The cluster name is set in the **/etc/cluster/cluster.conf** file via the **Cluster Configuration Tool** and displayed at the **Cluster Status Tool** in the Red Hat Cluster Suite cluster management GUI. ***FSName***, the file system name, can be 1 to 16 characters in length, and the name must be unique among all file systems in the cluster.
-u	***MegaBytes***	Specifies the initial size of each journal's unlinked tag file.
-V		Displays command version information.

4.2. Mounting a File System

Before you can mount a GFS2 file system, the file system must exist (see Section 4.1, “Making a File System”), the volume where the file system exists must be activated, and the supporting clustering and locking systems must be started (see *Configuring and Managing a Red Hat Cluster*). After those requirements have been met, you can mount the GFS2 file system as you would any Linux file system.

Note

Attempting to mount a GFS2 file system when the Cluster Manager (**cman**) has not been started produces the following error message:

```
[root@gfs-a24c-01 ~]# mount -t gfs2 -o noatime /dev/mapper/mpathap1
/mnt
gfs_controld join connect error: Connection refused
error mounting lockproto lock_dlm
```

To manipulate file ACLs, you must mount the file system with the **-o acl** mount option. If a file system is mounted without the **-o acl** mount option, users are allowed to view ACLs (with **getfacl**), but are not allowed to set them (with **setfacl**).

Usage

Mounting Without ACL Manipulation

```
mount BlockDevice MountPoint
```

Mounting With ACL Manipulation

```
mount -o acl BlockDevice MountPoint
```

-o acl

GFS2-specific option to allow manipulating file ACLs.

BlockDevice

Specifies the block device where the GFS2 file system resides.

MountPoint

Specifies the directory where the GFS2 file system should be mounted.

Example

In this example, the GFS2 file system on **/dev/vg01/lvol0** is mounted on the **/mygfs2** directory.

```
mount /dev/vg01/lvol0 /mygfs2
```

Complete Usage

```
mount BlockDevice MountPoint -o option
```

The **-o option** argument consists of GFS2-specific options (see Table 4.2, “GFS2-Specific Mount Options”) or acceptable standard Linux **mount -o** options, or a combination of both. Multiple ***option*** parameters are separated by a comma and no spaces.

Note

The **mount** command is a Linux system command. In addition to using GFS2-specific options described in this section, you can use other, standard, **mount** command options (for example, **-r**). For information about other Linux **mount** command options, see the Linux **mount** man page.

Table 4.2, "GFS2-Specific Mount Options" describes the available GFS2-specific **-o option** values that can be passed to GFS2 at mount time.

Note

This table includes descriptions of options that are used with local file systems only. Note, however, that for the Red Hat Enterprise Linux 6 release, Red Hat does not support the use of GFS2 as a single-node file system. Red Hat will continue to support single-node GFS2 file systems for mounting snapshots of cluster file systems (for example, for backup purposes).

Table 4.2. GFS2-Specific Mount Options

Option	Description
acl	Allows manipulating file ACLs. If a file system is mounted without the **acl** mount option, users are allowed to view ACLs (with **getfacl**), but are not allowed to set them (with **setfacl**).
data=[ordered|writeback]	When **data=ordered** is set, the user data modified by a transaction is flushed to the disk before the transaction is committed to disk. This should prevent the user from seeing uninitialized blocks in a file after a crash. When **data=writeback** mode is set, the user data is written to the disk at any time after it is dirtied; this does not provide the same consistency guarantee as **ordered** mode, but it should be slightly faster for some workloads. The default value is **ordered** mode.
ignore_local_fs **Caution:** This option should *not* be used when GFS2 file systems are shared.	Forces GFS2 to treat the file system as a multihost file system. By default, using **lock_nolock** automatically turns on the **localflocks** flag.
localflocks **Caution:** This option should not be used when GFS2 file systems are shared.	Tells GFS2 to let the VFS (virtual file system) layer do all flock and fcntl. The **localflocks** flag is automatically turned on by **lock_nolock**.
lockproto=*LockModuleName*	Allows the user to specify which locking protocol to use with the file system. If ***LockModuleName*** is not specified, the locking protocol name is read from the file system superblock.
locktable=*LockTableName*	Allows the user to specify which locking table to use with the file system.

Option	Description
quota=[off/account/on]	Turns quotas on or off for a file system. Setting the quotas to be in the **account** state causes the per UID/GID usage statistics to be correctly maintained by the file system; limit and warn values are ignored. The default value is **off**.
errors=panic|withdraw	When **errors=panic** is specified, file system errors will cause a kernel panic. The default behavior, which is the same as specifying **errors=withdraw**, is for the system to withdraw from the file system and make it inaccessible until the next reboot; in some cases the system may remain running. For information on the GFS2 withdraw function, see Section 4.14, "The GFS2 Withdraw Function".
discard/nodiscard	Causes GFS2 to generate "discard" I/O requests for blocks that have been freed. These can be used by suitable hardware to implement thin provisioning and similar schemes.
barrier/nobarrier	Causes GFS2 to send I/O barriers when flushing the journal. The default value is **on**. This option is automatically turned **off** if the underlying device does not support I/O barriers. Use of I/O barriers with GFS2 is highly recommended at all times unless the block device is designed so that it cannot lose its write cache content (for example, if it is on a UPS or it does not have a write cache).
quota_quantum=*secs*	Sets the number of seconds for which a change in the quota information may sit on one node before being written to the quota file. This is the preferred way to set this parameter. The value is an integer number of seconds greater than zero. The default is 60 seconds. Shorter settings result in faster updates of the lazy quota information and less likelihood of someone exceeding their quota. Longer settings make file system operations involving quotas faster and more efficient.
statfs_quantum=*secs*	Setting **statfs_quantum** to 0 is the preferred way to set the slow version of **statfs**. The default value is 30 secs which sets the maximum time period before **statfs** changes will be synced to the master **statfs** file. This can be adjusted to allow for faster, less accurate **statfs** values or slower more accurate values. When this option is set to 0, **statfs** will always report the true values.
statfs_percent=*value*	Provides a bound on the maximum percentage change in the **statfs** information on a local basis before it is synced back to the master **statfs** file, even if the time period has not expired. If the setting of **statfs_quantum** is 0, then this setting is ignored.

4.3. Unmounting a File System

The GFS2 file system can be unmounted the same way as any Linux file system — by using the **umount** command.

Note

The **umount** command is a Linux system command. Information about this command can be found in the Linux **umount** command man pages.

Usage

```
umount MountPoint
```

MountPoint

Specifies the directory where the GFS2 file system is currently mounted.

4.4. Special Considerations when Mounting GFS2 File Systems

GFS2 file systems that have been mounted manually rather than automatically through an entry in the **fstab** file will not be known to the system when file systems are unmounted at system shutdown. As a result, the GFS2 script will not unmount the GFS2 file system. After the GFS2 shutdown script is run, the standard shutdown process kills off all remaining user processes, including the cluster infrastructure, and tries to unmount the file system. This unmount will fail without the cluster infrastructure and the system will hang.

To prevent the system from hanging when the GFS2 file systems are unmounted, you should do one of the following:

- Always use an entry in the **fstab** file to mount the GFS2 file system.
- If a GFS2 file system has been mounted manually with the **mount** command, be sure to unmount the file system manually with the **umount** command before rebooting or shutting down the system.

If your file system hangs while it is being unmounted during system shutdown under these circumstances, perform a hardware reboot. It is unlikely that any data will be lost since the file system is synced earlier in the shutdown process.

4.5. GFS2 Quota Management

File-system quotas are used to limit the amount of file system space a user or group can use. A user or group does not have a quota limit until one is set. When a GFS2 file system is mounted with the **quota=on** or **quota=account** option, GFS2 keeps track of the space used by each user and group even when there are no limits in place. GFS2 updates quota information in a transactional way so system crashes do not require quota usages to be reconstructed.

To prevent a performance slowdown, a GFS2 node synchronizes updates to the quota file only periodically. The fuzzy quota accounting can allow users or groups to slightly exceed the set limit. To minimize this, GFS2 dynamically reduces the synchronization period as a hard quota limit is approached.

Note

As of the Red Hat Enterprise Linux 6.1 release, GFS2 supports the standard Linux quota facilities. In order to use this you will need to install the **quota** RPM. This is the preferred way to administer quotas on GFS2 and should be used for all new deployments of GFS2 using quotas. This section documents GFS2 quota management using these facilities.

For earlier releases of Red Hat Enterprise Linux, GFS2 required the **gfs2_quota** command to manage quotas. For information on using the **gfs2_quota** command, see Appendix A, *GFS2 Quota Management with the **gfs2_quota** Command*.

4.5.1. Configuring Disk Quotas

To implement disk quotas, use the following steps:

1. Set up quotas in enforcement or accounting mode.
2. Initialize the quota database file with current block usage information.
3. Assign quota policies. (In accounting mode, these policies are not enforced.)

Each of these steps is discussed in detail in the following sections.

4.5.1.1. Setting Up Quotas in Enforcement or Accounting Mode

In GFS2 file systems, quotas are disabled by default. To enable quotas for a file system, mount the file system with the **quota=on** option specified.

It is possible to keep track of disk usage and maintain quota accounting for every user and group without enforcing the limit and warn values. To do this, mount the file system with the **quota=account** option specified.

Usage

To mount a file system with quotas enabled, mount the file system with the **quota=on** option specified.

```
mount -o quota=on BlockDevice MountPoint
```

To mount a file system with quota accounting maintained, even though the quota limits are not enforced, mount the file system with the **quota=account** option specified.

```
mount -o quota=account BlockDevice MountPoint
```

To mount a file system with quotas disabled, mount the file system with the **quota=off** option specified. This is the default setting.

```
mount -o quota=off BlockDevice MountPoint
```

quota={on|off|account}

on - Specifies that quotas are enabled when the file system is mounted.

off - Specifies that quotas are disabled when the file system is mounted.

account - Specifies that user and group usage statistics are maintained by the file system, even though the quota limits are not enforced.

BlockDevice

Specifies the block device where the GFS2 file system resides.

MountPoint

Specifies the directory where the GFS2 file system should be mounted.

