Red Hat Enterprise Linux 8

配置 GFS2 文件系统

在高可用性集群中规划、管理、故障排除和配置 GFS2 文件系统

Last Updated: 2024-07-11

Red Hat Enterprise Linux 8 配置 GFS2 文件系统

在高可用性集群中规划、管理、故障排除和配置 GFS2 文件系统

法律通告

摘要

Red Hat Enterprise Linux (RHEL) Resilient Storage 附加组件提供 Red Hat Global File System 2 (GFS2)，它是在共享一个通用块设备的多个节点间管理一致性的集群文件系统。这个标题提供有关规划 GFS2 文件系统部署的信息，以及配置、故障排除和调优 GFS2 文件系统的流程。

目录

对红帽文档提供反馈

我们感谢您对我们文档的反馈。让我们了解如何改进它。

通过 Jira 提交反馈（需要帐户）

1. 登录到 Jira 网站。
2. 单击顶部导航栏中的 **Create**。
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您对改进的建议。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 规划 GFS2 文件系统部署

Red Hat Global File System 2 (GFS2) 是一个 64 位对称集群文件系统，它提供了一个共享名称空间，并管理共享一个常见块设备的多个节点间的一致性。GFS2 文件系统旨在提供尽量与本地文件系统类似的功能，同时在节点间强制实施完整集群一致性。为达到此目的，节点在文件系统资源中使用集群范围的锁定方案。这个锁定方案使用 TCP/IP 等通讯协议来交换锁定信息。

在某些情况下，Linux 文件系统 API 不允许 GFS2 的集群特性完全透明；例如，在 GFS2 中使用 POSIX 锁定的程序应该避免使用 **GETLK** 功能，因为在集群环境中，进程 ID 可能用于集群中的不同节点。然而，多数情况下 GFS2 文件系统的功能和本地文件系统的功能是一样的。

Red Hat Enterprise Linux (RHEL) Resilient Storage Add-On提供 GFS2，它依赖于 RHEL High Availability 附加组件来提供 GFS2 所需的集群管理功能。

gfs2.ko 内核模块实现 GFS2 文件系统，并加载在 GFS2 集群节点上。

要获得最佳 GFS2 性能，请务必考虑基础设计中给出的性能注意事项。和本地文件系统一样，GFS2 依赖于页面缓存以便通过本地缓存来提高经常使用数据的性能。为了在集群中的节点间保持一致性，缓存控制由 *glock* 状态机器提供。

重要

请确定您部署的 Red Hat High Availability Add-On红 满足您的需要并可支持。部署前请咨询权威红帽代表确认您的配置。

1.1. 要决定的主要 GFS2 参数

在安装和配置 GFS2 文件系统前，您应该规划很多关键的 GFS2 参数。

GFS2 节点

决定集群中的哪些节点将挂载 GFS2 文件系统。

文件系统的数量

决定初始创建多少个 GFS2 文件系统。之后，可以添加更多文件系统。

文件系统名称

每个 GFS2 文件系统都应该有一个唯一的名称。这个名称通常与 LVM 逻辑卷名称相同，在挂载 GFS2 文件系统时可作为 DLM 锁定表名称使用。例如，本指南在一些示例流程中使用文件系统名 **mydata1** 和 **mydata2**。

Journals（日志）

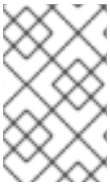
决定 GFS2 文件系统的日志数。GFS2 需要集群中的每个需要挂载文件系统的日志的节点都具有一个日志。例如，如果您有一个 16 个节点的集群，但只需要从两个节点挂载文件系统，则只需要两个日志。GFS2 允许您之后使用 **gfs2_jadd** 工具在其它服务器挂载文件系统时动态添加日志。

存储设备和分区

确定用于在文件系统中创建逻辑卷（使用 **lvmlockd**）的存储设备和分区。

时间协议

请确定 GFS2 节点中的时钟是同步的。推荐使用 Precision Time Protocol (PTP)，或在如果需要，使用由您的 Red Hat Enterprise Linux 系统提供的 Network Time Protocol (NTP) 软件。
不同 GFS2 节点间的系统时钟必须保持在几分钟内，以防止不必要的内节点（inode）时间戳更新。不必要的内节点时间戳更新会严重影响集群的性能。



注意

当同时在同一目录中执行多个生成和删除操作时，GFS2 可能会出现性能问题。如果在系统中造成性能问题，您应该尽可能将节点创建和删除的文件本地化到特定于该节点的目录。

1.2. GFS2 支持注意事项

要有资格获得运行 GFS2 文件系统的集群的支持，您必须考虑 GFS2 文件系统的支持策略。

1.2.1. 最大文件系统和集群大小

下表总结了当前的最大文件系统大小以及 GFS2 支持的节点数量。

表 1.1. GFS2 支持限制

参数	最大值
节点数	16（X86, PowerVM 中的 Power8） 4（z/VM 中的 s390x）
文件系统大小	所有支持的构架都为 100TB

GFS2 是基于 64 位构架，理论上可提供 8 EB 文件系统。如果您的系统需要比目前支持的更大的 GFS2 文件系统,请联络您的红帽服务代表，

在决定文件系统大小时，您应该考虑您的恢复需求。在大型文件系统上运行 **fsck.gfs2** 命令需要很长时间，且消耗大量内存。另外，当磁盘或者磁盘子系统失败时，恢复时间受您的备份介质速度的限制。有关 **fsck.gfs2** 命令需要的内存量的信息，请参阅 确定运行 fsck.gfs2 所需的内存。

1.2.2. 最小集群大小

虽然 GFS2 文件系统可在独立系统或作为集群配置的一部分实现，但红帽不支持将 GFS2 用作单节点文件系统使用，但以下情况除外：

- 红帽支持单节点 GFS2 文件系统，以便根据需要挂载集群文件系统的快照，例如，用于备份。
- 对于辅助站点灾难恢复(DR)节点，支持单节点集群挂载 GFS2 文件系统（其使用 DLM）。这个例外仅用于 DR 目的，不适用于将主集群工作负载传送到辅助站点。
  例如，在主站点离线的情况下，从辅助站点上挂载的文件系统复制数据。但是，不支持将工作负载从主站点直接迁移到单节点集群辅助站点。如果需要将全部工作负载迁移到单节点辅助站点，则辅助站点的大小必须与主站点一样。

  红帽建议当您在单节点集群中挂载 GFS2 文件系统时，您可以指定 **errors=panic** 挂载选项，以便在 GFS2 退出发生时单一节点集群会出现 panic，因为单节点集群在遇到文件系统错误时将无法隔离自己。

红帽支持很多为单一节点优化的高性能单节点文件系统，因此开销通常比集群文件系统低。红帽建议您在只需要单节点挂载文件系统的情况下首选使用这些文件系统，而不是 GFS2。有关 Red Hat Enterprise Linux 8 支持的文件系统的详情，请参考 管理文件系统。

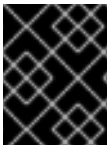
1.2.3. 共享存储注意事项

当在 LVM 之外使用 GFS2 文件系统时，红帽只支持在共享 LVM 逻辑卷中创建的 GFS2 文件系统。

当您将 GFS2 文件系统配置为集群文件系统时，您必须确定集群中的所有节点都可访问共享存储。不支持非对称集群配置（集群中有些节点可访问共享存储而其他节点不能访问共享存储）。这并不需要所有节点都确实挂载到 GFS2 文件系统中。

1.3. GFS2 格式化注意事项

要格式化 GFS2 文件系统以优化性能，您应该考虑这些建议。



重要

请确定您部署的 Red Hat High Availability Add-On红 满足您的需要并可支持。部署前请咨询权威红帽代表确认您的配置。

文件系统大小：越小越好
GFS2 是基于 64 位构架，理论上可提供 8 EB 文件系统。但是，目前支持的 64 位硬件的最大 GFS2 文件系统为 100TB。

注：虽然有可能使用 GFS2 大文件系统，但并不意味着推荐使用它们。GFS2 的一般原则是容量越小越好：最好使用 10 个 1TB 文件系统而不是使用一个 10TB 文件系统。

尽量保持 GFS2 文件系统较小的原因是：

- 备份每个文件系统需要较少的时间。
- 如果您需要使用 **fsck.gfs2** 命令检查文件系统，则需要较少的时间。
- 如果您需要使用 **fsck.gfs2** 命令检查文件系统，则需要较少的内存。

另外，需要较少的资源组来保持更好的性能。

当然，如果您的 GFS2 文件系统太小，可能没有足够空间，这本身就会产生一定后果。在决定大小之前，您应该考虑您自己的具体用例。

块大小：首选默认(4K)块
mkfs.gfs2 命令会根据设备拓扑尝试估算最佳块大小。通常, 4K 块是首选块大小,因为 4K 是 Red Hat Enterprise Linux 的默认页面大小（内存）。与其他一些文件系统不同，GFS2 使用 4K 内核缓冲进行大多数操作。如果您的块大小是 4K，那么内核会在操作缓冲区时进行较少的工作。

建议您使用默认块大小，这样可获得最好的性能。只有在需要有效存储许多小文件时才可能需要使用不同的块大小。

日志大小：默认值(128MB)通常是最优的
当您运行 **mkfs.gfs2** 命令来创建 GFS2 文件系统时，您可以指定日志的大小。如果您没有指定大小，则默认为 128MB，这应该适用于大多数应用程序。

有些系统管理员可能会认为 128MB 过大，并尝试将日志大小降低为最小的 8MB 或者更传统的 32MB。虽然这样可能会正常工作，但可能会严重影响性能。与很多日志记录文件系统类似，GFS2 每次写入元数据时，会在日志发出前将其提交到日志中。这样可确保如果系统崩溃或断电，当日志在挂载时自动重放时，您将恢复所有的元数据。但是，填充 8MB 日志不需要太多的文件系统活动，当日志满时，性能会慢，因为 GFS2 必须等待日志写入存储。

通常建议您使用默认的 128MB 的日志大小。如果您的文件系统非常小（例如 5GB），只有 128MB 日志可能不太现实。如果您有一个大的文件系统并且可以获得更多的存储空间，则使用 256MB 日志可能会提高性能。

资源组的大小和数目
当 GFS2 文件是系统使用 **mkfs.gfs2** 命令创建的，则它会将存储划分为统一的分片，称为资源组。它试图估算一个最佳资源组群大小（从 32MB 到 2GB）。您可以使用 **mkfs.gfs2** 命令的 **-r** 选项覆盖默认设置。

您的最佳资源组群大小取决于您的文件系统如何使用。考虑它的使用情况，以及是否产生大量碎片。

您应该使用不同的资源组大小进行测试，以了解哪些配置具有最佳性能。最好在将 GFS2 部署到完整产品前测试测试集群。

如果您的文件系统有太多资源组，每个资源组又太小，则块分配可能会浪费大量时间来为空闲块搜索成千上万个资源组。文件系统越满，搜索的资源组越多，每个资源组都需要一个集群范围的锁。这会导致性能下降。

然而,，果您的文件系统资源组太少，且每个资源组太大,块分配可能会更频繁地影响到同一资源组锁定，这也会影响到性能。例如：如果您有一个 10GB 文件系统被分为 5 个 2GB 资源组，则集群中的节点会比相同的文件系统被分成 320 个 32MB 的资源组更频繁竞争使用这 5 个资源组。如果您的文件系统计划已被完全占用，则这个问题会更加严重，因为每个块分配可能需要查看多个资源组才能找到可用块。GFS2 尝试以两种方式缓解这个问题：

- 首先，当资源组完全满时，它会记住，并尝试避免在将来分配时检查它们，直到块从中释放。如果您从不删除文件，则这个问题并不严重。但是，如果您的应用程序正在持续删除块并在文件系统中分配最多完全的新块，则竞争会非常高，并且会对性能有严重影响。

- 其次,当在现有文件中添加新块时，GFS2 将尝试将新块分组到与该文件相同的资源组群中。这么做的目的是为了提高性能：在旋转磁盘中，如果存储在物理位置上接近，则操作需要较少的时间。

最糟糕的情况是所有节点都在一个中央目录创建文件时，因为所有节点都会持续努力锁定同一资源组。

1.4. 集群中 GFS2 的注意事项

在决定您系统包含的节点数量时，请注意高可用性和性能之间有一个利弊关系。在有大量节点时，更难以扩展工作负载。因此,红帽不支持将 GFS2 用于部署超过 16 个节点的集群文件系统。

部署集群文件系统不是对单节点部署的"直接"替换。红帽建议您预留约 8-12 周的时间测试新的安装，以便测试该系统并确保其在所需性能水平正常工作。在此阶段，可解决所有性能和功能问题，且所有查询都可提交红帽支持团队。

红帽建议考虑部署集群的用户在部署前由红帽支持审核其配置，以避免以后出现可能的支持问题。

1.5. 硬件注意事项

在部署 GFS2 文件系统时，请考虑以下硬件方面的情况。

- 使用更高质量的存储选项
  GFS2 可以在更便宜的共享存储选项上操作，如 iSCSI 或以太网光纤通道(FCoE)，但如果您购买具有更大缓存容量的高质量存储，您将获得更好的性能。红帽对具有光纤通道互连的 SAN 存储执行高质量的、健全性和性能测试。作为常规规则，最好先部署已经测试的对象。

- 在部署前测试网络设备
  高质量的、更快的网络设备可使集群和 GFS2 速度更快，且更可靠。然而，您并不要一定要买最昂贵的硬件。某些最昂贵的网络交换机在传递多播数据包时会有问题，它们用于传递 **fcntl** 锁

(flocks)，而更便宜的商用网络交换机有时会更快、更可靠。红帽建议您在将其部署到生产环境中前，对设备进行全面测试。

第 2 章 关于 GFS2 使用的建议

在部署 GFS2 文件系统时，您应该考虑各种常规建议。

2.1. 配置 ATIME 更新

每个文件内节点和目录内节点都有三个与之关联的时间戳：

- **ctime** - 节点状态最后一次更改的时间
- **mtime** - 文件（或目录）数据最后一次修改的时间
- **atime** - 文件（或目录）数据最后一次访问的时间

