Red Hat Enterprise Linux 8

Configuring GFS2 file systems

A guide to the configuration and management of GFS2 file systems

Last Updated: 2019-05-07

Red Hat Enterprise Linux 8 Configuring GFS2 file systems

A guide to the configuration and management of GFS2 file systems

Legal Notice

Abstract

This guide provides information about configuring and managing GFS2 file systems for Red Hat Enterprise Linux 8.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the **Add Feedback** pop-up that appears below the highlighted text, and follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:

  1. Go to the Bugzilla website.

  2. As the Component, use **Documentation**.

  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.

  4. Click **Submit Bug**.

CHAPTER 1. PLANNING A GFS2 FILE SYSTEM DEPLOYMENT

The Red Hat GFS2 file system is a 64-bit symmetric cluster file system which provides a shared namespace and manages coherency between multiple nodes sharing a common block device. A GFS2 file system is intended to provide a feature set which is as close as possible to a local file system, while at the same time enforcing full cluster coherency between nodes. In a few cases, the Linux file system API does not allow the clustered nature of GFS2 to be totally transparent; for example, 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. In most cases however, the functionality of a GFS2 file system is identical to that of a local file system.

The Red Hat Enterprise Linux (RHEL) Resilient Storage Add-On provides GFS2, and it depends on the RHEL High Availability Add-On to provide the cluster management required by GFS2.

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

To get the best performance from GFS2, it is important to take into account the performance considerations which stem from the underlying design. Just like a local file system, GFS2 relies on the page cache in order to improve performance by local caching of frequently used data. In order to maintain coherency across the nodes in the cluster, cache control is provided by the *glock* state machine.

> **IMPORTANT**
>
> Make sure that your deployment of the 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.

1.1. KEY GFS2 PARAMETERS TO DETERMINE

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. 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. GFS2 allows you to add journals dynamically at a later point with the **gfs2_jadd** command as additional servers mount a file system.

Storage devices and partitions

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

Time protocol

Make sure that the clocks on the GFS2 nodes are synchronized. It is recommended that you use the Precision Time Protocol (PTP) or, if necessary for your configuration, 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.

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.

1.2. GFS2 SUPPORT CONSIDERATIONS

Table 1.1, "GFS2 Support Limits" summarizes the current maximum file system size and number of nodes that GFS2 supports.

Table 1.1. GFS2 Support Limits

Maximum number of node	16 (x86, Power8 on PowerVM) 4 (s390x under z/VM)
Maximum file system size	100G on all supported architectures

GFS2 is based on a 64-bit architecture, which can theoretically accommodate an 8 EB file system. If your system requires larger GFS2 file systems than are currently supported, contact your Red Hat service representative.

NOTE

Although a GFS2 file system can be implemented in a standalone system or as part of a cluster configuration, 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).

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 Determing required memory for running fsck.gfs2.

While a GFS2 file system may be used outside of LVM, Red Hat supports only GFS2 file systems that are created on a shared LVM logical volume.

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.

1.3. GFS2 FORMATTING 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.

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

IMPORTANT

Make sure that your deployment of the 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.

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 100TB and the current supported maximum size of a GFS2 file system for 32-bit hardware is 16TB.

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 you 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.

Block Size: Default (4K) Blocks Are Preferred

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.

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.

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.

1.4. 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, some knowledge of how block allocation works can help you optimize performance.

1.4.1. 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 file system 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.

1.4.2. 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 (it is faster to have the node that first opened the file do all the writing of new blocks).

1.4.3. Preallocate, if possible

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

1.5. 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. Red Hat recommends 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.

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

1.6. 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. Red Hat recommends trying equipment before deploying it into full production.

CHAPTER 2. RECOMMENDATIONS FOR GFS2 USAGE

This section provides general recommendations about GFS2 usage.

2.1. 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. For more information on the effect of these arguments on GFS2 file system performance, see GFS2 Node Locking.

2.2. 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.

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 **/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
```

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 **/dev/vg01/lvol0** and is mounted on directory **/mygfs2** with **atime** updates turned off.

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

2.3. 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.

2.4. SELINUX ON GFS2

Use of Security Enhanced Linux (SELinux) with GFS2 incurs a small performance penalty. To avoid this overhead, you may choose not to use SELinux with GFS2 even on a system with SELinux in enforcing mode. When mounting a GFS2 file system, you can 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
```

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.