Examples

In this example, the GFS2 file system on **/dev/vg01/lvol0** is mounted on the **/mygfs2** directory with quotas enabled.

```
mount -o quota=on /dev/vg01/lvol0 /mygfs2
```

In this example, the GFS2 file system on **/dev/vg01/lvol0** is mounted on the **/mygfs2** directory with quota accounting maintained, but not enforced.

```
mount -o quota=account /dev/vg01/lvol0 /mygfs2
```

4.5.1.2. Creating the Quota Database Files

After each quota-enabled file system is mounted, the system is capable of working with disk quotas. However, the file system itself is not yet ready to support quotas. The next step is to run the **quotacheck** command.

The **quotacheck** command examines quota-enabled file systems and builds a table of the current disk usage per file system. The table is then used to update the operating system's copy of disk usage. In addition, the file system's disk quota files are updated.

To create the quota files on the file system, use the **-u** and the **-g** options of the **quotacheck** command; both of these options must be specified for user and group quotas to be initialized. For example, if quotas are enabled for the **/home** file system, create the files in the **/home** directory:

```
quotacheck -ug /home
```

4.5.1.3. Assigning Quotas per User

The last step is assigning the disk quotas with the **edquota** command. Note that if you have mounted your file system in accounting mode (with the **quota=account** option specified), the quotas are not enforced.

To configure the quota for a user, as root in a shell prompt, execute the command:

```
edquota username
```

Perform this step for each user who needs a quota. For example, if a quota is enabled in **/etc/fstab** for the **/home** partition (**/dev/VolGroup00/LogVol02** in the example below) and the command **edquota testuser** is executed, the following is shown in the editor configured as the default for the system:

```
Disk quotas for user testuser (uid 501):
Filesystem                blocks     soft     hard    inodes   soft
hard
/dev/VolGroup00/LogVol02  440436        0        0
```

Note

The text editor defined by the **EDITOR** environment variable is used by **edquota**. To change the editor, set the **EDITOR** environment variable in your **~/.bash_profile** file to the full path of the editor of your choice.

The first column is the name of the file system that has a quota enabled for it. The second column shows how many blocks the user is currently using. The next two columns are used to set soft and hard block limits for the user on the file system.

The soft block limit defines the maximum amount of disk space that can be used.

The hard block limit is the absolute maximum amount of disk space that a user or group can use. Once this limit is reached, no further disk space can be used.

The GFS2 file system does not maintain quotas for inodes, so these columns do not apply to GFS2 file systems and will be blank.

If any of the values are set to 0, that limit is not set. In the text editor, change the desired limits. For example:

```
Disk quotas for user testuser (uid 501):
Filesystem                blocks     soft     hard    inodes   soft
hard
/dev/VolGroup00/LogVol02  440436   500000   550000
```

To verify that the quota for the user has been set, use the command:

```
quota testuser
```

4.5.1.4. Assigning Quotas per Group

Quotas can also be assigned on a per-group basis. Note that if you have mounted your file system in accounting mode (with the **account=on** option specified), the quotas are not enforced.

To set a group quota for the **devel** group (the group must exist prior to setting the group quota), use the following command:

```
edquota -g devel
```

This command displays the existing quota for the group in the text editor:

```
Disk quotas for group devel (gid 505):
Filesystem                blocks    soft     hard    inodes   soft   hard
/dev/VolGroup00/LogVol02  440400       0        0
```

The GFS2 file system does not maintain quotas for inodes, so these columns do not apply to GFS2 file systems and will be blank. Modify the limits, then save the file.

To verify that the group quota has been set, use the following command:

```
quota -g devel
```

4.5.2. Managing Disk Quotas

If quotas are implemented, they need some maintenance — mostly in the form of watching to see if the quotas are exceeded and making sure the quotas are accurate.

Of course, if users repeatedly exceed their quotas or consistently reach their soft limits, a system administrator has a few choices to make depending on what type of users they are and how much disk space impacts their work. The administrator can either help the user determine how to use less disk space or increase the user's disk quota.

You can create a disk usage report by running the **repquota** utility. For example, the command **repquota /home** produces this output:

```
*** Report for user quotas on device /dev/mapper/VolGroup00-LogVol02
Block grace time: 7days; Inode grace time: 7days
   Block limits   File limits
User  used soft hard grace used soft hard grace
----------------------------------------------------------------------
root      --      36       0       0              4     0     0
kristin   --     540       0       0            125     0     0
testuser  --  440400  500000  550000          37418     0     0
```

To view the disk usage report for all (option **-a**) quota-enabled file systems, use the command:

```
repquota -a
```

While the report is easy to read, a few points should be explained. The **--** displayed after each user is a quick way to determine whether the block limits have been exceeded. If the block soft limit is exceeded, a **+** appears in place of the first **-** in the output. The second **-** indicates the inode limit, but GFS2 file systems do not support inode limits so that character will remain as **-**. GFS2 file systems do not support a grace period, so the **grace** column will remain blank.

Note that the **repquota** command is not supported over NFS, irrespective of the underlying file system.

4.5.3. Keeping Quotas Accurate

If you enable quotas on your file system after a period of time when you have been running with quotas disabled, you should run the **quotacheck** command to create, check, and repair quota files. Additionally, you may want to run the **quotacheck** if you think your quota files may not be accurate, as may occur when a file system is not unmounted cleanly after a system crash.

For more information about the **quotacheck** command, see the **quotacheck** man page.

Note

Run **quotacheck** when the file system is relatively idle on all nodes because disk activity may affect the computed quota values.

4.5.4. Synchronizing Quotas with the quotasync Command

GFS2 stores all quota information in its own internal file on disk. A GFS2 node does not update this quota file for every file system write; rather, by default it updates the quota file once every 60 seconds. This is necessary to avoid contention among nodes writing to the quota file, which would cause a slowdown in performance.

As a user or group approaches their quota limit, GFS2 dynamically reduces the time between its quota-file updates to prevent the limit from being exceeded. The normal time period between quota synchronizations is a tunable parameter, **quota_quantum**. You can change this from its default value of 60 seconds using the **quota_quantum=** mount option, as described in Table 4.2, "GFS2-Specific Mount Options". The **quota_quantum** parameter must be set on each node and each time the file system is mounted. Changes to the **quota_quantum** parameter are not persistent across unmounts. You can update the **quota_quantum** value with the **mount -o remount**.

You can use the **quotasync** command to synchronize the quota information from a node to the on-disk quota file between the automatic updates performed by GFS2.

Usage

Synchronizing Quota Information

```
quotasync [-ug] -a|mntpnt...
```

u

Sync the user quota files.

g

Sync the group quota files

a

Sync all file systems that are currently quota-enabled and support sync. When -a is absent, a file system mountpoint should be specified.

mntpnt

Specifies the GFS2 file system to which the actions apply.

Tuning the Time Between Synchronizations

```
mount -o quota_quantum=secs,remount BlockDevice MountPoint
```

MountPoint

Specifies the GFS2 file system to which the actions apply.

secs

Specifies the new time period between regular quota-file synchronizations by GFS2. Smaller values may increase contention and slow down performance.

Examples

This example synchronizes all the cached dirty quotas from the node it is run on to the ondisk quota file for the file system **/mnt/mygfs2**.

```
# quotasync -ug /mnt/mygfs2
```

This example changes the default time period between regular quota-file updates to one hour (3600 seconds) for file system **/mnt/mygfs2** when remounting that file system on logical volume **/dev/volgroup/logical_volume**.

```
# mount -o quota_quantum=3600,remount /dev/volgroup/logical_volume
/mnt/mygfs2
```

4.5.5. References

For more information on disk quotas, see the **man** pages of the following commands:

- **quotacheck**
- **edquota**
- **repquota**
- **quota**

4.6. Growing a File System

The **gfs2_grow** command is used to expand a GFS2 file system after the device where the file system resides has been expanded. Running a **gfs2_grow** command on an existing GFS2 file system fills all spare space between the current end of the file system and the end of the device with a newly initialized GFS2 file system extension. When the fill operation is completed, the resource index for the file system is updated. All nodes in the cluster can then use the extra storage space that has been added.

The **gfs2_grow** command must be run on a mounted file system, but only needs to be run on one node in a cluster. All the other nodes sense that the expansion has occurred and automatically start using the new space.

Note

Once you have created a GFS2 file system with the **mkfs.gfs2** command, you cannot decrease the size of the file system.

Usage

```
gfs2_grow MountPoint
```

MountPoint

Specifies the GFS2 file system to which the actions apply.

Comments

Before running the **gfs2_grow** command:

- Back up important data on the file system.
- Determine the volume that is used by the file system to be expanded by running a **df** ***MountPoint*** command.
- Expand the underlying cluster volume with LVM. For information on administering LVM volumes, see *Logical Volume Manager Administration*.

After running the **gfs2_grow** command, run a **df** command to check that the new space is now available in the file system.

Examples

In this example, the file system on the **/mygfs2fs** directory is expanded.

```
[root@dash-01 ~]# gfs2_grow /mygfs2fs
FS: Mount Point: /mygfs2fs
FS: Device:      /dev/mapper/gfs2testvg-gfs2testlv
FS: Size:        524288 (0x80000)
FS: RG size:     65533 (0xfffd)
DEV: Size:       655360 (0xa0000)
The file system grew by 512MB.
gfs2_grow complete.
```

Complete Usage

```
gfs2_grow [Options] {MountPoint | Device} [MountPoint | Device]
```

MountPoint

Specifies the directory where the GFS2 file system is mounted.

Device

Specifies the device node of the file system.

Table 4.3, “GFS2-specific Options Available While Expanding A File System” describes the GFS2-specific options that can be used while expanding a GFS2 file system.

Table 4.3. GFS2-specific Options Available While Expanding A File System

Option	Description
-h	Help. Displays a short usage message.
-q	Quiet. Turns down the verbosity level.
-r MegaBytes	Specifies the size of the new resource group. The default size is 256MB.
-T	Test. Do all calculations, but do not write any data to the disk and do not expand the file system.
-V	Displays command version information.