如果默认在 GFS2 和其他 Linux 文件系统上启用了 **atime** 更新，那么每次读取文件时都需要更新其内节点。

因为很少有应用程序使用由 **atime** 提供的信息，因此这些更新可能需要大量不必要的写流量和文件锁定流量。该流量可能会降低性能，因此最好关闭或降低 **atime** 更新的频率。

以下是减少 **atime** 更新效果的方法：

- 使用 **relatime** (相对 atime)挂载，如果以前的 **atime** 更新比 **mtime** 或 **ctime** 更新晚，则其更新 **atime**。这是 GFS2 文件系统的默认挂载选项。
- 使用 **noatime** 或 **nodiratime** 挂载。使用 **noatime** 挂载 会禁用对那个文件系统上文件和目录的 **atime** 更新，而使用 **nodiratime** 的挂载会只禁用对那个文件系统上目录的 **atime** 更新，通常建议尽可能使用 **noatime** 或 **nodiratime** 挂载选项挂载 GFS2 文件系统，而在应用程序允许时，首选 **noatime**。有关这些参数对 GFS2 文件系统性能的影响的更多信息，请参阅 GFS2 节点锁定。

使用以下命令使用 **noatime** Linux 挂载选项挂载 GFS2 文件系统。

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

BlockDevice
指定 GFS2 文件系统所在的块设备。

MountPoint
指定要挂载 GFS2 文件系统的目录。

在这个示例中，GFS2 文件系统位于 **/dev/vg01/lvol0** 上，并挂载到目录 **/mygfs2**，**atime**更新关闭。

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

2.2. VFS 调优选项：研究和实验

和所有 Linux 文件系统一样，GFS2 位于名为虚拟文件系统(VFS)的层上。VFS 为大多数工作负载的缓存设置提供了很好的默认值，多数情况下不需要修改。但是，如果您有一个无法高效运行的工作负载（例如，缓存太大或太小），那么您可以使用 **sysctl**(8) 命令调整 **/proc/sys/vm** 目录中 **sysctl** 文件的值来提高性能。有关这些文件的文档可在内核源树 **Documentation/sysctl/vm.txt** 中找到。

例如，根据具体情况，可以调整 **dirty_background_ratio** 和 **vfs_cache_pressure** 的值。要获取当前值，请使用以下命令：

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

以下命令调整值：

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

您可以通过编辑 **/etc/sysctl.conf** 文件来永久更改这些参数的值。

要找到您的使用案例的最佳值，请在部署到完整产品前检查各种 VFS 选项并在测试集群中进行实验。

2.3. GFS2 中的 SELINUX

将 Security Enhanced Linux(SELinux)与 GFS2 一起使用会导致较小的性能损失。为了避免这种性能损失，您可能会选择在 GFS2 中不使用 SELinux，即使其所在的系统中的 SELinux 处于 enforcing 模式。挂载 GFS2 文件系统时，您可以使用 **mount**(8)手册页中描述的 **context**选项之一来确保 SELinux 不会尝试读取每个文件系统上的 **seclabel** 元素。SELinux 将假设文件系统中的所有内容都使用 **context** 挂载选项中提供的 **seclabel** 元素进行了标记。这也将加快处理速度，因为它避免了可能包含 **seclabel** 元素的扩展属性块的另一个磁盘读取。

例如：在 SELinux 处于 enforcing 模式的系统上，如果文件系统将要包含 Apache 内容，您可以使用以下 **mount** 命令来挂载 GFS2 文件系统。这个标签将应用于整个文件系统，它会保留在内存中，不会被写入磁盘。

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

如果您不确定文件系统是否包含 Apache 内容，您可以使用标签 **public_content_rw_t** 或 **public_content_t**，或者您可以定义一个新标签并定义一个策略。

请注意，在一个 Pacemaker 集群中，应该一直使用 Pacemaker 管理 GFS2 文件系统。您可在创建 GFS2 文件系统资源时指定挂载选项。

2.4. 使用 GFS2 设置 NFS

由于增加了 GFS2 锁定子系统及其集群性质的复杂性，因此通过 GFS2 设置 NFS 需要采取许多预防措施。

警告

如果 GFS2 文件系统是 NFS 导出的，则必须使用 **localflocks** 选项挂载文件系统。因为使用 **localflocks** 选项可防止您从多个位置安全地访问 GFS2 文件系统，并且同时从多个节点导出 GFS2 是不可行的，所以在使用此配置时，支持要求一次只能在一个节点上挂载 GFS2 文件系统。这样做预期的效果是将每个服务器的 POSIX 锁定强制为本地：非集群，相互独立。这是因为 GFS2 试图在集群节点间从 NFS 部署 POSIX 锁定会有很多问题。对于在 NFS 客户端中运行的应用程序，如果两个客户端从不同的服务器挂载，本地化 POSIX 锁定意味着两个客户端可同时拥有相同的锁定，从而导致数据崩溃。如果所有客户端都从一个服务器中挂载 NFS，那么不同服务器单独赋予同一锁定的问题就不存在。如果您不确定是否使用 **localflocks** 选项挂载文件系统，则不应使用此选项。立即联系红帽支持，来讨论合适的配置以避免数据丢失。我们不推荐通过 NFS 导出 GFS2，但在某些情况下并不支持。

对于所有其他（非 NFS）GFS2 应用程序，不要使用 **localflocks** 挂载文件系统，因此 GFS2 将管理集群中所有节点间的 POSIX 锁和 flocks。如果您指定了 **localflocks** 且不使用 NFS，则集群中的其他节点将不知道彼此的 POSIX 锁和 flock，从而使它们在集群环境中不安全。

除锁定注意事项外，您应该在通过 GFS2 文件系统配置 NFS 时考虑以下问题。

- 红帽只支持使用带主动/被动锁定的 NFSv3 进行红帽高可用性附加组件配置，并具有以下特征。此配置为文件系统提供了高可用性(HA)，减少了系统停机，因为故障节点在 NFS 服务器从一个节点故障切换到到另一个节点时不需要执行 **fsck** 命令。
  - 后端文件系统是在 2 到 16 个节点集群中运行的 GFS2 文件系统。
  - NFSv3 服务器的定义是一次从单一集群节点中导出整个 GFS2 文件系统的服务。
  - NFS 服务器可以从一个集群节点故障切换到另外一个节点（主动/被动配置）。
  - 不允许访问 GFS2 文件系统, *除非* 通过 NFS 服务器。这包括本地 GFS2 文件系统访问以及通过 Samba 或者集群的 Samba 访问。通过挂载该文件系统的集群节点在本地访问文件系统可能会导致数据崩溃。
  - 在该系统中不支持 NFS 配额。
- 对于 GFS2 的 NFS 导出，**fsid=** NFS 选项是强制的。
- 如果您的集群出现问题（例如，群集变得不够法定人数且隔离不成功），则集群逻辑卷和 GFS2 文件系统将被冻结，且在集群够法定人数前可能无法访问。在确定一个简单的故障切换解决方案时（比如在这个过程中定义的），您应该考虑这种可能性是否最适合您的系统。

2.5. 使用 GFS2 提供 SAMBA（SMB 或者 WINDOWS）文件

您可以使用带 CTDB 的 GFS2 文件系统提供的 Samba（SMB 或者 Windows）文件，该文件允许主动/主动配置。

不支持从 Samba 以外同时访问 Samba 共享的数据。目前不支持 GFS2 集群租期，这可延迟 Samba 文件服务。有关 Samba 的支持政策的更多信息，请参阅 RHEL Resilient Storage 的支持政策 - ctdb 常规策略和 RHEL Resilient Storage 的支持政策 - 通过其他协议导出 gfs2 内容 。

2.6. 为 GFS2 配置虚拟机

当在虚拟机中使用 GFS2 文件系统时，务必要正确配置每个节点中的虚拟机存储设置，以便强制关闭缓存。例如：在 **libvirt** 域中包括 **cache** 和 **io** 的这些设置应该允许 GFS2 的行为如预期。

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

另外，您可以在 device 元素中配置 **shareable** 属性。这表示在域间应该共享该设备（只要虚拟机监控程序支持和操作系统支持）。如果使用了 **shareable**，则该设备应使用 **cache='no'**。

2.7. 块分配

尽管只写数据的应用程序通常无法无需了解如何或在哪里分配块，但了解块分配如何工作可帮助您优化性能。

2.7.1. 在文件系统中保留空闲空间

当 GFS2 文件系统接近满时，块分配程序在分配新块时会比较困难。因此，所有分配器分配的块往往被压缩到资源组的末尾，或者更有可能出现文件碎片的小块中。该文件的碎片可能会导致性能问题。另外，当 GFS2 文件系统接近满时，GFS2 块分配程序会通过多个资源组花费较长时间搜索，同时添加了锁定竞争，且不一定在该文件系统中有足够剩余空间。这也可能导致性能问题。

由于这些原因，建议您不要运行一个超过 85% 的文件系统，但这个数字会根据工作负载的不同而有所不同。

2.7.2. 在可能的情况下，每个节点分配自己的文件

当为 GFS2 文件系统开发应用程序时，建议您在可能的情况下为每个节点分配它自己的文件。由于分布式锁管理器(DLM)的工作方式，如果所有文件都由一个节点分配，则会有更多锁竞争，而其他节点需要向这些文件添加块。

过去，术语"主锁"过去用来表示节点是锁请求的当前协调者，这些请求源自于本地或来集群中的远程节点。锁请求协调器的术语稍有误导，因为它实际上是一个资源（在 DLM 术语中），与哪个锁请求是排队的、授权的或拒绝的有关。在 DLM 中使用术语的意义上，应该视为 "对等中的第一个"，因为 DLM 是一个对等系统。

在 Linux 内核 DLM 实现中，在其上首先使用锁的节点会成为锁请求的协调者，在那之后没有变化。这是 Linux 内核 DLM 的实现详情，而不是一般的 DLM 属性。将来的更新可能会允许特定锁的锁请求的协调在节点间移动。

协调锁请求的位置对锁请求的启动者是透明的，但对请求延迟的影响除外。当前实现的一个结果是，如果初始工作负载有不平衡的情况（例如，在其他节点执行任何 I/O 命令前，一个节点扫描通过整个文件系统），与执行文件系统初始扫描的节点相比，可能会导致集群中其他节点的锁延迟更高。

与很多文件系统一样，GFS2 分配程序会尝试在同一文件中让块保持接近，以减少磁盘头的移动并提升性能。将块分配给文件的节点可能需要为新块使用和锁定同一资源组（除非该资源组中的所有块都被使用）。如果包含文件的资源组的锁请求协调者分配其数据块（让首先打开该文件的节点做所有新块的写操作会更快）时，文件系统将运行的更快。

2.7.3. 如果可能，预先分配

如果文件预先分配，可以完全避免块分配，且该文件系统可以更有效地运行。GFS2 包含 **fallocate(1)** 系统调用，您可以使用它预分配数据块。

第 3 章 管理 GFS2 文件系统

您可以使用各种命令和选项来创建、挂载、增加和管理 GFS2 文件系统。

3.1. GFS2 文件系统创建

您可以使用 **mkfs.gfs2** 命令创建 GFS2 文件系统。文件系统是在活跃的 LVM 卷中创建的。

3.1.1. GFS2 mkfs 命令

需要以下信息运行 **mkfs.gfs2** 命令来创建集群的 GFS2 文件系统：

- 锁协议/模块名称，即集群的 **lock_dlm**
- 集群名称
- 日志数（每个可能挂载文件系统的节点都需要一个日志）

注意

使用 **mkfs.gfs2** 命令创建 GFS2 文件系统后，您无法缩小文件系统的大小。但是您可以使用 **gfs2_grow** 命令增加现有文件系统的大小。

创建集群 GFS2 文件系统的格式如下。请注意，红帽不支持将 GFS2 作为单节点文件系统使用。

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

如果您愿意，您可以使用 **mkfs** 命令及指定 **gfs2** 文件系统类型的 **-t** 参数，后跟 GFS2 文件系统选项来创建 GFS2 文件系统。

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

警告

不正确的指定 *ClusterName:FSName* 参数可能会导致文件系统或者锁定空间崩溃。

ClusterName

创建 GFS2 文件系统的集群名称。

FSName

文件系统名称，长度为 1-16 个字符。集群上的所有 **lock_dlm** 文件系统的名称必须是唯一的。

NumberJournals

指定由 **mkfs.gfs2** 命令创建的日志数。每个挂载文件系统的节点都需要一个日志。对于 GFS2 文件系统来说，以后可以添加更多的日志而不会增大文件系统。

BlockDevice

指定逻辑设备或其他块设备

下表描述了 **mkfs.gfs2** 命令选项（标签和参数）。

表 3.1. 命令选项：mkfs.gfs2

标记	参数	描述
-c	**Megabytes**	将每个日志的配额更改文件的初始大小设为 **Megabytes**。
-D		启用调试输出。
-h		帮助信息。显示可用选项。
-J	**Megabytes**	以 MB 为单位指定日志大小。默认日志大小为 128MB，最小值为 32MB。最小值为 8MB。较大的日志提高了性能，虽然它们使用的内存超过较小的日志。
-j	**Number**	指定由 **mkfs.gfs2** 命令创建的日志数。每个挂载文件系统的节点都需要一个日志。如果没有指定这个选项，则会生成一个日志。对于 GFS2 文件系统，您可以稍后添加附加日志而不会增大文件系统。
-O		防止 **mkfs.gfs2** 命令在写文件系统前要求确认。
-p	**LockProtoName**	* 指定要使用的锁定协议名称。可以使用的锁定协议包括： * **lock_dlm** - 集群文件系统所需的标准锁定模块。 * **lock_nolock** - 当 GFS2 作为本地文件系统时使用（只有一个节点）。红帽不支持在生产环境中使用 GFS2 作为单节点文件系统。**lock_nolock** 应该仅用于备份目的或辅助站点灾难恢复节点，如 最小集群大小 中所述。使用 **lock_nolock** 时，您必须确保同时只有一个系统正在使用 GFS2 文件系统。
-q		静默。不要显示任何结果。