2.5. 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 effect of this is to force both POSIX locks and flocks from each server to be local: non-clustered, independent of each other. This is necessary because 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. For this reason, when using NFS over GFS2, it is always safest to specify the **-o localflocks** mount option so that NFS can coordinate both POSIX locks and the flocks among all clients mounting NFS.
>
> For all other (non-NFS) GFS2 applications, do not mount your file system using **localflocks**, so that GFS2 will manage the POSIX locks and flocks between all the nodes in the cluster (on a cluster-wide basis). If you specify **localflocks** and do not use NFS, the other nodes in the cluster will not have knowledge of each other's POSIX locks and flocks, thus making them unsafe in a clustered environment

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 High Availability (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.

2.6. SAMBA (SMB OR WINDOWS) FILE SERVING OVER GFS2

You can use Samba (SMB or Windows) file serving from a GFS2 file system with CTDB, which allows active/active configurations.

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.

2.7. CONFIGURING VIRTUAL MACHINES FOR GFS2

When using a GFS2 file system with a virtual machine, it is important that your VM storage settings on each node be configured properly in order to force the cache off. For example, including these settings for **cache** and **io** in the **libvirt** domain should allow GFS2 to behave as expected.

```
<driver name='qemu' type='raw' cache='none' io='native'/>
```

Alternately, you can configure the **shareable** attribute within the device element. This indicates that the device is expected to be shared between domains (as long as hypervisor and OS support this). If **shareable** is used, **cache='no'** should be used for that device.

CHAPTER 3. GFS2 FILE SYSTEMS

This section provides information on the commands and options you use to create, mount, and grow GFS2 file systems.

3.1. GFS2 FILE SYSTEM CREATION

You create a GFS2 file system with the **mkfs.gfs2** command. A file system is created on an activated LVM volume.

3.1.1. The GFS2 mkfs command

The following information is required to run the **mkfs.gfs2** command to create a clustered GFS2 file system:

- Lock protocol/module name, which is **lock_dlm** for a cluster
- Cluster name
- Number of journals (one journal required for each node that may be mounting the file system)

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.

The format for creating a clustered GFS2 file system is as follows. Note that Red Hat does not support the use of GFS2 as a single-node file system.

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

If you prefer, you can create a GFS2 file system by using the **mkfs** command with the **-t** parameter specifying a file system of type **gfs2**, followed by the GFS2 file system options.

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

WARNING

Improperly specifying the *ClusterName:FSName* parameter may cause file system or lock space corruption.

ClusterName

The name of the cluster for which the GFS2 file system is being created.

FSName

The file system name, which can be 1 to 16 characters long. The name must be unique for all **lock_dlm** file systems over the cluster.

NumberJournals

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.

BlockDevice

Specifies a logical or other block device

Table 3.1, "Command Options: **mkfs.gfs2**" describes the **mkfs.gfs2** command options (flags and parameters).

Table 3.1. 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.

Flag	Parameter	Description
-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.
-r	**Megabytes**	Specifies the size of the resource groups in megabytes. The minimum resource group size is 32 megabytes. The maximum resource group size is 2048 megabytes. 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 megabyte resource groups, and bigger file systems will have bigger RGs for better performance.

Flag	Parameter	Description
-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. * **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.
-V		Displays command version information.

3.1.2. Creating a GFS2 file system

The following example creates two GFS2 file systems. For both of these file systems, lock_dlm` is the locking protocol that the file system uses, since this is a clustered file system. Both file systems can be used in the cluster named **alpha**.

For the first file system, file system name is **mydata1**. it contains eight journals and is created on **/dev/vg01/lvol0**. For the second file system, the file system name is **mydata2**. It contains eight journals and is created on **/dev/vg01/lvol1**.

```
# mkfs.gfs2 -p lock_dlm -t alpha:mydata1 -j 8 /dev/vg01/lvol0
# mkfs.gfs2 -p lock_dlm -t alpha:mydata2 -j 8 /dev/vg01/lvol1
```

3.2. MOUNTING A GFS2 FILE SYSTEM

NOTE

You should always use Pacemaker to manage the GFS2 file system in a production environment rather than manually mounting the file system with a **mount** command, as this may cause issues at system shutdown as described in Unmounting a GFS2 file system.