4.7. Adding Journals to a File System

The **gfs2_jadd** command is used to add journals to a GFS2 file system. You can add journals to a GFS2 file system dynamically at any point without expanding the underlying logical volume. The **gfs2_jadd** command must be run on a mounted file system, but it needs to be run on only one node in the cluster. All the other nodes sense that the expansion has occurred.

Note

If a GFS2 file system is full, the **gfs2_jadd** will fail, even if the logical volume containing the file system has been extended and is larger than the file system. This is because in a GFS2 file system, journals are plain files rather than embedded metadata, so simply extending the underlying logical volume will not provide space for the journals.

Before adding journals to a GFS file system, you can use the **journals** option of the **gfs2_tool** to find out how many journals the GFS2 file system currently contains. The following example displays the number and size of the journals in the file system mounted at **/mnt/gfs2**.

```
[root@roth-01 ../cluster/gfs2]# gfs2_tool journals /mnt/gfs2
journal2 - 128MB
journal1 - 128MB
journal0 - 128MB
3 journal(s) found.
```

Usage

```
gfs2_jadd -j Number MountPoint
```

Number

Specifies the number of new journals to be added.

MountPoint

Specifies the directory where the GFS2 file system is mounted.

Examples

In this example, one journal is added to the file system on the **/mygfs2** directory.

```
gfs2_jadd -j1 /mygfs2
```

In this example, two journals are added to the file system on the **/mygfs2** directory.

```
gfs2_jadd -j2 /mygfs2
```

Complete Usage

```
gfs2_jadd [Options] {MountPoint | Device} [MountPoint | Device]
```

MountPoint

Specifies the directory where the GFS2 file system is mounted.

Device

Specifies the device node of the file system.

Table 4.4, “GFS2-specific Options Available When Adding Journals” describes the GFS2-specific options that can be used when adding journals to a GFS2 file system.

Table 4.4. GFS2-specific Options Available When Adding Journals

Flag	Parameter	Description
-h		Help. Displays short usage message.
-J	***MegaBytes***	Specifies the size of the new journals in megabytes. Default journal size is 128 megabytes. The minimum size is 32 megabytes. To add journals of different sizes to the file system, the **gfs2_jadd** command must be run for each size journal. The size specified is rounded down so that it is a multiple of the journal-segment size that was specified when the file system was created.
-j	***Number***	Specifies the number of new journals to be added by the **gfs2_jadd** command. The default value is 1.
-q		Quiet. Turns down the verbosity level.
-V		Displays command version information.

4.8. Data Journaling

Ordinarily, GFS2 writes only metadata to its journal. File contents are subsequently written to disk by the kernel's periodic sync that flushes file system buffers. An **fsync()** call on a file causes the file's data to be written to disk immediately. The call returns when the disk reports that all data is safely written.

Data journaling can result in a reduced **fsync()** time for very small files because the file data is written to the journal in addition to the metadata. This advantage rapidly reduces as the file size increases. Writing to medium and larger files will be much slower with data journaling turned on.

Applications that rely on **fsync()** to sync file data may see improved performance by using data journaling. Data journaling can be enabled automatically for any GFS2 files created in a flagged directory (and all its subdirectories). Existing files with zero length can also have data journaling turned on or off.

Enabling data journaling on a directory sets the directory to "inherit jdata", which indicates that all files and directories subsequently created in that directory are journaled. You can enable and disable data journaling on a file with the **chattr** command.

The following commands enable data journaling on the **/mnt/gfs2/gfs2_dir/newfile** file and then check whether the flag has been set properly.

```
[root@roth-01 ~]# chattr +j /mnt/gfs2/gfs2_dir/newfile
[root@roth-01 ~]# lsattr /mnt/gfs2/gfs2_dir
---------j--- /mnt/gfs2/gfs2_dir/newfile
```

The following commands disable data journaling on the **/mnt/gfs2/gfs2_dir/newfile** file and then check whether the flag has been set properly.

```
[root@roth-01 ~]# chattr -j /mnt/gfs2/gfs2_dir/newfile
[root@roth-01 ~]# lsattr /mnt/gfs2/gfs2_dir
------------ /mnt/gfs2/gfs2_dir/newfile
```

You can also use the **chattr** command to set the **j** flag on a directory. When you set this flag for a directory, all files and directories subsequently created in that directory are journaled. The following set of commands sets the **j** flag on the **gfs2_dir** directory, then checks whether the flag has been set properly. After this, the commands create a new file called **newfile** in the **/mnt/gfs2/gfs2_dir** directory and then check whether the **j** flag has been set for the file. Since the **j** flag is set for the directory, then **newfile** should also have journaling enabled.

```
[root@roth-01 ~]# chattr -j /mnt/gfs2/gfs2_dir
[root@roth-01 ~]# lsattr /mnt/gfs2
---------j--- /mnt/gfs2/gfs2_dir
[root@roth-01 ~]# touch /mnt/gfs2/gfs2_dir/newfile
[root@roth-01 ~]# lsattr /mnt/gfs2/gfs2_dir
---------j--- /mnt/gfs2/gfs2_dir/newfile
```

4.9. Configuring atime Updates

Each file inode and directory inode has three time stamps associated with it:

- **ctime** — The last time the inode status was changed
- **mtime** — The last time the file (or directory) data was modified
- **atime** — The last time the file (or directory) data was accessed

If **atime** updates are enabled as they are by default on GFS2 and other Linux file systems then every time a file is read, its inode needs to be updated.

Because few applications use the information provided by **atime**, those updates can require a significant amount of unnecessary write traffic and file locking traffic. That traffic can degrade performance; therefore, it may be preferable to turn off or reduce the frequency of **atime** updates.

Two methods of reducing the effects of **atime** updating are available:

- Mount with **relatime** (relative atime), which updates the **atime** if the previous **atime** update is older than the **mtime** or **ctime** update.
- Mount with **noatime**, which disables **atime** updates on that file system.

4.9.1. Mount with relatime

The **relatime** (relative atime) Linux mount option can be specified when the file system is mounted. This specifies that the **atime** is updated if the previous **atime** update is older than the **mtime** or **ctime** update.

Usage

```
mount  BlockDevice MountPoint -o relatime
```

BlockDevice

Specifies the block device where the GFS2 file system resides.

MountPoint

Specifies the directory where the GFS2 file system should be mounted.

Example

In this example, the GFS2 file system resides on the **/dev/vg01/lvol0** and is mounted on directory **/mygfs2**. The **atime** updates take place only if the previous **atime** update is older than the **mtime** or **ctime** update.

```
mount /dev/vg01/lvol0 /mygfs2 -o relatime
```

4.9.2. Mount with noatime

The **noatime** Linux mount option can be specified when the file system is mounted, which disables **atime** updates on that file system.

Usage

```
mount BlockDevice MountPoint -o noatime
```

BlockDevice

Specifies the block device where the GFS2 file system resides.

MountPoint

Specifies the directory where the GFS2 file system should be mounted.

Example

In this example, the GFS2 file system resides on the **/dev/vg01/lvol0** and is mounted on directory **/mygfs2** with **atime** updates turned off.

```
mount /dev/vg01/lvol0 /mygfs2 -o noatime
```

4.10. Suspending Activity on a File System

You can suspend write activity to a file system by using the **dmsetup suspend** command. Suspending write activity allows hardware-based device snapshots to be used to capture the file system in a consistent state. The **dmsetup resume** command ends the suspension.

Usage

Start Suspension

```
dmsetup suspend MountPoint
```

End Suspension

```
dmsetup resume MountPoint
```

MountPoint

Specifies the file system.

Examples

This example suspends writes to file system **/mygfs2**.

```
# dmsetup suspend /mygfs2
```

This example ends suspension of writes to file system **/mygfs2**.

```
# dmsetup resume /mygfs2
```

4.11. Repairing a File System

When nodes fail with the file system mounted, file system journaling allows fast recovery. However, if a storage device loses power or is physically disconnected, file system corruption may occur. (Journaling cannot be used to recover from storage subsystem failures.) When that type of corruption occurs, you can recover the GFS2 file system by using the **fsck.gfs2** command.

Important

The **fsck.gfs2** command must be run only on a file system that is unmounted from all nodes.

Important

You should not check a GFS2 file system at boot time with the **fsck.gfs2** command. The **fsck.gfs2** command cannot determine at boot time whether the file system is mounted by another node in the cluster. You should run the **fsck.gfs2** command manually only after the system boots.

To ensure that the **fsck.gfs2** command does not run on a GFS2 file system at boot time, modify the **/etc/fstab** file so that the final two columns for a GFS2 file system mount point show "0 0" rather than "1 1" (or any other numbers), as in the following example:

```
/dev/VG12/lv_svr_home    /svr_home        gfs2
defaults,noatime,nodiratime,noquota     0 0
```

Note

If you have previous experience using the gfs_fsck command on GFS file systems, note that the **fsck.gfs2** command differs from some earlier releases of **gfs_fsck** in the following ways:

- Pressing **Ctrl**+**C** while running the **fsck.gfs2** interrupts processing and displays a prompt asking whether you would like to abort the command, skip the rest of the current pass, or continue processing.
- You can increase the level of verbosity by using the **-v** flag. Adding a second **-v** flag increases the level again.
- You can decrease the level of verbosity by using the **-q** flag. Adding a second **-q** flag decreases the level again.
- The **-n** option opens a file system as read-only and answers **no** to any queries automatically. The option provides a way of trying the command to reveal errors without actually allowing the **fsck.gfs2** command to take effect.

Refer to the **fsck.gfs2** man page for additional information about other command options.

Running the **fsck.gfs2** command requires system memory above and beyond the memory used for the operating system and kernel. Each block of memory in the GFS2 file system itself requires approximately five bits of additional memory, or 5/8 of a byte. So to estimate how many bytes of memory you will need to run the **fsck.gfs2** command on your file system, determine how many blocks the file system contains and multiply that number by 5/8.

For example, to determine approximately how much memory is required to run the **fsck.gfs2** command on a GFS2 file system that is 16TB with a block size of 4K, first determine how many blocks of memory the file system contains by dividing 16Tb by 4K:

```
17592186044416 / 4096 = 4294967296
```

Since this file system contains 4294967296 blocks, multiply that number by 5/8 to determine how many bytes of memory are required:

```
4294967296 * 5/8 = 2684354560
```

This file system requires approximately 2.6GB of free memory to run the **fsck.gfs2** command. Note that if the block size was 1K, running the **fsck.gfs2** command would require four times the memory, or approximately 11GB.