标记	参数	描述
-r	**Megabytes**	以 MB 为单位指定资源组群大小。资源组群最小值为 32MB。资源组群最大值为 2048MB。在大型的文件系统中，大的资源组群可能会提高性能。如果没有指定，**mkfs.gfs2** 会根据文件系统的大小选择资源组大小：文件系统的平均大小将有 256MB 资源组，较大的文件系统将有较大的资源组，以提高性能。
-t	**LockTableName**	* 在使用 **lock_dlm** 协议时指定 lock 字段的唯一标识符；**lock_nolock** 协议不使用这个参数。 * 这个参数有两个由冒号（无空格）分开的部分，如下所示：**ClusterName:FSName**。 * **ClusterName** 是创建的 GFS2 文件系统的集群名称，只有集群成员才可以使用这个文件系统。 * **FSName**, 文件系统名称，长度可以是 1 到 16 个字符，且该名称在集群中的所有文件系统中必须是唯一的。
-V		显示命令版本信息。

3.1.2. 创建 GFS2 文件系统

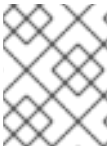
以下示例创建两个 GFS2 文件系统。对于这两个文件系统，lock_dlm' 是文件系统使用的锁定协议，因为这是一个集群的文件系统。这两个文件系统都可用于名为 **alpha** 的集群中。

对于第一个文件系统，文件系统名称为 **mydata1**。它包含八个日志，是在 **/dev/vg01/lvol0** 上创建的。对于第二个文件系统，文件系统名称为 **mydata2**。它包含八个日志，是在 **/dev/vg01/lvol1** 上创建的。

```
# 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. 挂载 GFS2 文件系统

在您挂载 GFS2 文件系统前，该文件系统必须存在，文件系统所在的卷必须被激活，且必须启动支持的集群和锁定系统。满足这些要求后，您可以将 GFS2 文件系统挂载到任意 Linux 文件系统中。



注意

您应该始终在生产环境中使用 Pacemaker 来管理 GFS2 文件系统，而不是使用 **mount** 命令手动挂载文件系统，因为这可能会在系统关闭时导致问题。

要操作文件 ACL，您必须使用 **-o acl** 挂载选项挂载文件系统。如果文件系统是没有使用 **-o acl** 挂载选项挂载的，则用户可以查看 ACL（使用 **getfacl**），但不允许设置它们（使用 **setfacl**）。

3.2.1. 在没有指定选项的情况下挂载 GFS2 文件系统

在这个示例中，**/dev/vg01/lvol0** 上的 GFS2 文件系统被挂载到 **/mygfs2** 目录。

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

3.2.2. 挂载指定挂载选项的 GFS2 文件系统

以下是挂载指定挂载选项的 GFS2 文件系统的命令格式。

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

BlockDevice

指定 GFS2 文件系统所在的块设备。

MountPoint

指定要挂载 GFS2 文件系统的目录。

o 选项 参数包括特定于 GFS2 的选项或可接受的标准 Linux **mount -o** 选项，或两者的组合。多个 **option** 参数用逗号分开，且没有空格。

注意

mount 命令是一个 Linux 系统命令。除了使用这些特定于 GFS2 的选项外，您还可以使用其它标准的 **mount** 命令选项（例如：**-r**）。有关其他 Linux **mount** 命令选项的详情，请查看 Linux **mount** 手册页。

下表描述了挂载时可传递给 GFS2 的可用的特定于 GFS2 的 **-o option** 值。

注意

这个表格包含了只用于本地文件系统选项的描述。但请注意，红帽不支持将 GFS2 作为单节点文件系统使用。红帽将继续支持单节点 GFS2 文件系统来挂载集群文件系统的快照（例如，用于备份目的）。

表 3.2. GFS2 特定挂载选项

选项	描述
acl	允许控制文件 ACL。如果文件系统没有通过 **acl** 挂载选项挂载，则允许用户查看 ACL（使用 **getfacl**），但不允许用户设置它们（使用 **setfacl**）。

选项	描述
data=[ordered|writeback]	当设置了 **data=ordered** 时，由事务修改的用户数据会在事务提交到磁盘前被刷新到磁盘。这样可让用户在崩溃后的文件中看到未初始化的块。当设置了 **data=writeback** 模式时，用户数据在被弄脏后随时写入磁盘；这不提供与 **ordered序** 模式一样的一致性保证，但对某些工作负载它应该稍快。默认值为 **ordered** 模式。
ignore_local_fs **注意：**当共享 GFS2 文件系统时，不应该使用这个选项。	强制 GFS2 将文件系统视为多主机文件系统。默认情况下，使用 **lock_nolock** 会自动打开 **localflocks** 标记。
localflocks **注意：**当共享 GFS2 文件系统时，不应该使用这个选项。	告诉 GFS2 让 VFS（虚拟文件系统）层完成所有 flock 和 fcntl 操作。**localflocks** 标志由 **lock_nolock** 自动打开。
lockproto=LockModuleName	允许用户指定文件系统要使用的锁定协议。如果没有指定 **LockModuleName**，则从文件系统超级块中读取锁定协议名称。
locktable=LockTableName	允许用户指定文件系统要使用的锁定表。
quota=[off/account/on]	为文件系统打开或者关闭配额。将配额设为 **account** 状态会导致文件系统正确维护每个 UID/GID 使用量统计；忽略限制和警告值。默认值为 **off**。
errors=panic|withdraw	当指定 **error=panic** 时，文件系统错误会导致内核 panic。当指定 **error=withdraw** 时（这是默认行为），文件系统错误会导致系统从文件系统退出，并使其无法访问直到下一次重启为止 ; 在某些情况下，系统可能保持运行。
discard/nodiscard	导致 GFS2 为释放的块生成 "discard" I/O 请求。它们可供合适的硬件用来实现精简配置和类似的方案。
barrier/nobarrier	导致 GFS2 在清除日志时发送 I/O 屏蔽。默认值为 **on**。如果底层设备不支持 I/O 障碍，则会自动将此选项变为 **off**。强烈推荐与 GFS2 一起使用 I/O 障碍，除非块设备被设计为不能丢失其写缓存内容（例如，如果它在 UPS 上，或者没有写缓存）。

选项	描述
quota_quantum=*secs*	在将更改的配额信息写入配额文件前将其保存在某个节点的秒数。这是设置此参数的首选方法。该数值是一个大于零的整数。默认为 60 秒。设定为较短的间隔会让配额信息更快地更新，且降低了用户超过其配额的情况出现。较长的间隔可让文件系统操作更迅速有效地包括配额。
statfs_quantum=*secs*	将 **statfs_quantum** 设为 0 是设置 **statfs** 慢版本的首选方法。默认值为 30 秒，它设置 **statfs** 更改将同步到主 **statfs** 文件前的最大时间段。这可以调整，以便允许更快、不太准确的 **statfs** 值或更慢的、更准确的值。当此选项设为 0 时，**statfs** 始终会报告 true 值。
statfs_percent=*value*	在同步回主 **statfs** 文件之前，在 **statfs** 信息中提供有关本地基础的最大百分比变化的绑定，即使时间段尚未过期。如果 **statfs_quantum** 的设置为 0，则忽略此设置。

3.2.3. 卸载 GFS2 文件系统

在系统关闭时卸载文件系统时，系统将无法了解手动挂载而不是通过 Pacemaker 自动挂载的 GFS2 文件系统。因此，GFS2 资源代理将不会卸载 GFS2 文件系统。关闭 GFS2 资源代理后，标准关闭进程会杀死所有剩余的用户进程，包括集群基础结构，并尝试卸载该文件系统。没有集群基础结构则卸载将失败，同时该系统会挂起。

要防止卸载 GFS2 文件系统时系统停滞，您应该进行以下操作之一：

- 总是使用 Pacemaker 管理 GFS2 文件系统。
- 如果使用 **mount** 命令手动挂载了 GFS2 文件系统，请确定在重启或关闭系统前，使用 **umount** 命令手动卸载文件系统。

如果在这些情况下关闭系统的过程中卸载文件系统时停滞，请执行硬件重启。这不太会出现丢失任何数据的情况，因为文件系统在关闭进程早期是同步的。

可使用 **umount** 命令像卸载任何 Linux 文件系统那样卸载 GFS2 文件系统。

注意

umount 命令是一个 Linux 系统命令。有关此命令的信息，请参阅 Linux **umount** 命令手册页。

使用

```
umount MountPoint
```

MountPoint

指定当前挂载 GFS2 文件系统的目录。

3.3. 备份 GFS2 文件系统

无论您的文件系统大小如何，在出现紧急事件时常规备份 GFS2 文件系统是很重要的。很多系统管理员会觉得很安全，因为他们使用了 RAID、多路径、镜像、快照和其它冗余形式对系统进行保护。但是，这些安全措施并不一定是完全足够的。

创建备份的过程可能会有问题，因为备份节点或节点集合通常涉及按顺序读取整个文件系统。如果从单一节点完成此操作，该节点将在缓存中保留所有信息，直到群集里的其他节点开始请求锁定为止。在集群运行时运行这种备份程序会对性能造成负面影响。

完成备份后丢弃缓存可减少其他节点重新拥有其集群锁和缓存所需的时间。但这不是个理想情况，因为其他节点在备份过程开始前已停止缓存它们缓存的数据。您可以在备份完成后使用以下命令丢弃缓存：

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

如果集群中的每个节点备份自己的文件，则速度较快，这样可以在节点之间分割任务。您可以对特定于节点的目录使用 **rsync** 命令的脚本来完成此操作。

红帽建议通过在 SAN 中创建硬件快照生成 GFS2 备份，向另一个系统中显示快照并进行备份。备份系统应该使用 **-o lockproto=lock_nolock** 挂载快照，因为它不在集群中。但请注意，红帽不支持在生产环境中使用 GFS2 作为单节点文件系统。这个选项应该仅用于备份目的或辅助站点灾难恢复节点，如 最小集群大小 中所述。使用这个选项时，您必须确保 GFS2 文件系统同时只被一个系统使用。

3.4. 在 GFS2 文件系统中挂起动作

您可以使用 **dmsetup suspend** 命令挂起对文件系统的写操作。暂停写活动允许使用基于硬件的设备快照捕获处于一致状态的文件系统。**dmsetup resume** 命令终止挂起。

在 GFS2 文件系统中暂停动作的命令格式如下。

```
dmsetup suspend MountPoint
```

本示例会挂起对文件系统 /**mygfs2** 的写操作。

```
# dmsetup suspend /mygfs2
```

GFS2 文件系统中终止活动的命令格式如下。

```
dmsetup resume MountPoint
```

这个示例结束了对文件系统 /**mygfs2** 写操作的挂起。

```
# dmsetup resume /mygfs2
```

3.5. 增大 GFS2 文件系统

gfs2_grow 命令用于在文件系统所在的设备扩展后扩展 GFS2 文件系统。在现有 GFS2 文件系统上运行 **gfs2_grow** 命令，会填满带有新初始化的 GFS2 文件系统扩展的设备末尾和文件系统当前末尾之间的所有备用空间。然后，集群中的所有节点都可以使用添加的额外存储空间。

注意

您不能缩小 GFS2 文件系统的大小。

gfs2_grow 命令必须在挂载的文件系统上运行。下面的流程会增加集群中挂载在逻辑卷 **shared_vg/shared_lv1** 上的、挂载点为 **/mnt/gfs2** 的 GFS2 文件系统的大小。

步骤

1. 对文件系统中的数据进行备份。
2. 如果您不知道文件系统所使用的逻辑卷要扩展，您可以通过运行 **df *mountpoint*** 命令来确定。这将以以下格式显示该设备名称：
   /dev/mapper/*vg-lv*

   例如，设备名称 **/dev/mapper/shared_vg-shared_lv1** 表示逻辑卷是 **shared_vg/shared_lv1**。
3. 在集群的一个节点上，使用 **lvextend** 命令扩展底层集群卷。如果您正在运行 RHEL 8.0，请使用 **--lockopt skiplv** 选项覆盖正常的逻辑卷锁定。对于运行 RHEL 8.1 或更高版本的系统这没必要。对于 RHEL 8.1 及之后的版本，请使用以下命令。

   ```
   # lvextend -L+1G shared_vg/shared_lv1
   Size of logical volume shared_vg/shared_lv1 changed from 5.00 GiB (1280 extents) to 6.00
   GiB (1536 extents).
   WARNING: extending LV with a shared lock, other hosts may require LV refresh.
   Logical volume shared_vg/shared_lv1 successfully resized.
   ```

   对于 RHEL 8.0，请使用以下命令。

   ```
   # 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).
   WARNING: extending LV with a shared lock, other hosts may require LV refresh.
   Logical volume shared_vg/shared_lv1 successfully resized.
   ```
4. 如果您在集群的每个额外节点上运行 RHEL 8.0，请刷新逻辑卷以更新该节点上的活跃逻辑卷。运行 RHEL 8.1 及之后的版本不需要这一步，因为在逻辑卷扩展时自动步骤。

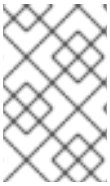
   ```
   # lvchange --refresh shared_vg/shared_lv1
   ```
5. 一个集群节点，增大 GFS2 文件系统的大小。如果没有在所有节点中刷新逻辑卷，则不要扩展该文件系统，否则该文件系统数据可能会在集群中不可用。

   ```
   # 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. 在所有节点上运行 **df** 命令，以检查新空间现在是否在文件系统中可用。请注意，所有节点上运行 **df** 命令可能需要 30 秒才能显示相同的文件系统大小

```
# 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. 在 GFS2 文件系统中添加日志

GFS2 需要为集群中需要挂载该文件系统的每个节点生成一个日志。如果在集群中添加了额外的节点，您可以使用 **gfs2_jadd** 命令将日志添加到 GFS2 文件系统。您可以在任意点动态在 GFS2 文件系统中添加日志，而不扩展基础逻辑卷。**gfs2_jadd** 命令必须运行在挂载的文件系统上，但只需运行在群集中的一个节点上。其它节点可以了解到扩展的发生。



注意

如果 GFS2 文件系统已满，则 **gfs2_jadd** 命令将失败，即使包含文件系统的逻辑卷已扩展并大于文件系统。这是因为在 GFS2 文件系统中，日志是纯文本文件而不是嵌入的元数据，因此只是扩展基础逻辑卷不会为日志提供空间。

在向 GFS2 文件系统添加日志前，您可以使用 **gfs2_edit -p jindex** 命令找出 GFS2 文件系统当前包含多少日志，如下例所示：