Before you can mount a GFS2 file system, the file system must exist, the volume where the file system exists must be activated, and the supporting clustering and locking systems must be started. After those requirements have been met, you can mount the GFS2 file system as you would any Linux file system.

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**).

3.2.1. Mounting a GFS2 file system with no options specified

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

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

The following is the format for the command to mount a GFS2 file system that specifies mount options.

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

BlockDevice

Specifies the block device where the GFS2 file system resides.

MountPoint

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

The **-o option** argument consists of GFS2-specific options (see Table 3.2, “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 3.2, “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 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 3.2. GFS2-Specific Mount Options

Option	Description

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 multi-host 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.
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. When **errors=withdraw** is specified, which is the default behavior, file system errors will cause the system to withdraw from the file system and make it inaccessible until the next reboot; in some cases the system may remain running.

Option	Description
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.

3.2.2. Unmounting a GFS2 file system

GFS2 file systems that have been mounted manually rather than automatically through Pacemaker will not be known to the system when file systems are unmounted at system shutdown. As a result, the GFS2 resource agent will not unmount the GFS2 file system. After the GFS2 resource agent is shut down, 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 Pacemaker to manage 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.

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.

3.3. BACKING UP A GFS2 FILE SYSTEM

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.

Red Hat recommends making a GFS2 backup by creating a hardware snapshot on the SAN, presenting the snapshot to another system, and backing it up there. The backup system should mount the snapshot with **-o lockproto=lock_nolock** since it will not be in a cluster.

3.4. SUSPENDING ACTIVITY ON A GFS2 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.

The format for the command to suspend activity on a GFS2 file system is as follows.

```
dmsetup suspend MountPoint
```

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

```
# dmsetup suspend /mygfs2
```

The format for the command to end suspension of activity on a GFS2 file system is as follows.

```
dmsetup resume MountPoint
```

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

```
# dmsetup resume /mygfs2
```

3.5. GROWING A GFS2 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 the **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. All nodes in the cluster can then use the extra storage space that has been added.

> **NOTE**
>
> You cannot decrease the size of a GFS2 file system.

The **gfs2_grow** command must be run on a mounted file system. The following procedure increases the size of the GFS2 file system in a cluster that is mounted on the logical volume **shared_vg/shared_lv1** with a mount point of **/mnt/gfs2**.

1. Perform a backup of the data on the file system.

2. If you do not know the logical volume that is used by the file system to be expanded, you can determine this by running the **df *mountpoint*** command. This will display the device name in the following format:
   /dev/mapper/*vg-lv*

   For example, the device name **/dev/mapper/shared_vg-shared_lv1** indicates that the logical volume is **shared_vg/shared_lv1**.

3. On one node of the cluster, expand the underlying cluster volume with the **lvextend** command, using the **--lockopt skiplv** option to override normal logical volume locking.

   ```
   # lvextend --lockopt skiplv -L+1G shared_vg/shared_lv1
   WARNING: skipping LV lock in lvmlockd.
   ```

```
Size of logical volume shared_vg/shared_lv1 changed from 5.00 GiB (1280 extents) to 6.00
GiB (1536 extents).
Logical volume shared_vg/shared_lv1 successfully resized.
```

4. On every additional node of the cluster, refresh the logical volume to update the active logical volume on that node.

    NOTE

    Failing to perform this step on each additional cluster node could make your data unavailable throughout the cluster.

    ```
    # lvchange --refresh shared_vg/shared_lv1
    ```

5. One one node of the cluster, increase the size of the GFS2 file system.

    ```
    # gfs2_grow /mnt/gfs2
    FS: Mount point:            /mnt/gfs2
    FS: Device:                 /dev/mapper/shared_vg-shared_lv1
    FS: Size:                   1310719 (0x13ffff)
    DEV: Length:                1572864 (0x180000)
    The file system will grow by 1024MB.
    gfs2_grow complete.
    ```

6. Run the **df** command on all nodes to check that the new space is now available in the file system. Note that it may take up to 30 seconds for the the **df** command on all nodes to show the same file system size

    ```
    # df -h /mnt/gfs2
    Filesystem                        Size  Used Avail Use% Mounted on
    /dev/mapper/shared_vg-shared_lv1  6.0G  4.5G  1.6G  75% /mnt/gfs2
    ```