Usage

```
fsck.gfs2 -y BlockDevice
```

-y

The **-y** flag causes all questions to be answered with **yes**. With the **-y** flag specified, the **fsck.gfs2** command does not prompt you for an answer before making changes.

BlockDevice

Specifies the block device where the GFS2 file system resides.

Example

In this example, the GFS2 file system residing on block device **/dev/testvol/testlv** is repaired. All queries to repair are automatically answered with **yes**.

```
[root@dash-01 ~]# fsck.gfs2 -y /dev/testvg/testlv
Initializing fsck
Validating Resource Group index.
Level 1 RG check.
(level 1 passed)
Clearing journals (this may take a while)...
Journals cleared.
Starting pass1
Pass1 complete
Starting pass1b
Pass1b complete
Starting pass1c
Pass1c complete
Starting pass2
Pass2 complete
Starting pass3
Pass3 complete
Starting pass4
Pass4 complete
Starting pass5
Pass5 complete
Writing changes to disk
fsck.gfs2 complete
```

4.12. Bind Mounts and Context-Dependent Path Names

GFS2 file systems do not provide support for Context-Dependent Path Names (CDPNs), which allow you to create symbolic links that point to variable destination files or directories. For this functionality in GFS2, you can use the **bind** option of the **mount** command.

The **bind** option of the **mount** command allows you to remount part of a file hierarchy at a different location while it is still available at the original location. The format of this command is as follows.

```
mount --bind olddir newdir
```

After executing this command, the contents of the ***olddir*** directory are available at two locations: ***olddir*** and ***newdir***. You can also use this option to make an individual file available at two locations.

For example, after executing the following commands the contents of **/root/tmp** will be identical to the contents of the previously mounted **/var/log** directory.

```
[root@menscryfa ~]# cd ~root
[root@menscryfa ~]# mkdir ./tmp
[root@menscryfa ~]# mount --bind /var/log /root/tmp
```

Alternately, you can use an entry in the **/etc/fstab** file to achieve the same results at mount time. The following **/etc/fstab** entry will result in the contents of **/root/tmp** being identical to the contents of the **/var/log** directory.

```
/var/log                /root/tmp               none    bind
0 0
```

After you have mounted the file system, you can use the **mount** command to see that the file system has been mounted, as in the following example.

```
[root@menscryfa ~]# mount | grep /tmp
/var/log on /root/tmp type none (rw,bind)
```

With a file system that supports Context-Dependent Path Names, you might have defined the **/bin** directory as a Context-Dependent Path Name that would resolve to one of the following paths, depending on the system architecture.

```
/usr/i386-bin
/usr/x86_64-bin
/usr/ppc64-bin
```

You can achieve this same functionality by creating an empty **/bin** directory. Then, using a script or an entry in the **/etc/fstab** file, you can mount each of the individual architecture directories onto the **/bin** directory with a **mount -bind** command. For example, you can use the following command as a line in a script.

```
mount --bind /usr/i386-bin /bin
```

Alternately, you can use the following entry in the **/etc/fstab** file.

```
/usr/1386-bin             /bin                none    bind            0 0
```

A bind mount can provide greater flexibility than a Context-Dependent Path Name, since you can use this feature to mount different directories according to any criteria you define (such as the value of **%fill** for the file system). Context-Dependent Path Names are more limited in what they can encompass. Note, however, that you will need to write your own script to mount according to a criteria such as the value of **%fill**.

Warning

When you mount a file system with the **bind** option and the original file system was mounted **rw**, the new file system will also be mounted **rw** even if you use the **ro** flag; the **ro** flag is silently ignored. In this case, the new file system might be marked as **ro** in the **/proc/mounts** directory, which may be misleading.

4.13. Bind Mounts and File System Mount Order

When you use the **bind** option of the **mount** command, you must be sure that the file systems are mounted in the correct order. In the following example, the **/var/log** directory must be mounted before executing the bind mount on the **/tmp** directory:

```
# mount --bind /var/log /tmp
```

The ordering of file system mounts is determined as follows:

- In general, file system mount order is determined by the order in which the file systems appear in the **fstab** file. The exceptions to this ordering are file systems mounted with the **_netdev** flag or file systems that have their own **init** scripts.
- A file system with its own **init** script is mounted later in the initialization process, after the file systems in the **fstab** file.
- File systems mounted with the **_netdev** flag are mounted when the network has been enabled on the system.

If your configuration requires that you create a bind mount on which to mount a GFS2 file system, you can order your **fstab** file as follows:

1. Mount local file systems that are required for the bind mount.
2. Bind mount the directory on which to mount the GFS2 file system.
3. Mount the GFS2 file system.

If your configuration requires that you bind mount a local directory or file system onto a GFS2 file system, listing the file systems in the correct order in the **fstab** file will not mount the file systems correctly since the GFS2 file system will not be mounted until the GFS2 **init** script is run. In this case, you should write an **init** script to execute the bind mount so that the bind mount will not take place until after the GFS2 file system is mounted.

The following script is an example of a custom **init** script. This script performs a bind mount of two directories onto two directories of a GFS2 file system. In this example, there is an existing GFS2 mount point at **/mnt/gfs2a**, which is mounted when the GFS2 **init** script runs, after cluster startup.

In this example script, the values of the **chkconfig** statement indicate the following:

- 345 indicates the run levels that the script will be started in
- 29 is the start priority, which in this case indicates that the script will run at startup time after the GFS2 **init** script, which has a start priority of 26
- 73 is the stop priority, which in this case indicates that the script will be stopped during shutdown before the GFS2 script, which has a stop priority of 74

The start and stop values indicate that you can manually perform the indicated action by executing a **service start** and a **service stop** command. For example, if the script is named **fredwilma**, then you can execute **service fredwilma start**.

This script should be put in the **/etc/init.d** directory with the same permissions as the other scripts in that directory. You can then execute a **chkconfig on** command to link the script to the indicated run levels. For example, if the script is named **fredwilma**, then you can execute **chkconfig fredwilma on**.

```
#!/bin/bash
#
# chkconfig: 345 29 73
# description: mount/unmount my custom bind mounts onto a gfs2
subdirectory
#
#
### BEGIN INIT INFO
# Provides:
```

```
### END INIT INFO

. /etc/init.d/functions
case "$1" in
  start)
 # In this example, fred and wilma want their home directories
 # bind-mounted over the gfs2 directory /mnt/gfs2a, which has
 # been mounted as /mnt/gfs2a
 mkdir -p /mnt/gfs2a/home/fred &> /dev/null
 mkdir -p /mnt/gfs2a/home/wilma &> /dev/null
 /bin/mount --bind /mnt/gfs2a/home/fred /home/fred
 /bin/mount --bind /mnt/gfs2a/home/wilma /home/wilma
        ;;

  stop)
 /bin/umount /mnt/gfs2a/home/fred
 /bin/umount /mnt/gfs2a/home/wilma
        ;;

  status)
        ;;

  restart)
        $0 stop
        $0 start
        ;;

  reload)
        $0 start
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|reload|status}"
        exit 1
esac

exit 0
```

4.14. The GFS2 Withdraw Function

The GFS2 *withdraw* function is a data integrity feature of GFS2 file systems in a cluster. If the GFS2 kernel module detects an inconsistency in a GFS2 file system following an I/O operation, the file system becomes unavailable to the cluster. The I/O operation stops and the system waits for further I/O operations to stop with an error, preventing further damage. When this occurs, you can stop any other services or applications manually, after which you can reboot and remount the GFS2 file system to replay the journals. If the problem persists, you can unmount the file system from all nodes in the cluster and perform file system recovery with the **fsck.gfs2** command. The GFS withdraw function is less severe than a kernel panic, which would cause another node to fence the node.

If your system is configured with the **gfs2** startup script enabled and the GFS2 file system is included in the **/etc/fstab** file, the GFS2 file system will be remounted when you reboot. If the GFS2 file system withdrew because of perceived file system corruption, it is recommended that you run the **fsck.gfs2** command before remounting the file system. In this case, in order to prevent your file system from remounting at boot time, you can perform the following procedure:

1. Temporarily disable the startup script on the affected node with the following command:

```
# chkconfig gfs2 off
```

2. Reboot the affected node, starting the cluster software. The GFS2 file system will not be mounted.
3. Unmount the file system from every node in the cluster.
4. Run the **fsck.gfs2** on the file system from one node only to ensure there is no file system corruption.
5. Re-enable the startup script on the affected node by running the following command:

```
# chkconfig gfs2 on
```

6. Remount the GFS2 file system from all nodes in the cluster.

An example of an inconsistency that would yield a GFS2 withdraw is an incorrect block count. When the GFS kernel deletes a file from a file system, it systematically removes all the data and metadata blocks associated with that file. When it is done, it checks the block count. If the block count is not one (meaning all that is left is the disk inode itself), that indicates a file system inconsistency since the block count did not match the list of blocks found.

You can override the GFS2 withdraw function by mounting the file system with the **-o errors=panic** option specified. When this option is specified, any errors that would normally cause the system to withdraw cause the system to panic instead. This stops the node's cluster communications, which causes the node to be fenced.

Internally, the GFS2 withdraw function works by having the kernel send a message to the **gfs_controld** daemon requesting withdraw. The **gfs_controld** daemon runs the **dmsetup** program to place the device mapper error target underneath the file system preventing further access to the block device. It then tells the kernel that this has been completed. This is the reason for the GFS2 support requirement to always use a CLVM device under GFS2, since otherwise it is not possible to insert a device mapper target.

The purpose of the device mapper error target is to ensure that all future I/O operations will result in an I/O error that will allow the file system to be unmounted in an orderly fashion. As a result, when the withdraw occurs, it is normal to see a number of I/O errors from the device mapper device reported in the system logs.

Occasionally, the withdraw may fail if it is not possible for the **dmsetup** program to insert the error target as requested. This can happen if there is a shortage of memory at the point of the withdraw and memory cannot be reclaimed due to the problem that triggered the withdraw in the first place.