```
# 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
```

在 GFS2 文件系统中添加日志的基本命令格式如下。

```
gfs2_jadd -j Number MountPoint
```

Number

指定要添加的新日志数目。

MountPoint

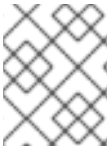
指定要挂载 GFS2 文件系统的目录。

在本例中，一个日志被添加到文件系统的 **/mygfs2** 目录中。

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

第 4 章 GFS2 配额管理

文件系统配额是用来限制某个用户或者组群使用的文件系统空间。在设置前用户或组群没有配额限制。当使用 **quota=on** 或 **quota=account** 选项挂载 GFS2 文件系统时，GFS2 会跟踪每个用户和组群使用的空间，即使没有限制。GFS2 以互动的方式更新配额信息，因此系统崩溃并不需要重建配额使用。为防止性能下降，GFS2 节点只会定期同步配额文件更新。模糊配额核算可让用户或组群稍微超过其设定的限制。为最小化这种情况，GFS2 会在接近硬配额限制时动态缩短同步周期。



注意

GFS2 支持标准 Linux 配额工具。要使用这个功能，您需要安装 **quota** RPM。这是在 GFS2 中管理配额的首选方法，应该在所有使用配额的 GFS2 新部署中使用。

有关磁盘配额的详情，请查看以下命令的 **手册** 页：

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

4.1. 配置 GFS2 磁盘配额

要为 GFS2 文件系统实现磁盘配额，请执行三个步骤：

实现磁盘配额要执行的步骤如下：

1. 使用强制模式或者计数（accounting）模式设置配额。
2. 使用当前块使用信息初始化配额数据库文件。
3. 分配配额策略。（在计数模式中不强制这些策略。）

在以下小节中会详细讨论这些步骤的具体内容。

4.1.1. 在强制或者计数模式下设置配额

在 GFS2 文件系统中，默认会禁用配额。要为文件系统启用配额，请使用指定的 **quota=on** 选项挂载文件系统。

要挂载启用了配额的文件系统，请在集群中创建 GFS2 文件系统资源时，为 **options** 参数指定 **quota=on**。例如：以下命令指定正在创建的 GFS2 **Filesystem** 资源将在启用了配额的情况下挂载。

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

可以跟踪磁盘使用情况，并为每个用户和组群保持配额记账，而不强制限制和警告值。为此，请使用指定的 **quota=account** 选项挂载文件系统。

要挂载禁用了配额的文件系统，请在集群中创建 GFS2 文件系统资源时为 **options** 参数指定 **quota=off**。

4.1.2. 创建配额数据库文件

挂载每个启用了配额的文件系统后，该系统就可以使用磁盘配额。但是，文件系统本身尚未准备好支持配额。下一步是运行 **quotacheck** 命令。

quotacheck 命令检查启用了配额的文件系统，并为每个文件系统构建一个当前磁盘使用情况的表。该表被用于更新磁盘用量的操作系统副本。另外，还会更新文件系统的磁盘配额文件。

要在文件系统上创建配额文件，请使用 **quotacheck** 命令的 **-u** 和 **-g** 选项；对于要初始化的用户和组配额，必须指定这两个选项。例如，如果为 **/home** 文件系统启用了配额，请在 **/home** 目录中创建文件：

```
# quotacheck -ug /home
```

4.1.3. 为每个用户分配配额

最后一步是使用 **edquota** 命令分配磁盘配额。请注意，如果您在记账模式下挂载了文件系统（指定了 **quota=account** 选项），则不强制实施配额。

要为一个用户配置配额，在 shell 提示符中以 root 用户执行以下命令：

```
# edquota username
```

为每个需要配额的用户执行这个步骤。例如，如果为 **/home** 分区（以下示例中的 **/dev/VolGroup00/LogVol02**）启用了配额，并且执行了命令 **edquota testuser**，则在编辑中显示的以下内容为系统的默认值：

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

注意

edquota 使用 **EDITOR** 环境变量定义的文本编辑器。要更改编辑器，请将 **~/.bash_profile** 文件中的 **EDITOR** 环境变量设为您选择的编辑器的完整路径。

第一列是启用了配额的文件系统的名称。第二列显示目前该用户使用的块数。下面的两列是为该用户在文件系统中设定软限制和硬限制。

软块限制定义可以使用的最大磁盘空间量。

硬块限制是用户或者组群可以使用的绝对最大磁盘空间量。达到这个限制后，就无法再使用其他磁盘空间。

GFS2 文件系统并不为内节点维护配额，因此这些列不适用于 GFS2 文件系统，并为空。

如果值为 0，则代表没有设定那个限制。在文本编辑器中更改限制。例如：

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

要验证是否为该用户设定了配额，使用以下命令：

```
# quota testuser
```

您也可以使用 **setquota** 命令从命令行设置配额。有关 **setquota** 命令的详情，请查看 **setquota**(8)手册页。

4.1.4. 为每个组群分配配额

配额可以针对单独组群进行分配。请注意，如果您已在计数模式下挂载了文件系统（指定了 **account=on** 选项），则不强制实施配额。

要为 **devel** 组设置组配额（在设置组配额前组必须存在），请使用以下命令：

```
# edquota -g devel
```

这个命令在文本编辑器中显示该组群的现有配额：

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

GFS2 文件系统并不为内节点维护配额，因此这些列不适用于 GFS2 文件系统，并为空。修改限制后保存文件。

要验证是否设定了组群配额，使用以下命令：

```
$ quota -g devel
```

4.2. 管理 GFS2 磁盘配额

如果使用了配额，则需要对其进行维护。大部分是查看是否超过了配额，并确定配额是准确的。

如果用户重复超过配额，或者持续达到其软限制，系统管理员可根据用户类型以及影响其工作的空间数量而决定一些选择。管理员可帮助用户决定如何使用较少的磁盘空间，或者增加用户的磁盘配额。

您可以通过运行 **repquota** 工具来创建磁盘使用情况报告。例如，命令 **repquota /home** 生成此输出：

```
*** 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
```

要查看所有启用了配额的文件系统的磁盘使用情况报告（选项 **-a**），请使用命令：

```
# repquota -a
```

每个用户后面显示的 **--** 是一种用来确定是否超过块限制的快速方法。如果超过了块软限制，则 **+** 会出现在输出中第一个 **-** 的位置 。第二个 **-** 表示 inode 限制，但 GFS2 文件系统不支持 inode 限制，因此该字符将保持为 **-**。GFS2 文件系统不支持宽限期，因此 **grace** 列将保持空白。

请注意，无论底层文件系统如何，对通过 NFS 的**repquota** 命令都不支持。

4.3. 使用 QUOTACHECK 命令准确保留 GFS2 磁盘配额

如果您在禁用了配额的情况下运行了一段时间后，对文件系统启用了配额，您应该运行 **quotacheck** 命令以创建、检查和修复配额文件。另外，如果您认为配额文件可能不准确，您可能要运行 **quotacheck** 命令，因为系统崩溃后文件系统未完全卸载时会发生这种情况。

有关 **quotacheck** 命令的详情，请查看 **quotacheck(8)** 手册页。

注意

当文件系统在所有节点上相对空闲时运行 **quotacheck**，因为磁盘活动可能会影响计算的配额值。

4.4. 使用 QUOTASYNC 命令同步配额

GFS2 在其磁盘的内部文件中保存所有配额信息。GFS2 节点不会在每次写入文件系统时更新这个配额文件，默认情况下它每 60 秒更新一次配额文件。这是避免节点间在写入配额文件时出现竞争所必需的，这会导致性能下降。

随着用户或组群接近其配额限制，GFS2 会动态减少更新配额文件的时间间隔以避免超过限制。配额同步之间的正常时间段是一个可调整的参数 **quota_quantum**。您可以使用 **quota_quantum=** 挂载选项更改默认值 60 秒，如 挂载指定了挂载选项的 GFS2 文件系统 中"特定于 GFS2 的挂载选项"表中所述。

必须在每个节点上及每次挂载文件系统时设置 **quota_quantum** 参数。对 **quota_quantum** 参数的更改在卸载过程中并不持久。您可以使用 **mount -o remount** 来更新 **quota_quantum** 值。

您可以使用 **quotasync** 命令在 GFS2 执行自动更新期间将配额信息从节点同步到磁盘上的配额文件。使用 **同步配额信息**

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

u

同步用户配额文件。

g

同步组配额文件

a

同步所有当前启用配额并支持同步的文件系统。当没有 -a 时，应该指定一个文件系统挂载点。

mountpoint

指定要执行操作的 GFS2 文件系统。

您可以通过指定 **quota-quantum** 挂载选项来调整同步之间的时间。

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

MountPoint

指定要执行操作的 GFS2 文件系统。

secs

指定 GFS2 对常规配额文件进行同步之间的新时间周期。数值越小，竞争越高，性能会下降。

以下示例将所有缓存的脏配额从其运行的节点同步到文件系统 **/mnt/mygfs2** 的磁盘上的配额文件。

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

在逻辑卷 **/dev/volgroup/logical_volume** 上重新挂载文件系统 **/mnt/mygfs2** 时，以下示例将该文件系统的常规配额文件更新的默认时间段更改为一小时（3600 秒）。

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

第 5 章 GFS2 文件系统修复

当挂载文件系统的节点失败时，文件系统日志允许快速恢复。但是，如果存储设备断电或者断开物理连接，则会发生文件系统崩溃。（无法使用日志进行存储子系统失败修复。）当这种损坏发生时，您可以使用 **fsck.gfs2** 命令恢复 GFS2 文件系统。

重要

只能对从所有节点都卸载了的文件系统运行 **fsck.gfs2** 命令。当文件系统作为 Pacemaker 集群资源进行管理时，您可以禁用文件系统资源，这会卸载文件系统。运行 **fsck.gfs2** 命令后，您可以重新启用文件系统资源。使用 **pcs resource disable** 的 **--wait** 选项指定的 *timeout* 超时值表示一个以秒为单位的值。

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

请注意，即使文件系统是资源组的一部分，如在加密文件系统部署中一样，为了文件系统上运行 fsck 命令，您只需禁用文件系统资源。您不能禁用整个资源组。

为确保 **fsck.gfs2** 命令不会在启动时在 GFS2 文件系统上运行，您可以在在集群中创建 GFS2 文件系统资源时设置 **options** 参数的 **run_fsck** 参数。指定 **"run_fsck=no"** 将表示您不应运行 **fsck** 命令。

5.1. 确定运行 FSCK.GFS2 所需的内存

运行 **fsck.gfs2** 命令可能需要超过操作系统和内核所使用的内存的系统内存。较大的文件系统可能需要额外内存才能运行这个命令。

下表显示了在 GFS2 文件系统上需要运行 **fsck.gfs2** 文件系统的可能的内存值，大小为 1TB、10TB 和 100TB，块大小为 4K。

GFS2 文件系统大小	运行 **fsck.gfs2** 所需的大概内存
1 TB	0.16 GB
10 TB	1.6 GB
100 TB	16 GB

请注意，如果文件系统的块大小较小，则需要更大的内存。例如：块大小为 1K 的 GFS2 文件系统需要这个表所示的内存量的四倍。

5.2. 修复 GFS2 文件系统

修复 GFS2 文件系统的 **fsck.gfs2** 命令的格式如下：

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

-y

-y 标志要求所有问题都回答 **yes**。指定了 **-y** 标志后，**fsck.gfs2** 命令在进行更改前不会提示您输入答案。

BlockDevice

指定 GFS2 文件系统所在的块设备。

在这个示例中修复了位于块设备 **/dev/testvg/testlv** 上的 GFS2 文件系统。所有关于修复的查询都自动回答为 **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
```

第 6 章 提高 GFS2 性能

您可以分析 GFS2 配置的许多方面来改进文件系统性能。

有关使用高可用性附加组件和 Red Hat Global File System 2(GFS2)部署和升级 Red Hat Enterprise Linux 集群的常规建议，请查看红帽客户门户网站上的文章 Red Hat Enterprise Linux Cluster、高可用性和 GFS 部署最佳实践。

6.1. GFS2 文件系统进行碎片整理

虽然 Red Hat Enterprise Linux 上没有 GFS2 的碎片整理工具，但您可以使用 **filefrag** 工具标识单个文件，将它们复制成临时文件，并重命名临时文件来替换原始文件来将它们进行碎片整理。

6.2. GFS2 节点锁定

要获得最佳 GFS2 文件系统性能，了解其操作的基本原理非常重要。单一节点文件系统带有一个缓存，其目的是在频繁使用时减少磁盘访问所造成的访问延迟。在 Linux 中，页面缓存（以及过去的缓冲缓存）提供了这个缓存功能。

使用 GFS2 时，每个节点都有其自身的页面缓存，该缓存中可能包含部分磁盘数据。GFS2 使用一个称为 *glocks*（发音为 gee-locks）的锁定机制，用于维护节点间缓存的完整性。glock 子系统提供了一个缓存管理功能，它使用 *分布式锁管理器*（DLM）作为底层通信层。

glocks 在每个内节点中为缓存提供保护，因此在每个内节点中都有一个锁定用来控制缓存层。如果该 glock 在共享模式（DLM 锁模式）下被授权：PR)然后，那个 glock 下的数据会同时被缓存在一个或多个节点上，因此所有节点都对数据具有本地访问权限。

如果 glock 在专用模式（DLM 锁模式）下被授权：EX）然后，只有一个节点可以在那个 glock 下缓存数据。此模式被修改数据（如 **write** 系统调用）的所有操作使用。

如果另一个节点请求 glock 但无法立即获得，那么 DLM 会向该节点发送一条信息，或者向目前使用 glock 阻止新请求的节点发送一条信息，要求它们释放其锁定。释放 glock 可能需要较长时间（与大多数文件系统操作相比）。释放一个共享的 glock 只需要将缓存设置为无效，它的速度比较快，并与缓存数据的数量相对应。