3.6. ADDING JOURNALS TO A GFS2 FILE SYSTEM

GFS2 requires one journal for each node in a cluster that needs to mount the file system. If you add additional nodes to the cluster, you can add journals to a GFS2 file system with the **gfs2_jadd** command. 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** command 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 GFS2 file system, you can find out how many journals the GFS2 file system currently contains with the **gfs2_edit -p jindex** command, as in the following example:

```
# gfs2_edit -p jindex /dev/sasdrives/scratch|grep journal
   3/3 [fc7745eb] 4/25 (0x4/0x19): File    journal0
```

```
  4/4 [8b70757d] 5/32859 (0x5/0x805b): File    journal1
  5/5 [127924c7] 6/65701 (0x6/0x100a5): File    journal2
```

The format for the basic command to add journals to a GFS2 file system is as follows.

```
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.

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

```
gfs2_jadd -j 1 /mygfs2
```

CHAPTER 4. 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
>
> 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 more information on disk quotas, see the **man** pages of the following commands:

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

4.1. CONFIGURING GFS2 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.

4.1.1. 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.

To mount a file system with quotas enabled, specify **quota=on** for the **options** argument when creating the GFS2 file system resource in a cluster. For example, the following command specifies that the GFS2 **Filesystem** resource being created will be mounted with quotas enabled.

```
# pcs resource create gfs2mount Filesystem options="quota=on" device=BLOCKDEVICE
directory=MOUNTPOINT fstype=gfs2 clone
```

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.

To mount a file system with quotas disabled, specify **quota=off** for the **options** argument when creating the GFS2 file system resource in a cluster.

4.1.2. 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
```

4.1.3. 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 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 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 following command:

```
# quota testuser
```

You can also set quotas from the command line with the **setquota** command. For information on the **setquota** command, see the **setquota**(8) man page.

4.1.4. 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
```

4.2. MANAGING GFS2 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.

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
```

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.

4.3. KEEPING GFS2 DISK QUOTAS ACCURATE WITH THE QUOTACHECK COMMAND

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** command 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.

4.4. 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. Table 25.2. 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|mountpoint..a`].
```

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.

mountpoint

Specifies the GFS2 file system to which the actions apply.

You can tune the time between synchronizations by specifying a **quota-quantum** mount option.

```
# 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.

The following example synchronizes all the cached dirty quotas from the node it is run on to the on-disk quota file for the file system **/mnt/mygfs2**.

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

This following 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
```

CHAPTER 5. GFS2 FILE SYSTEM REPAIR

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. When the file system is being managed as a Pacemaker cluster resource, you can disable the file system resource, which unmounts the file system. After running the **fsck.gfs2** command, you enable the file system resource again. The *timeout* value specified with the **--wait** option of the **pcs resource disable** indicates a value in seconds.

```
# pcs resource disable --wait=timeoutvalue resource_id
[fsck.gfs2]
# pcs resource enable resource_id
```

To ensure that **fsck.gfs2** command does not run on a GFS2 file system at boot time, you can set the **run_fsck** parameter of the **options** argument when creating the GFS2 file system resource in a cluster. Specifying **"run_fsck=no"** will indicate that you should not run the **fsck** command.

5.1. DETERMING REQUIRED MEMORY FOR RUNNING FSCK.GFS2

Running the **fsck.gfs2** command may require system memory above and beyond the memory used for the operating system and kernel. Larger file systems in particular may require additional memory to run this command.

The following table shows approximate values of memory that may be required to run **fsck.gfs2** file systems on GFS2 file systems that are 1TB, 10TB, and 100TB in size with a block size of 4K.

GFS2 file system size	Approximate memory required to run **fsck.gfs2**
1 TB	0.16 GB
10 TB	1.6 GB
100 TB	16 GB

Note that a smaller block size for the file system would require a larger amount of memory. For example, GFS2 file systems with a block size of 1K would require four times the amount of memory indicated in this table.

5.2. REPAIRING A GFS2 FILESYSTEM

The following shows the format of the **fsck.gfs2** command to repair a GFS2 filesystem.

```
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.

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

```
# 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
```

CHAPTER 6. IMPROVING GFS2 PERFORMANCE

This section provides advice for improving GFS2 peformance.