A withdraw does not always mean that there is an error in GFS2. Sometimes the withdraw function can be triggered by device I/O errors relating to the underlying block device. It is highly recommended to check the logs to see if that is the case if a withdraw occurs.

Chapter 5. Diagnosing and Correcting Problems with GFS2 File Systems

This chapter provides information about some common GFS2 issues and how to address them.

5.1. GFS2 File System Shows Slow Performance

You may find that your GFS2 file system shows slower performance than an ext3 file system. GFS2 performance may be affected by a number of influences and in certain use cases. Information that addresses GFS2 performance issues is found throughout this document.

5.2. GFS2 File System Hangs and Requires Reboot of One Node

If your GFS2 file system hangs and does not return commands run against it, but rebooting one specific node returns the system to normal, this may be indicative of a locking problem or bug. Should this occur, gather the following data:

- The gfs2 lock dump for the file system on each node:

  ```
  cat /sys/kernel/debug/gfs2/fsname/glocks >glocks.fsname.nodename
  ```

- The DLM lock dump for the file system on each node: You can get this information with the **dlm_tool**:

  ```
  dlm_tool lockdebug -sv lsname.
  ```

  In this command, *lsname* is the lockspace name used by DLM for the file system in question. You can find this value in the output from the **group_tool** command.

- The output from the **sysrq -t** command.

- The contents of the **/var/log/messages** file.

Once you have gathered that data, you can open a ticket with Red Hat Support and provide the data you have collected.

5.3. GFS2 File System Hangs and Requires Reboot of All Nodes

If your GFS2 file system hangs and does not return commands run against it, requiring that you reboot all nodes in the cluster before using it, check for the following issues.

- You may have had a failed fence. GFS2 file systems will freeze to ensure data integrity in the event of a failed fence. Check the messages logs to see if there are any failed fences at the time of the hang. Ensure that fencing is configured correctly.

- The GFS2 file system may have withdrawn. Check through the messages logs for the word **withdraw** and check for any messages and calltraces from GFS2 indicating that the file system has been withdrawn. A withdraw is indicative of file system corruption, a storage failure, or a bug. Unmount the file system, update the **gfs2-utils** package, and execute the **fsck** command on the file system to return it to service. Open a support ticket with Red Hat Support. Inform them you experienced a GFS2 withdraw and provide sosreports with logs.

  For information on the GFS2 withdraw function, see Section 4.14, “The GFS2 Withdraw Function”.

- This error may be indicative of a locking problem or bug. Gather data during one of these occurrences and open a support ticket with Red Hat Support, as described in Section 5.2, “GFS2 File System Hangs and Requires Reboot of One Node”.

5.4. GFS2 File System Does Not Mount on Newly-Added Cluster Node

If you add a new node to a cluster and you find that you cannot mount your GFS2 file system on that node, you may have fewer journals on the GFS2 file system than you have nodes attempting to access the GFS2 file system. You must have one journal per GFS2 host you intend to mount the file system on (with the exception of GFS2 file systems mounted with the **spectator** mount option set, since these do not require a journal). You can add journals to a GFS2 file system with the **gfs2_jadd** command, as described in Section 4.7, “Adding Journals to a File System”.

5.5. Space Indicated as Used in Empty File System

If you have an empty GFS2 file system, the **df** command will show that there is space being taken up. This is because GFS2 file system journals consume space (number of journals * journal size) on disk. If you created a GFS2 file system with a large number of journals or specified a large journal size then you will be see (number of journals * journal size) as already in use when you execute the **df**. Even if you did not specify a large number of journals or large journals, small GFS2 file systems (in the 1GB or less range) will show a large amount of space as being in use with the default GFS2 journal size.

Chapter 6. Configuring a GFS2 File System in a Pacemaker Cluster

The following procedure is an outline of the steps required to set up a Pacemaker cluster that includes a GFS2 file system.

1. On each node in the cluster, install the High Availability and Resilient Storage packages.

```
# yum groupinstall 'High Availability' 'Resilient Storage'
```

2. Create the Pacemaker cluster and configure fencing for the cluster. For information on configuring a Pacemaker cluster, see *Configuring the Red Hat High Availability Add-On with Pacemaker*.

3. On each node in the cluster, enable the **clvmd** service. If you will be using cluster-mirrored volumes, enable the **cmirrord** service.

```
# chkconfig clvmd on
# chkconfig cmirrord on
```

   After you enable these daemons, when starting and stopping Pacemaker or the cluster through normal means using **pcs cluster start**, **pcs cluster stop**, **service pacemaker start**, or **service pacemaker stop**, the **clvmd** and **cmirrord** daemons will be started and stopped as needed.

4. On one node in the cluster, perform the following steps:

   a. Set the global Pacemaker parameter **no_quorum_policy** to **freeze**.

Note

By default, the value of **no-quorum-policy** is set to **stop**, indicating that once quorum is lost, all the resources on the remaining partition will immediately be stopped. Typically this default is the safest and most optimal option, but unlike most resources, GFS2 requires quorum to function. When quorum is lost both the applications using the GFS2 mounts and the GFS2 mount itself cannot be correctly stopped. Any attempts to stop these resources without quorum will fail which will ultimately result in the entire cluster being fenced every time quorum is lost.

To address this situation, you can set the **no-quorum-policy=freeze** when GFS2 is in use. This means that when quorum is lost, the remaining partition will do nothing until quorum is regained.

```
# pcs property set no-quorum-policy=freeze
```

   b. After ensuring that the locking type is set to 3 in the **/etc/lvm/lvm.conf** file to support clustered locking, Create the clustered LV and format the volume with a GFS2 file system. Ensure that you create enough journals for each of the nodes in your cluster.

```
# pvcreate /dev/vdb
# vgcreate -Ay -cy cluster_vg /dev/vdb
# lvcreate -L5G -n cluster_lv cluster_vg
# mkfs.gfs2 -j2 -p lock_dlm -t rhel7-demo:gfs2-demo
/dev/cluster_vg/cluster_lv
```

c. Configure a **clusterfs** resource.

   You should not add the file system to the **/etc/fstab** file because it will be managed as a Pacemaker cluster resource. Mount options can be specified as part of the resource configuration with **options=*options***. Run the **pcs resource describe Filesystem** command for full configuration options.

   This cluster resource creation command specifies the **noatime** mount option.

```
# pcs resource create clusterfs Filesystem
device="/dev/cluster_vg/cluster_lv"
directory="/var/mountpoint" fstype="gfs2" "options=noatime"
op monitor interval=10s on-fail=fence clone interleave=true
```

d. Verify that GFS2 is mounted as expected.

```
# mount |grep /mnt/gfs2-demo
/dev/mapper/cluster_vg-cluster_lv on /mnt/gfs2-demo type gfs2
(rw,noatime,seclabel)
```

5. (Optional) Reboot all cluster nodes to verify GFS2 persistence and recovery.

Appendix A. GFS2 Quota Management with the `gfs2_quota` Command

As of the Red Hat Enterprise Linux 6.1 release, GFS2 supports the standard Linux quota facilities. In order to use this you will need to install the **quota** RPM. This is the preferred way to administer quotas on GFS2 and should be used for all new deployments of GFS2 using quotas. For information on using the standard Linux quota facilities, see Section 4.5, “GFS2 Quota Management”.

For earlier releases of Red Hat Enterprise Linux, GFS2 required the **gfs2_quota** command to manage quotas. This appendix documents the use of the **gfs2_quota** command for managing GFS2 file system quotas.

A.1. Setting Quotas with the `gfs2_quota` command

Two quota settings are available for each user ID (UID) or group ID (GID): a *hard limit* and a *soft limit*.

A hard limit is the amount of space that can be used. The file system will not let the user or group use more than that amount of disk space. A hard limit value of *zero* means that no limit is enforced.

A soft limit is usually a value less than the hard limit. The file system will notify the user or group when the soft limit is reached to warn them of the amount of space they are using. A soft limit value of *zero* means that no limit is enforced.

You can set limits using the **gfs2_quota** command. The command only needs to be run on a single node where GFS2 is mounted.

By default, quota enforcement is not set on GFS2 file systems. To enable quota accounting, use the **quota=** of the **mount** command when mounting the GFS2 file system, as described in Section A.4, “Enabling/Disabling Quota Enforcement”.

Usage

Setting Quotas, Hard Limit

```
gfs2_quota limit -u User -l Size -f MountPoint
```

```
gfs2_quota limit -g Group -l Size -f MountPoint
```

Setting Quotas, Warn Limit

```
gfs2_quota warn -u User -l Size -f MountPoint
```

```
gfs2_quota warn -g Group -l Size -f MountPoint
```

User

A user ID to limit or warn. It can be either a user name from the password file or the UID number.

Group

A group ID to limit or warn. It can be either a group name from the group file or the GID number.

Size

Specifies the new value to limit or warn. By default, the value is in units of megabytes. The additional **-k**, **-s** and **-b** flags change the units to kilobytes, sectors, and file system blocks, respectively.

MountPoint

Specifies the GFS2 file system to which the actions apply.

Examples

This example sets the hard limit for user *Bert* to 1024 megabytes (1 gigabyte) on file system **/mygfs2**.

```
# gfs2_quota limit -u Bert -l 1024 -f /mygfs2
```

This example sets the soft limit for group ID 21 to 50 kilobytes on file system **/mygfs2**.

```
# gfs2_quota warn -g 21 -l 50 -k -f /mygfs2
```

A.2. Displaying Quota Limits and Usage with the `gfs2_quota` Command

Quota limits and current usage can be displayed for a specific user or group using the **gfs2_quota get** command. The entire contents of the quota file can also be displayed using the **gfs2_quota list** command, in which case all IDs with a non-zero hard limit, soft limit, or value are listed.

Usage

Displaying Quota Limits for a User

```
gfs2_quota get -u User -f MountPoint
```

Displaying Quota Limits for a Group

```
gfs2_quota get -g Group -f MountPoint
```

Displaying Entire Quota File

```
gfs2_quota list -f MountPoint
```

User

A user ID to display information about a specific user. It can be either a user name from the password file or the UID number.

Group

A group ID to display information about a specific group. It can be either a group name from the group file or the GID number.