释放一个专用 glock 需要清除日志，并将所有更改的数据写入磁盘，然后象共享 glock 一样使缓存失效。

单一节点文件系统与 GFS2 之间的区别在于，单一节点文件系统只有一个缓存，GFS2 在每个节点中都有单独的缓存。在这两种情况下，访问缓存的数据的延迟程度类似，但如果另一个节点之前缓冲了同样的数据，GFS2 对未缓存的数据访问的时间延迟要大得多。

诸如 **read** (缓冲的)、**stat** 和 **readdir** 等操作只需要一个共享 glock。**write** (缓冲的)、**mkdir**、**rmdir** 和 **unlink** 等操作需要一个专用 glock。如果没有分配发生，则直接 I/O 读写操作需要一个延迟 glock，如果写需要一个分配（即扩展该文件或漏洞填充），则直接 I/O 读写操作需要一个专用 glock。

这有两个主要的性能注意事项。首先，只读操作可在集群中并行化，因为它们可以在每个节点中独立运行。其次，如果有多个节点竞争访问同一个 inode，则需要一个专用 glock 的操作会降低性能。因此对每个节点上工作集的考虑是 GFS2 文件系统性能的一个重要因素，比如当您执行文件系统备份时，如 备份 GFS2 文件系统 中所述。

这样做的另一个后果是，我们建议尽可能对 GFS2 使用 **noatime** 或 **nodiratime** 挂载选项，在应用程序允许这样做的情况下，首选 **noatime**。这可防止读需要专用锁来更新 **atime** 时间戳。

对于关注工作集或缓存效率的用户，GFS2 提供了可让您监控 GFS2 文件系统性能的工具：Performance Co-Pilot 和 GFS2 追踪点。

注意

由于采用 GFS2 缓存的方法，在出现以下任意情况之一时都会获得最佳性能：

- 在所有节点上以只读方式使用内节点。
- 只在单一节点中写入或修改内节点。

请注意，在创建和删除文件的过程中插入和删除目录条目计算为写入目录内节点。

有可能会打破这个规则，但并不常发生。过度忽略这个规则会对性能有严重影响。

如果您在具有读/写映射的 GFS2 上 **mmap**（）一个文件，但只从该文件读，则这只计数为一次读。

如果您没有设置 **noatime mount** 参数，则读取也将会导致更新文件时间戳的写操作。我们建议所有 GFS2 用户都应该使用 **noatime** 挂载，除非它们对 **atime** 有特定的要求。

6.3. POSIX 锁定的问题

当使用 Posix 锁定时，您应该考虑以下问题：

- 使用 Flocks 比使用 Posix 锁要快。
- 在 GFS2 中使用 Posix 锁的程序应该避免在集群环境中使用 **GETLK** 函数，进程 ID 可能用于集群中的不同节点。

6.4. 使用 GFS2 的性能调整

通常可以更改有问题的应用程序保存其数据的方法，以便获得显著的性能优势。

典型的问题程序示例为电子邮件服务器。通常为每个用户使用一个包含文件的 spool 目录(**mbox**)，或者为每个用户指定一个包含每条消息的一个目录(**maildir**)来进行布局。当请求到达 IMAP 时，理想的情况是为每个用户赋予一个与特定节点的亲和性。这样一来，他们查看和删除电子邮件信息的请求就会从那个节点的缓存中提供。如果那个节点失败，则可以在不同节点中重启该会话。

当邮件通过 SMTP 方法到达时，那么独立的节点可以再次设置，在默认情况下把特定用户的邮件发送到一个特定的节点。如果默认节点没有启动，则信息可以通过接收节点直接保存到用户的邮件 spool 中。同样，这个设计的目的是在通常情况下只在一个节点中保留缓存的文件集合，但在节点失败时允许直接访问。

这个设置可以最佳地使用 GFS2 的页面缓存，同时使故障对应用程序透明，无论 **imap** 或 **smtp**。

备份通常是另一个值得关注的地方。如果可能最好直接从节点备份每个节点的工作组件，这样可缓存具体的内节点组。如果您有一个在特定时间点定期运行的备份脚本，且发现会因为 GFS2 中运行的其他应用程序而造成反应延迟时，则通常代表没有有效地使用页面缓存。

如果您可以在运行备份脚本时停止运行应用程序，则这就不是一个问题。相反，如果备份脚本只从一个节点中执行备份，则完成后大部分的文件系统内容会在那个节点中被缓存，这会对以后从其他节点访问有一定的性能损失。通过以下命令在备份完成后丢弃备份节点上的 VFS 页面缓存可将此影响缓解到一定程度：

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

然而，这并非是一个好的解决方案，因为它要保证每个节点中的工作集合是共享的，或多数操作是集群的只读操作，或大量操作是从一个单一节点进行访问的。

6.5. 使用 GFS2 锁定转储对 GFS2 性能进行故障排除

如果由于使用 GFS2 缓存而导致集群性能增加，您可能会看到大量和增长的 I/O 等待时间。您可以使用 GFS2 的锁转储信息来确定问题的原因。

GFS2 锁转储信息可以从 **debugfs** 文件中收集，该文件可在以下路径名称中找到，假设 **debugfs** 被挂载到 **/sys/kernel/debug/** ：

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

文件的内容是一系列行。每行以 G 开头: 表示一个 glock，后面的行缩进一个空格，代表文件中紧靠它们前面的一项与 glock 相关的信息。

使用 **debugfs** 文件的最佳方法是，在应用程序遇到问题时使用 **cat** 命令获得一份文件完整内容的副本（如果您有大量 RAM 和大量缓存的 inode，则可能需要很长时间），然后在以后查看得到的数据。

注意

创建 **debugfs** 文件的两个副本会很有用，一个副本需要几秒钟甚至一两分钟。通过比较与同一个 glock 号相关的两个跟踪中的持有者信息，您可以判断工作负载是否有进展（只是缓慢）或者它是否被卡住（这一直是一个 bug，应该立即报告给红帽支持）。

debugfs 文件中以 H 开头的行:（持有者）代表锁请求已授权或等待授权。持有者行 f 上的 flags 字段：显示什么：'W' 标志指的是等待请求，"H"标记指的是已授予的请求。有大量等待请求的 glock 可能是正在经历特殊竞争的请求。

下表显示了 glock 标记和 glock 持有者标记的含义。

表 6.1. Glock 标记

标志	名称	含义
b	Blocking	当设置了 locked 标记时有效，并指示可能会阻止来自 DLM 请求的操作。对于演示操作和"try"锁定，这个标记会被清除。这个标志的目的是允许收集 DLM 响应时间的统计数据，与其他节点与降级锁无关。
d	Pending demote	推迟（远程）降级请求
D	Demote	降级请求（本地或者远程）
f	Log flush	释放这个 glock 前需要提交该日志
F	Frozen	回复忽略的远程节点 -- 恢复正在进行。这个标志与文件系统冻结无关，它使用不同的机制，但仅在恢复中使用。
i	Invalidate in progress	这个 glock 下无效页面的进程中

标志	名称	含义
I	Initial	设定何时将 DLM 锁定与这个 glock 关联
l	Locked	这个 glock 处于更改状态中
L	LRU	glock 出现在 LRU 列表中时设置
o	Object	设置 glock 何时与对象关联（即类型为 2 glocks 的内节点，以及类型为 3 的 glock 的资源组）
p	Demote in progress	glock 正在响应降级请求
q	Queued	设定当拥有者排队到 glock 时，并在 glock 保留时清除，但没有剩余所有者。作为算法的一部分使用，计算 glock 的最小保留时间。
r	Reply pending	从远程节点接收的回复正在等待过程中
y	Dirty	释放这个 glock 前需要冲刷到磁盘中的数据

表 6.2. Glock 拥有者标记

标志	名称	含义
a	Async	不要等待 glock 结果（以后轮询结果）
A	Any	接受任意兼容锁定模式
c	No cache	取消锁定时，立即降级 DLM 锁定
e	No expire	忽略随后的锁定取消请求
E	exact	必须有准确的锁定模式
F	First	设定赋予这个锁定的第一个拥有者
H	Holder	表示赋予请求的锁定
p	Priority	在队列头启用 ENQUEUE 拥有者
t	Try	"try" 锁定

标志	名称	含义
T	Try 1CB	发送回调的 "try" 锁定
W	Wait	等待请求完成的设置

在确定了导致问题的 glock 后，下一步就是找出与其相关的 inode 。glock 号（n: 在 G:行上）表示这个。它是格式 *type/number*，如果 *type* 为 2，则 glock 是一个内节点 glock， *number* 是一个内节点号。要跟踪 inode ，您可以运行 **find -inum *number***，其中 *number* 是 glocks 文件中从十六进制格式转换为十进制格式的 inode 号。

警告

如果您在正在经历锁竞争的文件系统上运行 **find** 命令，您可能会使问题更糟。当您查找竞争的 inode 时，最好在运行 **find** 命令前停止应用程序。

下表显示了不同 glock 类型的含义。

表 6.3. Glock 类型

类型号	锁定类型	使用
1	Trans	事务锁定
2	Inode	内节点元数据和数据
3	Rgrp	源组群元数据
4	Meta	超级块
5	Iopen	内节点最近探测
6	Flock	**flock**(2)syscall
8	Quota	配额操作
9	Journal	Journal mutex

如果识别出的 glock 是不同的类型，那么最可能是类型 3（资源组）。如果您在正常负载时看到大量进程正在等待其它 glock 类型，请向红帽支持提交报告。

如果您看到在资源组锁定中排队了大量等待的请求，那么可能有很多原因。一个原因可能是，在文件系统中，相对于资源组有大量的节点。另一个可能的原因是，文件系统可能接近完全被使用（平均来讲，需要较长的搜索时间）通过添加更多存储并使用 **gfs2_grow** 命令扩展文件系统来改善这两种情况。

6.6. 启用数据日志

通常 GFS2 只将元数据写入其日志中。文件内容会随后由内核定期进行的同步（同步会清除文件系统缓冲）写入磁盘。对一个文件的 **fsync（）** 调用会导致文件的数据被立即写入磁盘。当磁盘报告所有数据已安全写入时，调用会返回。

对于非常小的文件，数据日志可导致减少的 **fsync（）** 时间，因为除了写入元数据外，文件数据还会写入日志。随着文件的增大，这个优势会明显降低。在启用数据日志的情况下，对中等和较大文件的写操作会较慢。

依赖于 **fsync（）** 同步文件数据的应用程序可能会通过数据日志来提高性能。在标记的目录及其所有子目录中创建的 GFS2 文件可自动启用数据日志。现有的长度为零的文件也可以打开或者关闭数据日志。

在一个目录上启动数据日志会把目录设定为 "inherit jdata"，这代表以后所有在这个目录中生成的文件和目录都会进行日志。您可以使用 **chattr** 命令对文件启用和禁用数据日志。

以下命令对 **/mnt/gfs2/gfs2_dir/newfile** 文件启用数据日志，然后检查是否正确设置了标志。

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

以下命令对 **/mnt/gfs2/gfs2_dir/newfile** 文件禁用数据日志，然后检查是否正确设置了标志。

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

您还可以使用 **chattr** 命令对目录设置 **j** 标志。当您为某个目录设定此标记时，以后在那个目录中生成的所有文件和目录都会进行日志操作。以下命令集对 **gfs2_dir** 目录设置 **j** 标志，然后检查是否正确设置了标志。之后，命令会在 **/mnt/gfs2/gfs2_dir** 目录中创建一个名为 **newfile** 的新文件，然后检查是否为该文件设置了 **j** 标志。因为为该目录设置了 **j** 标志，因此 **newfile** 应该页启用了日志。

```
# 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
```

第 7 章 诊断并修正 GFS2 文件系统的问题

以下流程描述了一些常见的 GFS2 问题，并提供了有关如何解决它们的信息。

7.1. GFS2 文件系统对节点不可用（GFS2 撤回功能）

GFS2 *withdraw* 功能是 GFS2 文件系统的数据完整性功能，可防止因为硬件或者内核软件造成潜在的文件系统损坏。如果 GFS2 内核模块在任何指定集群节点中使用 GFS2 文件系统时检测到不一致的情况，它会从该文件系统中撤回（withdraw），并使它在相应的节点上不可用，直到卸载并重新挂载该节点（或者被探测到有问题的机器被重启）。所有其他挂载的 GFS2 文件系统在那个节点中仍能完全正常工作。（GFS2 撤回功能没有内核的 panic 严重，内核 panic 会导致该节点被隔离。）

可能导致 GFS2 撤回的主要原因：

- 内节点一致性错误
- 资源组一致性错误
- 日志一致性错误
- Magic number 元数据一致性错误
- 元数据类型一致性错误

因为不一致性导致 GFS2 撤回的一个示例是，文件内节点的不正确的块计数。当 GFS2 删除一个文件时，它会系统性地删除该文件引用的所有数据和元数据块。在完成后，它会检查内节点的块计数。如果块计数不是 1（1 代表所有剩下的都是磁盘内节点），这表示文件系统不一致，因为内节点的块数量与该文件使用的实际块不匹配。在很多情况下，问题可能是由硬件故障造成的（内存、主板、HBA、磁盘驱动器、电缆等出问题）。也可能是由内核的程序漏洞（另一个内核模块意外覆盖 GFS2 内存）或者实际文件系统损坏（由 GFS2 错误导致）造成的。

在大多数情况下，从 GFS2 文件系统中恢复的最佳方法是重新引导或者隔离该节点。撤回的 GFS2 文件系统将给您一个将服务重新定位到集群的另一个节点的机会。重新定位服务后，您可以重新引导节点或使用这个命令强制进行隔离。

```
pcs stonith fence node
```

警告

不要尝试使用 **umount** 和 **mount** 命令手动卸载和重新挂载文件系统。您必须使用 **pcs** 命令，否则 Pacemaker 会检测到文件系统服务已消失，并隔离节点。

导致撤回的一致性问题可能会导致无法停止文件系统服务，因为它可能导致系统挂起。

如果重新挂载后问题仍然存在，您应该停止该文件系统服务以从集群中的所有节点卸载文件系统，然后在按照以下流程重启服务前使用 **fsck.gfs2** 命令执行文件系统检查。