For general recommendations 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 the Red Hat Customer Portal at https://access.redhat.com/kb/docs/DOC-40821.

6.1. GFS2 FILE SYSTEM DEFRAGMENTATION

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.

6.2. GFS2 NODE LOCKING

In order to get the best performance from a GFS2 file system, it is 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.

Operations such as **read** (buffered), **stat,** and **readdir** only require a shared glock. Operations such as **write** (buffered), **mkdir**, **rmdir**, and **unlink** require an exclusive glock. Direct I/O read/write operations require a deferred glock if no allocation is taking place, or an exclusive glock if the write requires an allocation (that is, extending the file, or hole filling).

There are two main performance considerations which follow from this. First, read-only operations parallelize extremely well across a cluster, since they can run independently on every node. Second, operations requiring an exclusive glock can reduce performance, if there are multiple nodes contending for access to the same inode(s). Consideration of the working set on each node is thus an important factor in GFS2 file system performance such as when, for example, you perform a file system backup, as described in Backing up a GFS2 file system .

A further consequence of this is that we recommend the use of the **noatime** and **nodiratime** mount options with GFS2 whenever possible. This prevents reads from requiring exclusive locks to update the **atime** timestamp.

For users who are concerned about the working set or caching efficiency, GFS2 provides tools that allow you to monitor the performance of a GFS2 file system: Performance Co-Pilot and GFS2 tracepoints.

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.

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**.

6.3. 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.

6.4. 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 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.

6.5. 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.

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 (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 6.1, "Glock flags" shows the meanings of the different glock flags and Table 6.2, "Glock holder flags" shows the meanings of the different glock holder flags.

Table 6.1. 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

Flag	Name	Meaning
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 6.2. 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

Flag	Name	Meaning
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.

WARNING

If you run the **find** command 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** command when you are looking for contended inodes.

Table 6.3, “Glock types” shows the meanings of the different glock types.

Table 6.3. 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

Type number	Lock type	Use
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, 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 reasons 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.

6.6. ENABLING 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.

```
# chattr +j /mnt/gfs2/gfs2_dir/newfile
# 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.

```
# chattr -j /mnt/gfs2/gfs2_dir/newfile
# 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.

```
# chattr -j /mnt/gfs2/gfs2_dir
# lsattr /mnt/gfs2
---------j--- /mnt/gfs2/gfs2_dir
# touch /mnt/gfs2/gfs2_dir/newfile
# lsattr /mnt/gfs2/gfs2_dir
---------j--- /mnt/gfs2/gfs2_dir/newfile
```

CHAPTER 7. DIAGNOSING AND CORRECTING PROBLEMS WITH GFS2 FILE SYSTEMS

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

7.1. GFS2 FILESYSTEM UNAVAILABLE TO A NODE (THE GFS2 WITHDRAW FUNCTION)

The GFS2 *withdraw* function is a data integrity feature of the GFS2 file system that prevents potential file system damage due to faulty hardware or kernel software. If the GFS2 kernel module detects an inconsistency while using a GFS2 file system on any given cluster node, it withdraws from the file system, leaving it unavailable to that node until it is unmounted and remounted (or the machine detecting the problem is rebooted). All other mounted GFS2 file systems remain fully functional on that node. (The GFS2 withdraw function is less severe than a kernel panic, which causes the node to be fenced.)

The main categories of inconsistency that can cause a GFS2 withdraw are as follows:

- Inode consistency error
- Resource group consistency error
- Journal consistency error
- Magic number metadata consistency error
- Metadata type consistency error

An example of an inconsistency that would cause a GFS2 withdraw is an incorrect block count for a file's inode. When GFS2 deletes a file, it systematically removes all the data and metadata blocks referenced by that file. When done, it checks the inode's block count. If the block count is not 1 (meaning all that is left is the disk inode itself), that indicates a file system inconsistency, since the inode's block count did not match the actual blocks used for the file.

In many cases, the problem may have been caused by faulty hardware (faulty memory, motherboard, HBA, disk drives, cables, and so forth). It may also have been caused by a kernel bug (another kernel module accidentally overwriting GFS2's memory), or actual file system damage (caused by a GFS2 bug).