MountPoint

Specifies the GFS2 file system to which the actions apply.

Command Output

GFS2 quota information from the **gfs2_quota** command is displayed as follows:

```
user User: limit:LimitSize warn:WarnSize value:Value

group Group: limit:LimitSize warn:WarnSize value:Value
```

The ***LimitSize***, ***WarnSize***, and ***Value*** numbers (values) are in units of megabytes by default. Adding the **-k**, **-s**, or **-b** flags to the command line change the units to kilobytes, sectors, or file system blocks, respectively.

User

A user name or ID to which the data is associated.

Group

A group name or ID to which the data is associated.

LimitSize

The hard limit set for the user or group. This value is zero if no limit has been set.

Value

The actual amount of disk space used by the user or group.

Comments

When displaying quota information, the **gfs2_quota** command does not resolve UIDs and GIDs into names if the **-n** option is added to the command line.

Space allocated to GFS2's hidden files can be left out of displayed values for the root UID and GID by adding the **-d** option to the command line. This is useful when trying to match the numbers from **gfs2_quota** with the results of a **du** command.

Examples

This example displays quota information for all users and groups that have a limit set or are using any disk space on file system **/mygfs2**.

```
# gfs2_quota list -f /mygfs2
```

This example displays quota information in sectors for group **users** on file system **/mygfs2**.

```
# gfs2_quota get -g users -f /mygfs2 -s
```

A.3. Synchronizing Quotas with the gfs2_quota Command

GFS2 stores all quota information in its own internal file on disk. A GFS2 node does not update this quota file for every file system write; rather, by default it updates the quota file once every 60 seconds. This is necessary to avoid contention among nodes writing to the quota file, which would cause a slowdown in performance.

As a user or group approaches their quota limit, GFS2 dynamically reduces the time between its quota-file updates to prevent the limit from being exceeded. The normal time period between quota synchronizations is a tunable parameter, **quota_quantum**. You can change this from its default value of 60 seconds using the **quota_quantum=** mount option, as described in Table 4.2, "GFS2-Specific Mount Options". The **quota_quantum** parameter must be set on each node and each time the file system is mounted. Changes to the **quota_quantum** parameter are not persistent across unmounts. You can update the **quota_quantum** value with the **mount -o remount**.

You can use the **gfs2_quota sync** command to synchronize the quota information from a node to the on-disk quota file between the automatic updates performed by GFS2.

Usage

Synchronizing Quota Information

```
gfs2_quota sync -f MountPoint
```

MountPoint

Specifies the GFS2 file system to which the actions apply.

Tuning the Time Between Synchronizations

```
mount -o quota_quantum=secs,remount BlockDevice MountPoint
```

MountPoint

Specifies the GFS2 file system to which the actions apply.

secs

Specifies the new time period between regular quota-file synchronizations by GFS2. Smaller values may increase contention and slow down performance.

Examples

This example synchronizes the quota information from the node it is run on to file system **/mygfs2**.

```
# gfs2_quota sync -f /mygfs2
```

This example changes the default time period between regular quota-file updates to one hour (3600 seconds) for file system **/mnt/mygfs2** when remounting that file system on logical volume **/dev/volgroup/logical_volume**.

```
# mount -o quota_quantum=3600,remount /dev/volgroup/logical_volume
/mnt/mygfs2
```

A.4. Enabling/Disabling Quota Enforcement

In GFS2 file systems, quota enforcement is disabled by default. To enable quota enforcement for a file system, mount the file system with the **quota=on** option specified.

Usage

```
mount -o quota=on BlockDevice MountPoint
```

To mount a file system with quota enforcement disabled, mount the file system with the **quota=off** option specified. This is the default setting.

```
mount -o quota=off BlockDevice MountPoint
```

-o quota={on|off}

Specifies that quota enforcement is enabled or disabled when the file system is mounted.

BlockDevice

Specifies the block device where the GFS2 file system resides.

MountPoint

Specifies the directory where the GFS2 file system should be mounted.

Examples

In this example, the GFS2 file system on **/dev/vg01/lvol0** is mounted on the **/mygfs2** directory with quota enforcement enabled.

```
# mount -o quota=on /dev/vg01/lvol0 /mygfs2
```

A.5. Enabling Quota Accounting

It is possible to keep track of disk usage and maintain quota accounting for every user and group without enforcing the limit and warn values. To do this, mount the file system with the **quota=account** option specified.

Usage

```
mount -o quota=account BlockDevice MountPoint
```

-o quota=account

Specifies that user and group usage statistics are maintained by the file system, even though the quota limits are not enforced.

BlockDevice

Specifies the block device where the GFS2 file system resides.

MountPoint

Specifies the directory where the GFS2 file system should be mounted.

Example

In this example, the GFS2 file system on **/dev/vg01/lvol0** is mounted on the **/mygfs2** directory with quota accounting enabled.

```
# mount -o quota=account /dev/vg01/lvol0 /mygfs2
```

Appendix B. Converting a File System from GFS to GFS2

Since the Red Hat Enterprise Linux 6 release does not support GFS file systems, you must upgrade any existing GFS file systems to GFS2 file systems with the **gfs2_convert** command. Note that you must perform this conversion procedure on a Red Hat Enterprise Linux 5 system before upgrading to Red Hat Enterprise Linux 6.

Warning

Before converting the GFS file system, you must back up the file system, since the conversion process is irreversible and any errors encountered during the conversion can result in the abrupt termination of the program and consequently an unusable file system.

Before converting the GFS file system, you must use the **gfs_fsck** command to check the file system and fix any errors.

If the conversion from GFS to GFS2 is interrupted by a power failure or any other issue, restart the conversion tool. Do not attempt to execute the **fsck.gfs2** command on the file system until the conversion is complete.

When converting full or nearly full file systems, it is possible that there will not be enough space available to fit all the GFS2 file system data structures. In such cases, the size of all the journals is reduced uniformly such that everything fits in the available space.

B.1. Conversion of Context-Dependent Path Names

GFS2 file systems do not provide support for Context-Dependent Path Names (CDPNs), which allow you to create symbolic links that point to variable destination files or directories. To achieve the same functionality as CDPNs in GFS2 file systems, you can use the **bind** option of the **mount** command.

The **gfs2_convert** command identifies CDPNs and replaces them with empty directories with the same name. In order to configure bind mounts to replace the CDPNs, however, you need to know the full paths of the link targets of the CDPNs you are replacing. Before converting your file system, you can use the **find** command to identify the links.

The following command lists the symlinks that point to a **hostname** CDPN:

```
[root@smoke-01 gfs]# find /mnt/gfs -lname @hostname
/mnt/gfs/log
```

Similarly, you can execute the **find** command for other CDPNs (**mach**, **os**, **sys**, **uid**, **gid**, **jid**). Note that since CDPN names can be of the form **@hostname** or **{hostname}**, you will need to run the **find** command for each variant.

For more information on bind mounts and context-dependent pathnames in GFS2, see Section 4.12, “Bind Mounts and Context-Dependent Path Names”.

B.2. GFS to GFS2 Conversion Procedure

Use the following procedure to convert a GFS file system to a GFS2 file system.

1. On a Red Hat Enterprise Linux system, make a backup of your existing GFS file system.
2. Unmount the GFS file system from all nodes in the cluster.
3. Execute the **gfs_fsck** command on the GFS file system to ensure there is no file system corruption.
4. Execute **gfs2_convert *gfsfilesystem***. The system will display warnings and confirmation questions before converting ***gfsfilesystem*** to GFS2.
5. Upgrade to Red Hat Enterprise Linux 6.

The following example converts a GFS file system on block device **/dev/shell_vg/500g** to a GFS2 file system.

```
[root@shell-01 ~]#  /root/cluster/gfs2/convert/gfs2_convert
/dev/shell_vg/500g
gfs2_convert version 2 (built May 10 2010 10:05:40)
Copyright (C) Red Hat, Inc.  2004-2006  All rights reserved.

Examining file system...................
This program will convert a gfs1 filesystem to a gfs2 filesystem.
WARNING: This can't be undone.  It is strongly advised that you:

   1. Back up your entire filesystem first.
   2. Run gfs_fsck first to ensure filesystem integrity.
   3. Make sure the filesystem is NOT mounted from any node.
   4. Make sure you have the latest software versions.
Convert /dev/shell_vg/500g from GFS1 to GFS2? (y/n)y
Converting resource groups...................
Converting inodes.
24208 inodes from 1862 rgs converted.
Fixing file and directory information.
18 cdpn symlinks moved to empty directories.
Converting journals.
Converting journal space to rg space.
Writing journal #1...done.
Writing journal #2...done.
Writing journal #3...done.
Writing journal #4...done.
Building GFS2 file system structures.
Removing obsolete GFS1 file system structures.
Committing changes to disk.
/dev/shell_vg/500g: filesystem converted successfully to gfs2.
```

Appendix C. GFS2 tracepoints and the debugfs glocks File

This appendix describes both the glock **debugfs** interface and the GFS2 tracepoints. It is intended for advanced users who are familiar with file system internals who would like to learn more about the design of GFS2 and how to debug GFS2-specific issues.

C.1. GFS2 tracepoint Types

There are currently three types of GFS2 tracepoints: *glock* (pronounced "gee-lock") tracepoints, *bmap* tracepoints and *log* tracepoints. These can be used to monitor a running GFS2 file system and give additional information to that which can be obtained with the debugging options supported in previous releases of Red Hat Enterprise Linux. Tracepoints are particularly useful when a problem, such as a hang or performance issue, is reproducible and thus the tracepoint output can be obtained during the problematic operation. In GFS2, glocks are the primary cache control mechanism and they are the key to understanding the performance of the core of GFS2. The bmap (block map) tracepoints can be used to monitor block allocations and block mapping (lookup of already allocated blocks in the on-disk metadata tree) as they happen and check for any issues relating to locality of access. The log tracepoints keep track of the data being written to and released from the journal and can provide useful information on that part of GFS2.

The tracepoints are designed to be as generic as possible. This should mean that it will not be necessary to change the API during the course of Red Hat Enterprise Linux 6. On the other hand, users of this interface should be aware that this is a debugging interface and not part of the normal Red Hat Enterprise Linux 6 API set, and as such Red Hat makes no guarantees that changes in the GFS2 tracepoints interface will not occur.