1. 重新引导受影响的节点。
2. 在 Pacemaker 中禁用非克隆文件系统服务，以从集群中的每个节点卸载文件系统。

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

3. 从集群的一个节点对文件系统设备运行 **fsck.gfs2** 命令，来检查和修复文件系统损坏。

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

4. 通过重新启用文件系统服务从所有节点中重新挂载 GFS2 文件系统：

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

您可以使用文件系统服务中指定的 **-o error=panic** 选项挂载文件系统来覆盖 GFS2 撤回功能。

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

当指定这个选项时，所有会导致系统撤回的错误都是强制造成一个内核 panic。这样可停止节点的通讯，从而可以隔离该节点。这对于长时间无人值守、而无需监控或干预的集群特别有用。

GFS2 撤回的内部工作原理是，断开锁定协议以确保以后所有的文件系统操作都会出现 I/O 错误。因此，当发生撤回时，通常会在系统日志中看到来自设备映射器的 I/O 错误。

7.2. GFS2 文件系统挂起，需要重启一个节点

如果您的 GFS2 文件系统挂起且没有返回针对它运行的命令，但重启一个特定节点会使系统恢复正常，这可能表示有锁定问题或者存在程序漏洞。如果出现这种情况，在出现这个问题时收集 GFS2 数据并创建一个红帽支持问题单，如收集 GFS2 数据进行故障排除所述。

7.3. GFS2 文件系统挂起，需要重启所有节点

如果您的 GFS2 文件系统挂起且不会返回针对它运行的命令，并需要您重启集群中的所有节点才可以使用它，检查以下问题。

- 您可能有一个失败的隔离（fence）。GFS2 文件系统将停滞，以保证在出现隔离失败时的数据完整性。检查信息日志，查看在挂起时是否有失败的隔离。确定正确配置了隔离。
- GFS2 文件系统可能已经撤回。检查消息日志中的词 **withdraw**，并检查 GFS2 中的任何信息，以及调用表示已经撤回的文件系统的 GFS2 的跟踪。撤回通常代表文件系统崩溃、存储失败或存在程序漏洞。根据具体情况，尽早进行以下操作来卸载文件系统：

  a. 重启发生撤回的节点。

  ```
  # /sbin/reboot
  ```

  b. 停止该文件系统资源在所有节点中卸载 GFS2 文件系统。

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

  c. 使用 **gfs2_edit savemeta... 捕获元数据** 命令所需的文件。您应该确定有足够的空间来容纳该文件，在某些情况下，所需空间可能比较大。在本例中，元数据被保存到 **/root** 目录中的一个文件中。

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

d. 更新 **gfs2-utils** 软件包。

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

e. 在一个节点上，对文件系统运行 **fsck.gfs2** 命令，以确保文件系统的完整性，并修复任何损坏。

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

f. 在 **fsck.gfs2** 命令完成后，重新启用文件系统资源使其返回服务：

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

g. 创建一个红帽支持问题单。告知您遇到 GFS2 撤回，并提供 **sosreports** 和 **gfs2_edit savemeta** 命令生成的日志和调试信息。
在 GFS2 撤回的某些情况下，试图访问文件系统或其块设备的命令可能会挂起。在这些情况下，需要一个"硬"重启来重启集群。

有关 GFS2 撤回功能的详情，请参考 GFS2 文件系统对节点不可用(GFS2 撤回功能)。

- 这个出错信息表示可能有锁定问题或者程序漏洞。在出现这个问题时收集 GFS2 数据并创建一个红帽支持问题单，如收集 GFS2 数据进行故障排除所述。

7.4. GFS2 文件系统不会挂载到新添加的集群节点中

如果您在集群中添加新节点，且发现您无法在那个节点中挂载 GFS2 文件系统，则 GFS2 文件系统中的日志可能比尝试访问 GFS2 文件系统的节点要少。在您要在其上挂载文件系统的每个 GFS2 主机都要有一个日志（通过 **spectator** 挂载选项集挂载的 GFS2 文件系统除外，因为这不需要日志）。您可以使用 **gfs2_jadd** 命令向 GFS2 文件系统添加日志，如 向 GFS2 文件系统添加日志 中所述。

7.5. 在空文件系统中的空间被标记为已被使用

如果您有一个空的 GFS2 文件系统，**df** 命令将显示有空间被占用。这是因为 GFS2 文件系统日志消耗磁盘上的空间（日志数*日志大小）。如果您创建了一个具有大量日志的 GFS2 文件系统，或者指定较大的日志大小，那么在执行 **df** 命令时，您将看到（日志数*日志大小）已在使用。即使您没有设置大量日志或者大的日志，小 GFS2 文件系统（1GB 或者更小）也会显示部分空间被使用（默认 GFS2 日志大小）。

7.6. 为故障排除收集 GFS2 数据

如果您的 GFS2 文件系统挂起，且不会返回针对它运行的命令，您应该收集以下数据并创建一个红帽支持问题单：

- 每个节点上文件系统的 GFS2 锁转储数据：

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

- 每个节点上文件系统的 DLM 锁转储数据：您可以使用 **dlm_tool** 获取此信息：

```
dlm_tool lockdebug -sv lsname
```

在这个命令中，*lsname* 是有问题的文件系统中 DLM 使用的锁定空间名称。您可以在 **group_tool** 命令的输出中找到这个值。

- **sysrq -t** 命令的输出。
- **/var/log/messages** 文件的内容。

收集到这些数据后，创建一个红帽支持问题单。

第 8 章 集群中的 GFS2 文件系统

使用以下管理流程在红帽高可用性集群中配置 GFS2 文件系统。

8.1. 在集群中配置 GFS2 文件系统

您可以按照以下流程设置包含 GFS2 文件系统的 Pacemaker 集群。在这个示例中，您在双节点集群的三个逻辑卷上创建三个 GFS2 文件系统。

先决条件

- 在集群节点上安装并启动集群软件，并创建一个基本的双节点集群。
- 为集群配置隔离。

有关创建 Pacemaker 集群并为集群配置隔离的详情，请参考 使用 Pacemaker 创建红帽高可用性集群。

步骤

1. 在集群中的两个节点上，启用与您的系统架构对应的弹性存储存储库。例如，要为 x86_64 系统启用 Resilient Storage 仓库，您可以输入以下 **subscription-manager** 命令：

```
# subscription-manager repos --enable=rhel-8-for-x86_64-resilientstorage-rpms
```

请注意，弹性存储存储库是高可用性存储库的超集。如果启用弹性存储存储库，则不需要启用高可用性存储库。

2. 在集群的两个节点上安装 **lvm2-lockd**、**gfs2-utils** 和 **dlm** 软件包。要支持这些软件包，您必须订阅 AppStream 频道和 Resilient Storage 频道。

```
# yum install lvm2-lockd gfs2-utils dlm
```

3. 在集群的两个节点上，将 **/etc/lvm/lvm.conf** 文件中的 **use_lvmlockd** 配置选项设置为 **use_lvmlockd=1**。

```
...
use_lvmlockd = 1
...
```

4. 将全局 Pacemaker 参数 **no-quorum-policy** 设置为 **freeze**。

注意

默认情况下，将 **no-quorum-policy** 的值设置为 **stop**，表示一旦仲裁丢失，剩余分区上的所有资源都会立即停止。通常，这个默认行为是最安全、最优的选项，但与大多数资源不同，GFS2 要求使用 quorum 才可以正常工作。当使用 GFS2 挂载的应用程序和 GFS2 挂载都丢失时，就无法正确停止 GFS2 挂载。任何在没有 quorum 的情况下停止这些资源的尝试都会失败，并最终会在每次 quorum 都丢失时保护整个集群。

要解决这个问题，请在使用 GFS2 时将 **no-quorum-policy** 设置为 **freeze**。这意味着，当 quorum 丢失时，剩余的分区将不会进行任何操作，直到 quorum 功能被恢复。

```
[root@z1 ~]# pcs property set no-quorum-policy=freeze
```

5. 设置 **dlm** 资源。这是在集群中配置 GFS2 文件系统所需的依赖软件包。这个示例创建 **dlm** 资源作为名为 **locking** 的资源组的一部分。

```
[root@z1 ~]# pcs resource create dlm --group locking ocf:pacemaker:controld op
monitor interval=30s on-fail=fence
```

6. 克隆 **locking** 资源组，以便资源组可以在集群的两个节点上都活跃。

```
[root@z1 ~]# pcs resource clone locking interleave=true
```

7. 建立 **lvmlockd** 资源，来作为 **locking** 资源组的一部分。

```
[root@z1 ~]# pcs resource create lvmlockd --group locking ocf:heartbeat:lvmlockd op
monitor interval=30s on-fail=fence
```

8. 检查集群的状态，以确保在集群的两个节点上启动了 **locking** 资源组。

```
[root@z1 ~]# pcs status --full
Cluster name: my_cluster
[...]

Online: [ z1.example.com (1) z2.example.com (2) ]

Full list of resources:

 smoke-apc      (stonith:fence_apc):    Started z1.example.com
 Clone Set: locking-clone [locking]
     Resource Group: locking:0
         dlm    (ocf::pacemaker:controld):      Started z1.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z1.example.com
     Resource Group: locking:1
         dlm    (ocf::pacemaker:controld):      Started z2.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z2.example.com
     Started: [ z1.example.com z2.example.com ]
```

9. 在集群的一个节点中，创建两个共享卷组。一个卷组将包含两个 GFS2 文件系统，另一个卷组将包含一个 GFS2 文件系统。

注意

如果您的 LVM 卷组包含一个或多个位于远程块存储上的物理卷，如 iSCSI 目标，则红帽建议您确保服务在 Pacemaker 启动前启动。有关为 Pacemaker 集群使用的远程物理卷配置启动顺序的详情，请参考 为不由 Pacemaker 管理的资源依赖项配置启动顺序。

以下命令在 **/dev/vdb** 上创建共享卷组 **shared_vg1**。

```
[root@z1 ~]# vgcreate --shared shared_vg1 /dev/vdb
  Physical volume "/dev/vdb" successfully created.
  Volume group "shared_vg1" successfully created
```

```
  VG shared_vg1 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

以下命令在 **/dev/vdc** 上创建共享卷组 **shared_vg2**。

```
[root@z1 ~]# vgcreate --shared shared_vg2 /dev/vdc
  Physical volume "/dev/vdc" successfully created.
  Volume group "shared_vg2" successfully created
  VG shared_vg2 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

10. 在集群的第二个节点上：

    a. (RHEL 8.5 及更高版本)如果您通过在 **lvm.conf** 文件中设置 **use_devicesfile = 1** 来启用设备文件的使用，请将共享设备添加到设备文件中。默认情况下，不启用设备文件的使用。

    ```
    [root@z2 ~]# lvmdevices --adddev /dev/vdb
    [root@z2 ~]# lvmdevices --adddev /dev/vdc
    ```

    b. 为每个共享卷组启动锁管理器。

    ```
    [root@z2 ~]# vgchange --lockstart shared_vg1
      VG shared_vg1 starting dlm lockspace
      Starting locking.  Waiting until locks are ready...
    [root@z2 ~]# vgchange --lockstart shared_vg2
      VG shared_vg2 starting dlm lockspace
      Starting locking.  Waiting until locks are ready...
    ```

11. 在集群的一个节点中，创建共享逻辑卷并使用 GFS2 文件系统格式化卷。每个挂载文件系统的节点都需要一个日志。确保为集群中的每个节点创建足够日志。锁表名称的格式为 *ClusterName:FSName*，其中 *ClusterName* 是正在创建的 GFS2 文件系统的集群名称，*FSName* 是文件系统名称，它对集群中的所有 **lock_dlm** 文件系统必须是唯一的。

```
[root@z1 ~]# lvcreate --activate sy -L5G -n shared_lv1 shared_vg1
  Logical volume "shared_lv1" created.
[root@z1 ~]# lvcreate --activate sy -L5G -n shared_lv2 shared_vg1
  Logical volume "shared_lv2" created.
[root@z1 ~]# lvcreate --activate sy -L5G -n shared_lv1 shared_vg2
  Logical volume "shared_lv1" created.

[root@z1 ~]# mkfs.gfs2 -j2 -p lock_dlm -t my_cluster:gfs2-demo1
/dev/shared_vg1/shared_lv1
[root@z1 ~]# mkfs.gfs2 -j2 -p lock_dlm -t my_cluster:gfs2-demo2
/dev/shared_vg1/shared_lv2
[root@z1 ~]# mkfs.gfs2 -j2 -p lock_dlm -t my_cluster:gfs2-demo3
/dev/shared_vg2/shared_lv1
```

12. 为每个逻辑卷创建一个 **LVM-activate** 资源，以便在所有节点中自动激活该逻辑卷。

    a. 为卷组 **shared_vg1** 中的逻辑卷 **shared_lv1** 创建名为 **sharedlv1** 的 **LVM-activate** 资源。此命令还会创建包含该资源的资源组 **shared_vg1**。在这个示例中，资源组的名称与包含逻辑卷的共享卷组的名称相同。

```
[root@z1 ~]# pcs resource create sharedlv1 --group shared_vg1 ocf:heartbeat:LVM-activate lvname=shared_lv1 vgname=shared_vg1 activation_mode=shared vg_access_mode=lvmlockd
```

b. 为卷组 **shared_vg1** 中的逻辑卷 **shared_lv2** 创建名为 **sharedlv2** 的 **LVM-activate** 资源。此资源也是资源组 **shared_vg1** 的一部分。

```
[root@z1 ~]# pcs resource create sharedlv2 --group shared_vg1 ocf:heartbeat:LVM-activate lvname=shared_lv2 vgname=shared_vg1 activation_mode=shared vg_access_mode=lvmlockd
```

c. 为卷组 **shared_vg2** 中的逻辑卷 **shared_lv1** 创建名为 **sharedlv3** 的 **LVM-activate** 资源。此命令还会创建包含该资源的资源组 **shared_vg2**。

```
[root@z1 ~]# pcs resource create sharedlv3 --group shared_vg2 ocf:heartbeat:LVM-activate lvname=shared_lv1 vgname=shared_vg2 activation_mode=shared vg_access_mode=lvmlockd
```

13. 克隆两个新资源组。

```
[root@z1 ~]# pcs resource clone shared_vg1 interleave=true
[root@z1 ~]# pcs resource clone shared_vg2 interleave=true
```