In most cases, the best way to recover from a withdrawn GFS2 file system is to reboot or fence the node. The withdrawn GFS2 file system will give you an opportunity to relocate services to another node in the cluster. After services are relocated you can reboot the node or force a fence with this command.

```
# pcs stonith fence node
```

> **WARNING**
>
> Do not try to unmount and remount the file system manually with the **umount** and **mount** commands. You must use the **pcs** command, otherwise Pacemaker will detect the file system service has disappeared and fence the node.

The consistency problem that caused the withdraw may make stopping the file system service impossible as it may cause the system to hang.

If the problem persists after a remount, you should stop the file system service to unmount the file system from all nodes in the cluster, then perform a file system check with the fsck.gfs2 command before restarting the service with the following procedure.

1. Reboot the affected node.

2. Disable the non-clone file system service in Pacemaker to unmount the file system from every node in the cluster.

   ```
   # pcs resource disable --wait=100 mydata_fs
   ```

3. From one node of the cluster, run the **fsck.gfs2** command on the file system device to check for and repair any file system damage.

   ```
   # fsck.gfs2 -y /dev/vg_mydata/mydata > /tmp/fsck.out
   ```

4. Remount the GFS2 file system from all nodes by re-enabling the file system service:

   ```
   # pcs resource enable --wait=100 mydata_fs
   ```

You can override the GFS2 withdraw function by mounting the file system with the **-o errors=panic** option specified in the file system service.

```
# pcs resource update mydata_fs “options=noatime,errors=panic”
```

When this option is specified, any errors that would normally cause the system to withdraw force a kernel panic instead. This stops the node’s communications, which causes the node to be fenced. This is especially useful for clusters that are left unattended for long periods of time without monitoring or intervention.

Internally, the GFS2 withdraw function works by disconnecting the locking protocol to ensure that all further file system operations result in I/O errors. 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.

7.2. 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 GFS2 data during one of these occurences and open a support ticket with Red Hat Support, as described in Gathering GFS2 data for troubleshooting.

7.3. 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 call traces from GFS2 indicating that the file system has been withdrawn. A withdraw is indicative of file system corruption, a storage failure, or a bug. At the earliest time when it is convenient to unmount the file system, you should perform the following procedure:

  a. Reboot the node on which the withdraw occurred.

  ```
  # /sbin/reboot
  ```

  b. Stop the file system resource to unmount the GFS2 file system on all nodes.

  ```
  # pcs resource disable --wait=100 mydata_fs
  ```

  c. Capture the metadata with the **gfs2_edit savemeta...** command. You should ensure that there is sufficient space for the file, which in some cases may be large. In this example, the metadata is saved to a file in the **/root** directory.

  ```
  # gfs2_edit savemeta /dev/vg_mydata/mydata /root/gfs2metadata.gz
  ```

  d. Update the **gfs2-utils** package.

  ```
  # sudo yum update gfs2-utils
  ```

  e. On one node, run the **fsck.gfs2** command on the file system to ensure file system integrity and repair any damage.

  ```
  # fsck.gfs2 -y /dev/vg_mydata/mydata > /tmp/fsck.out
  ```

  f. After the **fsck.gfs2** command has completed, re-enable the file system resource to return it to service:

  ```
  # pcs resource enable --wait=100 mydata_fs
  ```

  g. Open a support ticket with Red Hat Support. Inform them you experienced a GFS2 withdraw and provide logs and the debugging information generated by the **sosreports** and **gfs2_edit savemeta** commands.
  In some instances of a GFS2 withdraw, commands can hang that are trying to access the file system or its block device. In these cases a hard reboot is required to reboot the cluster.

  For information on the GFS2 withdraw function, see GFS2 filesystem unavailable to a node (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 Gathering GFS2 data for troubleshooting.

7.4. GFS2 FILE SYSTEM DOES NOT MOUNT ON NEWLY ADDED CLUSTER NODE

If you add a new node to a cluster and find that you cannot mount your GFS2 file system on that node, you may have fewer journals on the GFS2 file system than 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. Adding journals to a GFS2 file system.

7.5. 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** command. 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.

7.6. GATHERING GFS2 DATA FOR TROUBLESHOOTING

If your GFS2 file system hangs and does not return commands run against it and you find that you need to open a ticket with Red Hat Support, you should first 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.