Tracepoints are a generic feature of Red Hat Enterprise Linux 6 and their scope goes well beyond GFS2. In particular they are used to implement the **blktrace** infrastructure and the **blktrace** tracepoints can be used in combination with those of GFS2 to gain a fuller picture of the system performance. Due to the level at which the tracepoints operate, they can produce large volumes of data in a very short period of time. They are designed to put a minimum load on the system when they are enabled, but it is inevitable that they will have some effect. Filtering events with a variety of means can help reduce the volume of data and help focus on obtaining just the information which is useful for understanding any particular situation.

C.2. Tracepoints

The tracepoints can be found under **/sys/kernel/debug/tracing/** directory assuming that **debugfs** is mounted in the standard place at the **/sys/kernel/debug** directory. The **events** subdirectory contains all the tracing events that may be specified and, provided the **gfs2** module is loaded, there will be a **gfs2** subdirectory containing further subdirectories, one for each GFS2 event. The contents of the **/sys/kernel/debug/tracing/events/gfs2** directory should look roughly like the following:

```
[root@chywoon gfs2]# ls
enable            gfs2_bmap       gfs2_glock_queue          gfs2_log_flush
filter            gfs2_demote_rq  gfs2_glock_state_change   gfs2_pin
gfs2_block_alloc  gfs2_glock_put  gfs2_log_blocks           gfs2_promote
```

To enable all the GFS2 tracepoints, enter the following command:

```
[root@chywoon gfs2]# echo -n 1
>/sys/kernel/debug/tracing/events/gfs2/enable
```

To enable a specific tracepoint, there is an **enable** file in each of the individual event subdirectories. The same is true of the **filter** file which can be used to set an event filter for each event or set of events. The meaning of the individual events is explained in more detail below.

The output from the tracepoints is available in ASCII or binary format. This appendix does not currently cover the binary interface. The ASCII interface is available in two ways. To list the current content of the ring buffer, you can enter the following command:

```
[root@chywoon gfs2]# cat /sys/kernel/debug/tracing/trace
```

This interface is useful in cases where you are using a long-running process for a certain period of time and, after some event, want to look back at the latest captured information in the buffer. An alternative interface, **/sys/kernel/debug/tracing/trace_pipe**, can be used when all the output is required. Events are read from this file as they occur; there is no historical information available through this interface. The format of the output is the same from both interfaces and is described for each of the GFS2 events in the later sections of this appendix.

A utility called **trace-cmd** is available for reading tracepoint data. For more information on this utility, see the link in Section C.10, “References”. The **trace-cmd** utility can be used in a similar way to the **strace** utility, for example to run a command while gathering trace data from various sources.

C.3. Glocks

To understand GFS2, the most important concept to understand, and the one which sets it aside from other file systems, is the concept of glocks. In terms of the source code, a glock is a data structure that brings together the DLM and caching into a single state machine. Each glock has a 1:1 relationship with a single DLM lock, and provides caching for that lock state so that repetitive operations carried out from a single node of the file system do not have to repeatedly call the DLM, and thus they help avoid unnecessary network traffic. There are two broad categories of glocks, those which cache metadata and those which do not. The inode glocks and the resource group glocks both cache metadata, other types of glocks do not cache metadata. The inode glock is also involved in the caching of data in addition to metadata and has the most complex logic of all glocks.

Table C.1. Glock Modes and DLM Lock Modes

Glock mode	DLM lock mode	Notes
UN	IV/NL	Unlocked (no DLM lock associated with glock or NL lock depending on I flag)
SH	PR	Shared (protected read) lock
EX	EX	Exclusive lock
DF	CW	Deferred (concurrent write) used for Direct I/O and file system freeze

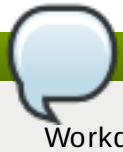
Glocks remain in memory until either they are unlocked (at the request of another node or at the request of the VM) and there are no local users. At that point they are removed from the glock hash table and freed. When a glock is created, the DLM lock is not associated with the glock immediately. The DLM lock becomes associated with the glock upon the first request to the DLM, and if this request is successful then the 'I' (initial) flag will be set on the glock. Table C.4, “Glock flags” shows the meanings of the different glock flags. Once the DLM has been associated with the glock, the DLM lock will always remain at least at NL (Null) lock mode until the glock is to be freed. A demotion of the DLM lock from NL to unlocked is always the last operation in the life of a glock.

Note

This particular aspect of DLM lock behavior has changed since Red Hat Enterprise Linux 5, which does sometimes unlock the DLM locks attached to glocks completely, and thus Red Hat Enterprise Linux 5 has a different mechanism to ensure that LVBs (lock value blocks) are preserved where required. The new scheme that Red Hat Enterprise Linux 6 uses was made possible due to the merging of the **lock_dlm** lock module (not to be confused with the DLM itself) into GFS2.

Each glock can have a number of "holders" associated with it, each of which represents one lock request from the higher layers. System calls relating to GFS2 queue and dequeue holders from the glock to protect the critical section of code.

The glock state machine is based on a workqueue. For performance reasons, tasklets would be preferable; however, in the current implementation we need to submit I/O from that context which prohibits their use.



Note

Workqueues have their own tracepoints which can be used in combination with the GFS2 tracepoints if desired

Table C.2, “Glock Modes and Data Types” shows what state may be cached under each of the glock modes and whether that cached state may be dirty. This applies to both inode and resource group locks, although there is no data component for the resource group locks, only metadata.

Table C.2. Glock Modes and Data Types

Glock mode	Cache Data	Cache Metadata	Dirty Data	Dirty Metadata
UN	No	No	No	No
SH	Yes	Yes	No	No
DF	No	Yes	No	No
EX	Yes	Yes	Yes	Yes

C.4. The glock debugfs Interface

The glock **debugfs** interface allows the visualization of the internal state of the glocks and the holders and it also includes some summary details of the objects being locked in some cases. Each line of the file either begins G: with no indentation (which refers to the glock itself) or it begins with a different letter, indented with a single space, and refers to the structures associated with the glock immediately above it in the file (H: is a holder, I: an inode, and R: a resource group) . Here is an example of what the content of this file might look like:

```
G:  s:SH n:5/75320 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:EX n:3/258028 f:yI t:EX d:EX/0 a:3 r:4
 H: s:EX f:tH e:0 p:4466 [postmark] gfs2_inplace_reserve_i+0x177/0x780
[gfs2]
 R: n:258028 f:05 b:22256/22256 i:16800
```

```
G:  s:EX n:2/219916 f:yfI t:EX d:EX/0 a:0 r:3
 I: n:75661/219916 t:8 f:0x10 d:0x00000000 s:7522/7522
G:  s:SH n:5/127205 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:EX n:2/50382 f:yfI t:EX d:EX/0 a:0 r:2
G:  s:SH n:5/302519 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/313874 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/271916 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/312732 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
```

The above example is a series of excerpts (from an approximately 18MB file) generated by the command **cat /sys/kernel/debug/gfs2/unity:myfs/glocks >my.lock** during a run of the postmark benchmark on a single node GFS2 file system. The glocks in the figure have been selected in order to show some of the more interesting features of the glock dumps.

The glock states are either EX (exclusive), DF (deferred), SH (shared) or UN (unlocked). These states correspond directly with DLM lock modes except for UN which may represent either the DLM null lock state, or that GFS2 does not hold a DLM lock (depending on the I flag as explained above). The s: field of the glock indicates the current state of the lock and the same field in the holder indicates the requested mode. If the lock is granted, the holder will have the H bit set in its flags (f: field). Otherwise, it will have the W wait bit set.

The n: field (number) indicates the number associated with each item. For glocks, that is the type number followed by the glock number so that in the above example, the first glock is n:5/75320; that is, an **iopen** glock which relates to inode 75320. In the case of inode and **iopen** glocks, the glock number is always identical to the inode's disk block number.

Note

The glock numbers (n: field) in the debugfs glocks file are in hexadecimal, whereas the tracepoints output lists them in decimal. This is for historical reasons; glock numbers were always written in hex, but decimal was chosen for the tracepoints so that the numbers could easily be compared with the other tracepoint output (from **blktrace** for example) and with output from **stat**(1).

The full listing of all the flags for both the holder and the glock are set out in Table C.4, “Glock flags” and Table C.5, “Glock holder flags”. The content of lock value blocks is not currently available by means of the glock **debugfs** interface.

Table C.3, “Glock types” shows the meanings of the different glock types.

Table C.3. Glock types

Type number	Lock type	Use
1	trans	Transaction lock
2	inode	Inode metadata and data
3	rgrp	Resource group metadata
4	meta	The superblock

Type number	Lock type	Use
5	iopen	Inode last closer detection
6	flock	**flock**(2) syscall
8	quota	Quota operations
9	journal	Journal mutex

One of the more important glock flags is the l (locked) flag. This is the bit lock that is used to arbitrate access to the glock state when a state change is to be performed. It is set when the state machine is about to send a remote lock request by mean of the DLM, and only cleared when the complete operation has been performed. Sometimes this can mean that more than one lock request will have been sent, with various invalidations occurring between times.

Table C.4, “Glock flags” shows the meanings of the different glock flags.

Table C.4. Glock flags

Flag	Name	Meaning
d	Pending demote	A deferred (remote) demote request
D	Demote	A demote request (local or remote)
f	Log flush	The log needs to be committed before releasing this glock
F	Frozen	Replies from remote nodes ignored - recovery is in progress.
i	Invalidate in progress	In the process of invalidating pages under this glock
I	Initial	Set when DLM lock is associated with this glock
l	Locked	The glock is in the process of changing state
L	LRU	Set when the glock is on the LRU list`
o	Object	Set when the glock is associated with an object (that is, an inode for type 2 glocks, and a resource group for type 3 glocks)
p	Demote in progress	The glock is in the process of responding to a demote request
q	Queued	Set when a holder is queued to a glock, and cleared when the glock is held, but there are no remaining holders. Used as part of the algorithm the calculates the minimum hold time for a glock.
r	Reply pending	Reply received from remote node is awaiting processing
y	Dirty	Data needs flushing to disk before releasing this glock

When a remote callback is received from a node that wants to get a lock in a mode that conflicts with that being held on the local node, then one or other of the two flags D (demote) or d (demote pending) is set. In order to prevent starvation conditions when there is contention on a particular lock, each lock is assigned a minimum hold time. A node which has not yet had the lock for the minimum hold time is allowed to retain that lock until the time interval has expired.