14. 配置排序限制，以确保首先启动包含 **dlm** 和 **lvmlockd** 资源的 **locking** 资源组。

```
[root@z1 ~]# pcs constraint order start locking-clone then shared_vg1-clone
Adding locking-clone shared_vg1-clone (kind: Mandatory) (Options: first-action=start then-action=start)
[root@z1 ~]# pcs constraint order start locking-clone then shared_vg2-clone
Adding locking-clone shared_vg2-clone (kind: Mandatory) (Options: first-action=start then-action=start)
```

15. 配置共存限制，以确保 **vg1** 和 **vg2** 资源组在与 **locking** 资源组相同的节点上启动。

```
[root@z1 ~]# pcs constraint colocation add shared_vg1-clone with locking-clone
[root@z1 ~]# pcs constraint colocation add shared_vg2-clone with locking-clone
```

16. 在集群中的两个节点中，验证逻辑卷是否活跃。这可能会延迟几秒钟。

```
[root@z1 ~]# lvs
  LV        VG         Attr       LSize
  shared_lv1 shared_vg1  -wi-a----- 5.00g
  shared_lv2 shared_vg1  -wi-a----- 5.00g
  shared_lv1 shared_vg2  -wi-a----- 5.00g

[root@z2 ~]# lvs
  LV        VG         Attr       LSize
  shared_lv1 shared_vg1  -wi-a----- 5.00g
  shared_lv2 shared_vg1  -wi-a----- 5.00g
  shared_lv1 shared_vg2  -wi-a----- 5.00g
```

17. 创建文件系统资源在所有节点中自动挂载每个 GFS2 文件系统。

您不应该将文件系统添加到 **/etc/fstab** 文件，因为它将作为 Pacemaker 集群资源进行管理。可以通过 **options=*options*** 将挂载选项指定为资源配置的一部分。运行 **pcs resource describe Filesystem** 命令显示完整的配置选项。

以下命令可创建文件系统资源。这些命令在包含该文件系统逻辑卷资源的资源组中添加每个资源。

```
[root@z1 ~]# pcs resource create sharedfs1 --group shared_vg1
ocf:heartbeat:Filesystem device="/dev/shared_vg1/shared_lv1" directory="/mnt/gfs1"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence
[root@z1 ~]# pcs resource create sharedfs2 --group shared_vg1
ocf:heartbeat:Filesystem device="/dev/shared_vg1/shared_lv2" directory="/mnt/gfs2"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence
[root@z1 ~]# pcs resource create sharedfs3 --group shared_vg2
ocf:heartbeat:Filesystem device="/dev/shared_vg2/shared_lv1" directory="/mnt/gfs3"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence
```

验证步骤

1. 验证 GFS2 文件系统是否挂载到集群的两个节点中。

```
[root@z1 ~]# mount | grep gfs2
/dev/mapper/shared_vg1-shared_lv1 on /mnt/gfs1 type gfs2 (rw,noatime,seclabel)
/dev/mapper/shared_vg1-shared_lv2 on /mnt/gfs2 type gfs2 (rw,noatime,seclabel)
/dev/mapper/shared_vg2-shared_lv1 on /mnt/gfs3 type gfs2 (rw,noatime,seclabel)

[root@z2 ~]# mount | grep gfs2
/dev/mapper/shared_vg1-shared_lv1 on /mnt/gfs1 type gfs2 (rw,noatime,seclabel)
/dev/mapper/shared_vg1-shared_lv2 on /mnt/gfs2 type gfs2 (rw,noatime,seclabel)
/dev/mapper/shared_vg2-shared_lv1 on /mnt/gfs3 type gfs2 (rw,noatime,seclabel)
```

2. 检查集群的状态。

```
[root@z1 ~]# pcs status --full
Cluster name: my_cluster
[...]

Full list of resources:

 smoke-apc      (stonith:fence_apc):    Started z1.example.com
 Clone Set: locking-clone [locking]
     Resource Group: locking:0
         dlm    (ocf::pacemaker:controld):      Started z2.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z2.example.com
     Resource Group: locking:1
         dlm    (ocf::pacemaker:controld):      Started z1.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z1.example.com
     Started: [ z1.example.com z2.example.com ]
 Clone Set: shared_vg1-clone [shared_vg1]
     Resource Group: shared_vg1:0
         sharedlv1      (ocf::heartbeat:LVM-activate):  Started z2.example.com
         sharedlv2      (ocf::heartbeat:LVM-activate):  Started z2.example.com
         sharedfs1      (ocf::heartbeat:Filesystem):    Started z2.example.com
         sharedfs2      (ocf::heartbeat:Filesystem):    Started z2.example.com
     Resource Group: shared_vg1:1
```

```
        sharedlv1      (ocf::heartbeat:LVM-activate):  Started z1.example.com
        sharedlv2      (ocf::heartbeat:LVM-activate):  Started z1.example.com
        sharedfs1      (ocf::heartbeat:Filesystem):    Started z1.example.com
        sharedfs2      (ocf::heartbeat:Filesystem):    Started z1.example.com
    Started: [ z1.example.com z2.example.com ]
Clone Set: shared_vg2-clone [shared_vg2]
    Resource Group: shared_vg2:0
        sharedlv3      (ocf::heartbeat:LVM-activate):  Started z2.example.com
        sharedfs3      (ocf::heartbeat:Filesystem):    Started z2.example.com
    Resource Group: shared_vg2:1
        sharedlv3      (ocf::heartbeat:LVM-activate):  Started z1.example.com
        sharedfs3      (ocf::heartbeat:Filesystem):    Started z1.example.com
    Started: [ z1.example.com z2.example.com ]

...
```

其他资源

- 配置 GFS2 文件系统
- 在 Microsoft Azure 上配置红帽高可用性集群
- 在 AWS 上配置红帽高可用性集群
- 在 Google Cloud Platform 上配置红帽高可用性集群
- 在 Alibaba Cloud 上为 Red Hat High Availability 集群配置共享块存储

8.2. 在集群中配置加密的 GFS2 文件系统

(RHEL 8.4 及更高版本)您可以按照以下流程创建一个包含 LUKS 加密的 GFS2 文件系统的 Pacemaker 集群。在这个示例中，您在逻辑卷上创建一个 GFS2 文件系统，并加密文件系统。使用 **crypt** 资源代理支持加密 GFS2 文件系统，该代理支持 LUKS 加密。

此流程有三个部分：

- 在 Pacemaker 集群中配置共享逻辑卷
- 加密逻辑卷并创建一个 **crypt** 资源
- 使用 GFS2 文件系统格式化加密的逻辑卷并为集群创建文件系统资源

8.2.1. 在 Pacemaker 集群中配置共享逻辑卷

先决条件

- 在两个集群节点上安装并启动集群软件，并创建一个基本的双节点集群。
- 为集群配置隔离。

有关创建 Pacemaker 集群并为集群配置隔离的详情，请参考 使用 Pacemaker 创建红帽高可用性集群。

步骤

1. 在集群中的两个节点上，启用与您的系统架构对应的弹性存储存储库。例如，要为 x86_64 系统启用 Resilient Storage 仓库，您可以输入以下 **subscription-manager** 命令：

```
# subscription-manager repos --enable=rhel-8-for-x86_64-resilientstorage-rpms
```

请注意，弹性存储存储库是高可用性存储库的超集。如果启用弹性存储存储库，则不需要启用高可用性存储库。

2. 在集群的两个节点上安装 **lvm2-lockd**、**gfs2-utils** 和 **dlm** 软件包。要支持这些软件包，您必须订阅 AppStream 频道和 Resilient Storage 频道。

```
# yum install lvm2-lockd gfs2-utils dlm
```

3. 在集群的两个节点上，将 **/etc/lvm/lvm.conf** 文件中的 **use_lvmlockd** 配置选项设置为 **use_lvmlockd=1**。

```
...
use_lvmlockd = 1
...
```

4. 将全局 Pacemaker 参数 **no-quorum-policy** 设置为 **freeze**。

注意

默认情况下，**no-quorum-policy** 的值被设置为 **stop**，表示当仲裁丢失时，剩余分区上的所有资源将被立即停止。通常，这个默认行为是最安全、最优的选项，但与大多数资源不同，GFS2 要求使用 quorum 才可以正常工作。当使用 GFS2 挂载的应用程序和 GFS2 挂载都丢失时，就无法正确停止 GFS2 挂载。任何在没有 quorum 的情况下停止这些资源的尝试都会失败，并最终会在每次 quorum 都丢失时保护整个集群。

要解决这个问题，请在使用 GFS2 时将 **no-quorum-policy** 设置为 **freeze**。这意味着，当 quorum 丢失时，剩余的分区将不会进行任何操作，直到 quorum 功能被恢复。

```
[root@z1 ~]# pcs property set no-quorum-policy=freeze
```

5. 设置 **dlm** 资源。这是在集群中配置 GFS2 文件系统所需的依赖软件包。这个示例创建 **dlm** 资源作为名为 **locking** 的资源组的一部分。

```
[root@z1 ~]# pcs resource create dlm --group locking ocf:pacemaker:controld op
monitor interval=30s on-fail=fence
```

6. 克隆 **locking** 资源组，以便资源组可以在集群的两个节点上都活跃。

```
[root@z1 ~]# pcs resource clone locking interleave=true
```

7. 设置 **lvmlockd** 资源作为组 **locking** 的一部分。

```
[root@z1 ~]# pcs resource create lvmlockd --group locking ocf:heartbeat:lvmlockd op
monitor interval=30s on-fail=fence
```

8. 检查集群的状态，以确保在集群的两个节点上启动了 **locking** 资源组。

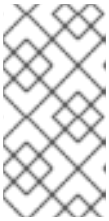
```
[root@z1 ~]# pcs status --full
Cluster name: my_cluster
[...]

Online: [ z1.example.com (1) z2.example.com (2) ]

Full list of resources:

 smoke-apc      (stonith:fence_apc):    Started z1.example.com
 Clone Set: locking-clone [locking]
     Resource Group: locking:0
         dlm    (ocf::pacemaker:controld):      Started z1.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z1.example.com
     Resource Group: locking:1
         dlm    (ocf::pacemaker:controld):      Started z2.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z2.example.com
     Started: [ z1.example.com z2.example.com ]
```

9. 在集群的一个节点中创建一个共享卷组。



注意

如果您的 LVM 卷组包含一个或多个位于远程块存储上的物理卷，如 iSCSI 目标，则红帽建议您确保服务在 Pacemaker 启动前启动。有关为 Pacemaker 集群使用的远程物理卷配置启动顺序的详情，请参考 为不由 Pacemaker 管理的资源依赖项配置启动顺序。

以下命令在 **/dev/sda1** 上创建共享卷组 **shared_vg1**。

```
[root@z1 ~]# vgcreate --shared shared_vg1 /dev/sda1
  Physical volume "/dev/sda1" successfully created.
  Volume group "shared_vg1" successfully created
  VG shared_vg1 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

10. 在集群的第二个节点上：

    a. (RHEL 8.5 及更高版本)如果您通过在 **lvm.conf** 文件中设置 **use_devicesfile = 1** 来启用设备文件的使用，请将共享设备添加到集群中第二个节点上的设备文件中。默认情况下，不启用设备文件的使用。

    ```
    [root@z2 ~]# lvmdevices --adddev /dev/sda1
    ```

    b. 为共享卷组启动锁管理器。

    ```
    [root@z2 ~]# vgchange --lockstart shared_vg1
      VG shared_vg1 starting dlm lockspace
      Starting locking.  Waiting until locks are ready...
    ```

11. 在集群的一个节点中创建共享逻辑卷。

```
[root@z1 ~]# lvcreate --activate sy -L5G -n shared_lv1 shared_vg1
  Logical volume "shared_lv1" created.
```

12. 为逻辑卷创建一个 **LVM-activate** 资源，以便在所有节点中自动激活逻辑卷。
以下命令为卷组 **shared_vg1** 中的逻辑卷 **shared_lv1** 创建名为 **sharedlv1** 的 **LVM-activate** 资源。此命令还会创建包含该资源的资源组 **shared_vg1**。在这个示例中，资源组的名称与包含逻辑卷的共享卷组的名称相同。

```
[root@z1 ~]# pcs resource create sharedlv1 --group shared_vg1 ocf:heartbeat:LVM-activate lvname=shared_lv1 vgname=shared_vg1 activation_mode=shared vg_access_mode=lvmlockd
```

13. 克隆新资源组。

```
[root@z1 ~]# pcs resource clone shared_vg1 interleave=true
```

14. 配置排序限制，以确保首先启动包含 **dlm** 和 **lvmlockd** 资源的 **locking** 资源组。

```
[root@z1 ~]# pcs constraint order start locking-clone then shared_vg1-clone
Adding locking-clone shared_vg1-clone (kind: Mandatory) (Options: first-action=start then-action=start)
```

15. 配置 colocation 约束，以确保 **vg1** 和 **vg2** 资源组在与 **locking** 资源组相同的节点上启动。

```
[root@z1 ~]# pcs constraint colocation add shared_vg1-clone with locking-clone
```

验证步骤

在集群中的两个节点中，验证逻辑卷是否活跃。这可能会延迟几秒钟。

```
[root@z1 ~]# lvs
  LV        VG         Attr       LSize
  shared_lv1 shared_vg1  -wi-a----- 5.00g

[root@z2 ~]# lvs
  LV        VG         Attr       LSize
  shared_lv1 shared_vg1  -wi-a----- 5.00g
```

8.2.2. 加密逻辑卷并创建 crypt 资源

先决条件

- 您已在 Pacemaker 集群中配置了共享逻辑卷。

步骤

1. 在集群的一个节点中，创建一个包含 crypt 密钥的新文件，并在文件中设置权限，使其只能被 root 读取。

```
[root@z1 ~]# touch /etc/crypt_keyfile
[root@z1 ~]# chmod 600 /etc/crypt_keyfile
```

2. 创建 crypt 密钥。

```
[root@z1 ~]# dd if=/dev/urandom bs=4K count=1 of=/etc/crypt_keyfile
```

```
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000306202 s, 13.4 MB/s
[root@z1 ~]# scp /etc/crypt_keyfile root@z2.example.com:/etc/
```

3. 使用 **-p** 参数将 crypt keyfile 分发到集群中的其他节点，以保留您设置的权限。

```
[root@z1 ~]# scp -p /etc/crypt_keyfile root@z2.example.com:/etc/
```

4. 在要配置加密的 GFS2 文件系统的 LVM 卷中创建加密设备。

```
[root@z1 ~]# cryptsetup luksFormat /dev/shared_vg1/shared_lv1 --type luks2 --key-
file=/etc/crypt_keyfile
WARNING!
========
This will overwrite data on /dev/shared_vg1/shared_lv1 irrevocably.