If the time interval has expired, then the D (demote) flag will be set and the state required will be recorded. In that case the next time there are no granted locks on the holders queue, the lock will be demoted. If the time interval has not expired, then the d (demote pending) flag is set instead. This also schedules the state machine to clear d (demote pending) and set D (demote) when the minimum hold time has expired.

The I (initial) flag is set when the glock has been assigned a DLM lock. This happens when the glock is first used and the I flag will then remain set until the glock is finally freed (which the DLM lock is unlocked).

C.5. Glock Holders

Table C.5, “Glock holder flags” shows the meanings of the different glock holder flags.

Table C.5. Glock holder flags

Flag	Name	Meaning
a	Async	Do not wait for glock result (will poll for result later)
A	Any	Any compatible lock mode is acceptable
c	No cache	When unlocked, demote DLM lock immediately
e	No expire	Ignore subsequent lock cancel requests
E	Exact	Must have exact lock mode
F	First	Set when holder is the first to be granted for this lock
H	Holder	Indicates that requested lock is granted
p	Priority	Enqueue holder at the head of the queue
t	Try	A "try" lock
T	Try 1CB	A "try" lock that sends a callback
W	Wait	Set while waiting for request to complete

The most important holder flags are H (holder) and W (wait) as mentioned earlier, since they are set on granted lock requests and queued lock requests respectively. The ordering of the holders in the list is important. If there are any granted holders, they will always be at the head of the queue, followed by any queued holders.

If there are no granted holders, then the first holder in the list will be the one that triggers the next state change. Since demote requests are always considered higher priority than requests from the file system, that might not always directly result in a change to the state requested.

The glock subsystem supports two kinds of "try" lock. These are useful both because they allow the taking of locks out of the normal order (with suitable back-off and retry) and because they can be used to help avoid resources in use by other nodes. The normal t (try) lock is just what its name indicates; it is a "try" lock that does not do anything special. The T (**try 1CB**) lock, on the other hand, is identical to the t lock except that the DLM will send a single callback to current incompatible lock holders. One use of the T (**try 1CB**) lock is with the **iopen** locks, which are used to arbitrate among the nodes when an inode's **i_nlink** count is zero, and determine which of the nodes will be responsible for deallocating the inode. The **iopen** glock is normally held in the shared state, but when the **i_nlink** count becomes zero and **->delete_inode**() is called, it will request an exclusive lock with T (**try 1CB**) set. It will continue to deallocate the inode if the lock is granted. If the lock is not granted it will result in the node(s) which were preventing the grant of the lock marking their glock(s) with the D (demote) flag, which is checked at **->drop_inode**() time in order to ensure that the deallocation is not forgotten.

This means that inodes that have zero link count but are still open will be deallocated by the node on which the final **close**() occurs. Also, at the same time as the inode's link count is decremented to zero the inode is marked as being in the special state of having zero link count but still in use in the resource group bitmap. This functions like the ext3 file system3's orphan list in that it allows any subsequent reader of the bitmap to know that there is potentially space that might be reclaimed, and to attempt to reclaim it.

C.6. Glock tracepoints

The tracepoints are also designed to be able to confirm the correctness of the cache control by combining them with the blktrace output and with knowledge of the on-disk layout. It is then possible to check that any given I/O has been issued and completed under the correct lock, and that no races are present.

The **gfs2_glock_state_change** tracepoint is the most important one to understand. It tracks every state change of the glock from initial creation right through to the final demotion which ends with **gfs2_glock_put** and the final NL to unlocked transition. The l (locked) glock flag is always set before a state change occurs and will not be cleared until after it has finished. There are never any granted holders (the H glock holder flag) during a state change. If there are any queued holders, they will always be in the W (waiting) state. When the state change is complete then the holders may be granted which is the final operation before the l glock flag is cleared.

The **gfs2_demote_rq** tracepoint keeps track of demote requests, both local and remote. Assuming that there is enough memory on the node, the local demote requests will rarely be seen, and most often they will be created by umount or by occasional memory reclaim. The number of remote demote requests is a measure of the contention between nodes for a particular inode or resource group.

When a holder is granted a lock, **gfs2_promote** is called, this occurs as the final stages of a state change or when a lock is requested which can be granted immediately due to the glock state already caching a lock of a suitable mode. If the holder is the first one to be granted for this glock, then the f (first) flag is set on that holder. This is currently used only by resource groups.

C.7. Bmap tracepoints

Block mapping is a task central to any file system. GFS2 uses a traditional bitmap-based system with two bits per block. The main purpose of the tracepoints in this subsystem is to allow monitoring of the time taken to allocate and map blocks.

The **gfs2_bmap** tracepoint is called twice for each bmap operation: once at the start to display the bmap request, and once at the end to display the result. This makes it easy to match the requests and results together and measure the time taken to map blocks in different parts of the file system, different file offsets, or even of different files. It is also possible to see what the average extent sizes being returned are in comparison to those being requested.

To keep track of allocated blocks, **gfs2_block_alloc** is called not only on allocations, but also on freeing of blocks. Since the allocations are all referenced according to the inode for which the block is intended, this can be used to track which physical blocks belong to which files in a live file system. This is particularly useful when combined with **blktrace**, which will show problematic I/O patterns that may then be referred back to the relevant inodes using the mapping gained by means of this tracepoint.

C.8. Log tracepoints

The tracepoints in this subsystem track blocks being added to and removed from the journal (**gfs2_pin**), as well as the time taken to commit the transactions to the log (**gfs2_log_flush**). This can be very useful when trying to debug journaling performance issues.

The **gfs2_log_blocks** tracepoint keeps track of the reserved blocks in the log, which can help show if the log is too small for the workload, for example.

The **gfs2_ail_flush** tracepoint (Red Hat Enterprise Linux 6.2 and later) is similar to the **gfs2_log_flush** tracepoint in that it keeps track of the start and end of flushes of the AIL list. The AIL list contains buffers which have been through the log, but have not yet been written back in place and this is periodically flushed in order to release more log space for use by the filesystem, or when a process requests a sync or fsync.

C.9. Glock Statistics

GFS2 maintains statistics that can help track what is going on within the file system. This allows you to spot performance issues.

GFS2 maintains two counters:

- **dcount**, which counts the number of DLM operations requested. This shows how much data has gone into the mean/variance calculations.
- **qcount**, which counts the number of **syscall** level operations requested. Generally **qcount** will be equal to or greater than **dcount**.

In addition, GFS2 maintains three mean/variance pairs. The mean/variance pairs are smoothed exponential estimates and the algorithm used is the one used to calculate round trip times in network code. The mean and variance pairs maintained in GFS2 are not scaled, but are in units of integer nanoseconds.

- srtt/srttvar: Smoothed round trip time for non-blocking operations
- srttb/srttvarb: Smoothed round trip time for blocking operations
- irtt/irttvar: Inter-request time (for example, time between DLM requests)

A non-blocking request is one which will complete right away, whatever the state of the DLM lock in question. That currently means any requests when (a) the current state of the lock is exclusive (b) the requested state is either null or unlocked or (c) the "try lock" flag is set. A blocking request covers all the other lock requests.

Larger times are better for IRTTs, whereas smaller times are better for the RTTs.

Statistics are kept in two **sysfs** files:

- The **glstats** file. This file is similar to the **glocks** file, except that it contains statistics, with one glock per line. The data is initialized from "per cpu" data for that glock type for which the glock is created (aside from counters, which are zeroed). This file may be very large.
- The **lkstats** file. This contains "per cpu" stats for each glock type. It contains one statistic per line, in which each column is a cpu core. There are eight lines per glock type, with types following on from each other.

C.10. References

For more information about tracepoints and the GFS2 **glocks** file, see the following resources:

- For information on glock internal locking rules, see http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/filesystems/gfs2-glocks.txt;h=0494f78d87e40c225eb1dc1a1489acd891210761;hb=HEAD.
- For information on event tracing, see http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/trace/events.txt;h=09bd8e9029892e4e1d48078de4d076e24eff3dd2;hb=HEAD.
- For information on the **trace-cmd** utility, see http://lwn.net/Articles/341902/.

Appendix D. Revision History

Revision 9.1-3 **Thu Mar 23 2017** **Steven Levine**
Update to 6.9 GA publication.

Revision 9.1-2 **Wed Mar 8 2017** **Steven Levine**
Version for 6.9 GA publication.

Revision 9.1-1 **Fri Dec 16 2016** **Steven Levine**
Version for 6.9 Beta publication.

Revision 8.1-5 **Wed Apr 27 2016** **Steven Levine**
Preparing document for 6.8 GA publication.

Revision 8.1-2 **Wed Mar 9 2016** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.8 Beta release

Revision 7.1-5 **Wed Jul 8 2015** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.7

Revision 7.1-4 **Mon Apr 13 2015** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.7 Beta release

Revision 7.0-9 **Wed Oct 8 2014** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.6

Revision 7.0-8 **Thu Aug 7 2014** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.6 Beta release

Revision 6.0-6 **Wed Nov 13 2013** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.5

Revision 6.0-5 **Fri Sep 27 2013** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.5 Beta release

Revision 5.0-7 **Mon Feb 18 2013** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.4

Revision 5.0-5 **Mon Nov 26 2012** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.4 Beta release

Revision 4.0-2 **Thu Mar 28 2012** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.3

Revision 3.0-2 **Thu Dec 1 2011** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.2

Revision 3.0-1 **Mon Sep 19 2011** **Steven Levine**
Initial revision for Red Hat Enterprise Linux 6.2 Beta release

Revision 2.0-1 **Thu May 19 2011** **Steven Levine**

Initial revision for Red Hat Enterprise Linux 6.1

Revision 1.0-1 Wed Nov 15 2010 Steven Levine

Initial revision for Red Hat Enterprise Linux 6

Index

W