Are you sure? (Type 'yes' in capital letters): YES
```

5. 创建 crypt 资源，作为 **shared_vg1** 卷组的一部分。

```
[root@z1 ~]# pcs resource create crypt --group shared_vg1 ocf:heartbeat:crypt
crypt_dev="luks_lv1" crypt_type=luks2 key_file=/etc/crypt_keyfile
encrypted_dev="/dev/shared_vg1/shared_lv1"
```

验证步骤

确保 crypt 资源已创建了 crypt 设备，本例中为 **/dev/mapper/luks_lv1**。

```
[root@z1 ~]# ls -l /dev/mapper/
...
lrwxrwxrwx 1 root root 7 Mar 4 09:52 luks_lv1 -> ../dm-3
...
```

8.2.3. 使用 GFS2 文件系统格式化加密的逻辑卷，并为集群创建一个文件系统资源

先决条件

- 您已加密逻辑卷并创建 crypt 资源。

步骤

1. 在集群的一个节点中，使用 GFS2 文件系统格式化卷。每个挂载文件系统的节点都需要一个日志。确保为集群中的每个节点创建足够日志。锁表名称的格式为 *ClusterName:FSName*，其中 *ClusterName* 是正在创建的 GFS2 文件系统的集群名称，*FSName* 是文件系统名称，它对集群中的所有 **lock_dlm** 文件系统必须是唯一的。

```
[root@z1 ~]# mkfs.gfs2 -j3 -p lock_dlm -t my_cluster:gfs2-demo1 /dev/mapper/luks_lv1
/dev/mapper/luks_lv1 is a symbolic link to /dev/dm-3
This will destroy any data on /dev/dm-3
Are you sure you want to proceed? [y/n] y
Discarding device contents (may take a while on large devices): Done
Adding journals: Done
```

```
Building resource groups: Done
Creating quota file: Done
Writing superblock and syncing: Done
Device:                    /dev/mapper/luks_lv1
Block size:                4096
Device size:               4.98 GB (1306624 blocks)
Filesystem size:           4.98 GB (1306622 blocks)
Journals:                  3
Journal size:              16MB
Resource groups:           23
Locking protocol:          "lock_dlm"
Lock table:                "my_cluster:gfs2-demo1"
UUID:                      de263f7b-0f12-4d02-bbb2-56642fade293
```

2. 创建文件系统资源在所有节点中自动挂载 GFS2 文件系统。
不要将该文件系统添加到 **/etc/fstab** 文件中，因为它将作为 Pacemaker 集群资源进行管理。可以通过 **options=*options*** 将挂载选项指定为资源配置的一部分。运行 **pcs resource describe Filesystem** 命令以了解完整的配置选项。

   以下命令创建文件系统资源。这个命令在包含该文件系统逻辑卷资源的资源组中添加资源。

```
[root@z1 ~]# pcs resource create sharedfs1 --group shared_vg1
ocf:heartbeat:Filesystem device="/dev/mapper/luks_lv1" directory="/mnt/gfs1"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence
```

验证步骤

1. 验证 GFS2 文件系统是否挂载到集群的两个节点上。

```
[root@z1 ~]# mount | grep gfs2
/dev/mapper/luks_lv1 on /mnt/gfs1 type gfs2 (rw,noatime,seclabel)

[root@z2 ~]# mount | grep gfs2
/dev/mapper/luks_lv1 on /mnt/gfs1 type gfs2 (rw,noatime,seclabel)
```

2. 检查集群的状态。

```
[root@z1 ~]# pcs status --full
Cluster name: my_cluster
[...]

Full list of resources:

 smoke-apc      (stonith:fence_apc):    Started z1.example.com
 Clone Set: locking-clone [locking]
     Resource Group: locking:0
         dlm    (ocf::pacemaker:controld):      Started z2.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z2.example.com
     Resource Group: locking:1
         dlm    (ocf::pacemaker:controld):      Started z1.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z1.example.com
   Started: [ z1.example.com z2.example.com ]
 Clone Set: shared_vg1-clone [shared_vg1]
   Resource Group: shared_vg1:0
```

```
        sharedlv1      (ocf::heartbeat:LVM-activate):  Started z2.example.com
        crypt      (ocf::heartbeat:crypt) Started z2.example.com
        sharedfs1      (ocf::heartbeat:Filesystem):    Started z2.example.com
    Resource Group: shared_vg1:1
        sharedlv1      (ocf::heartbeat:LVM-activate):  Started z1.example.com
        crypt      (ocf::heartbeat:crypt)  Started z1.example.com
        sharedfs1      (ocf::heartbeat:Filesystem):    Started z1.example.com
      Started:  [z1.example.com z2.example.com ]
...
```

其他资源

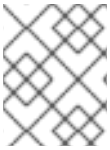
- 配置 GFS2 文件系统

8.3. 将 GFS2 文件系统从 RHEL7 迁移到 RHEL8

当配置包含 GFS2 文件系统的 RHEL 8 集群时，您可以使用现有的 Red Hat Enterprise 7 逻辑卷。

在 Red Hat Enterprise Linux 8 中，LVM 使用 LVM 锁守护进程 **lvmlockd** ，而不是 **clvmd** 在管理主动/主动集群中的共享存储设备。这要求您配置作为共享逻辑卷使用的主动/主动集群所需的逻辑卷。另外，这需要您使用 **LVM-activate** 资源来管理 LVM 卷，并使用 **lvmlockd** 资源代理来管理 **lvmlockd** 守护进程。有关配置使用带有共享逻辑卷的 GFS2 文件系统的 Pacemaker 集群的信息，请参阅在集群中配置 GFS2 文件系统。

要在配置包含 GFS2 文件系统的 RHEL8 集群时使用现有的 Red Hat Enterprise Linux 7 逻辑卷，请从 RHEL8 集群中执行以下步骤。在这个示例中，集群的 RHEL 7 逻辑卷是卷组 **upgrade_gfs_vg** 的一部分。



注意

RHEL8 集群的名称必须与 RHEL7 集群的名称相同，其中包括 GFS2 文件系统才能使现有文件系统有效。

步骤

1. 确定包含 GFS2 文件系统的逻辑卷当前不活跃。只有所有节点都停止使用卷组时，这个步骤才安全。

2. 从集群中的一个节点中，强制将卷组更改为本地。

```
[root@rhel8-01 ~]# vgchange --lock-type none --lock-opt force upgrade_gfs_vg
Forcibly change VG lock type to none? [y/n]: y
  Volume group "upgrade_gfs_vg" successfully changed
```

3. 从集群中的一个节点，将本地卷组改为共享卷组

```
[root@rhel8-01 ~]# vgchange --lock-type dlm upgrade_gfs_vg
   Volume group "upgrade_gfs_vg" successfully changed
```

4. 在集群的每个节点中，开始锁定卷组。

```
[root@rhel8-01 ~]# vgchange --lockstart upgrade_gfs_vg
  VG upgrade_gfs_vg starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

```
[root@rhel8-02 ~]# vgchange --lockstart upgrade_gfs_vg
  VG upgrade_gfs_vg starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

执行此流程后，您可以为每个逻辑卷创建一个 **LVM-activate** 资源。

第 9 章 GFS2 追踪点和 GLOCK DEBUGFS 接口

关于 GFS2 追踪点和 glock **debugfs** 接口本文档面向熟悉文件系统内部的高级用户和想了解更多有关 GFS2 设计以及如何调试特定于 GFS2 问题的用户。

以下章节描述了 GFS2 追踪点和 GFS2 **glocks** 文件。

9.1. GFS2 追踪点（TRACEPOINT）类型

目前有三种类型的 GFS2 追踪点：*glock*（发音为 "gee-lock"）追踪点、*bmap* 追踪点和 *log* 追踪点。它们可以用来监控正在运行的 GFS2 文件系统。当问题（如挂起或性能问题）可以重复出现时，追踪点就会非常有用，因此可以在有问题的操作过程中获得追踪点的输出。在 GFS2 中，glocks 是主要缓存控制机制，它们是了解 GFS2 内核性能的关键。bmap（块映射）追踪点可用于监控发生的块分配和块映射（查找磁盘元数据树中已分配的块），以及检查与访问位置相关的任何问题。日志追踪点跟踪写入日志和从日志释放的数据，并提供了有关 GFS2 的那部分的有用信息。

追踪点的设计原则是尽可能通用。这意味着在 Red Hat Enterprise Linux 8 中不需要更改 API。另外，使用这个接口的用户应该意识到，它只是一个调试界面，而不是一般的 Red Hat Enterprise Linux 8 API 集的一部分，因此红帽并不保证不会在以后修改 GFS2 追踪点接口。

追踪点是 Red Hat Enterprise Linux 的一般功能，其范围超出了 GFS2。特别是它们用于实现 **blktrace** 基础架构，并且可将 **blktrace** 追踪点可与 GFS2 结合使用，以获得系统性能的全面状况。取决于追踪点操作的级别，它们可能会在短时间内产生大量数据。虽然追踪点被设计为在启用时有最小的系统负载，但它们无可避免地会产生一些影响。通过不同方法过滤事件可帮助减少数据的数量，并帮助只获得有助于了解特定情形的信息。

9.2. 追踪点（TRACEPOINTS）

追踪点可以在 **/sys/kernel/debug/** 目录下找到，假设 **debugfs** 挂载在标准位置 **/sys/kernel/debug** 目录。**events** 子目录包含所有可以被指定的追踪事件，如果载入了 **gfs2** 模块，则将有一个包含其他子目录的 **gfs2** 子目录，每个 GFS2 事件一个。**/sys/kernel/debug/tracing/events/gfs2** 目录的内容类似如下：

```
[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
```

要启用所有 GFS2 追踪点，请输入以下命令：

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

要启用特定的追踪点，在每个事件子目录中都有一个 **enable** 文件。**filter** 文件也是如此，其可用于为每个事件或一组事件设置事件过滤器。下面详细解释了各个事件的含义。

追踪点的输出以 ASCII 或二进制格式提供。本附录目前不包含二进制接口。ASCII 接口有两种方式。要列出环缓冲（ring buffer）的当前内容，您可以输入以下命令：

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

当您在一定时间段内使用长时间运行的过程时，且希望在这些事件进行之后重新查看缓冲区中最新捕获的信息时，这个接口很有用。需要所有输出时，可以使用另一个接口 **/sys/kernel/debug/tracing/trace_pipe**。事件发生时会从这个文件中进行读取，这个接口不提供历史数据。两个接口的输出格式是相同，在本附录后续部分对每个 GFS2 事件都有说明。

可以使用名为 **trace-cmd** 的工具来读取追踪点数据。有关这个工具的详情，请参考 http://lwn.net/Articles/341902/。**trace-cmd** 工具与 **strace** 工具的使用方式类似，例如在从各种源收集追踪数据时运行命令。

9.3. GLOCKS

对于 GFS2，最重要的一个独特的概念就是 glocks。就源代码而言，glock 是一个数据结构，它将合并 DLM 并缓存到单个状态机器。每个 glock 与一个 DLM 锁定有一个 1:1 的关系，为该锁定状态提供缓存，以便从文件系统中单一节点执行的重复操作不必重复调用 DLM，这有助于避免不必要的网络流量。有两大类 glock，一类是有元数据缓存的 glock，另外是没有元数据缓存的 glock。内节点 glock 和资源组 glock 都有元数据缓存，其他类型的 glock 不缓存元数据。内节点 glock 除元数据外还会缓存数据，并且具有所有 glocks 中最复杂的逻辑。

表 9.1. Glock 模式和 DLM 锁定模式

Glock 模式	DLM 锁定模式	备注
UN	IV/NL	未锁定（没有与 glock 相关的 DLM 锁定，或 NL 锁定取决于 I 标记）
SH	PR	共享（读保护）锁定
EX	EX	专用锁定
DF	CW	用于直接 I/O 和文件系统停止的延迟（并性写）

Glocks 会一直保留在内存中，直到其被解锁（根据另一个节点的请求，或虚拟机的请求）或不再有本地用户。此时它们会从 glock 哈希表中移除并释放。当 glock 创建时，DLM 锁定不会立即与 glock 关联。当第一次请求 DLM 时，DLM 锁与 glock 关联，如果这个请求成功，那么会在 glock 上设置 'I'（初始）标记。glock debugfs 接口 中的"Glock Flags" 表显示不同的 glock 标记的含义。当 DLM 与 glock 关联后，DLM 锁定将始终保持至少为 NL（Null）锁定模式，直到 glock 被释放。当 DLM 锁从 NL 变为解锁状态始终是 glock 生命周期中的最后一个操作。

每个 glock 都有多个与它关联的"拥有者（holder）"，每个都代表来自更高层的一个锁定请求。与 GFS2 队列以及从 glock 中退出队列的持有者相关的系统调用来保护代码的关键部分。

glock 状态机器基于一个工作队列。出于性能的原因，tasklet 将是首选；但是，在当前的实现中，我们需要从那些禁止他们使用的上下文中提交 I/O。

注意

Workqueue 有自己的追踪点，它们可与 GFS2 追踪点结合使用。

下表显示了在每个 glock 模式下可能会缓存哪些状态，以及缓存的状态是否为脏。这适用于内节点和资源组锁定，尽管资源组锁中没有数据组件，只有元数据。

表 9.2. Glock 模式和数据类型

Glock 模式	缓存数据	缓存元数据	脏数据	脏元数据
UN	否	否	否	否
SH	是	是	否	否
DF	否	是	否	否
EX	是	是	是	是

9.4. GLOCK DEBUGFS 接口

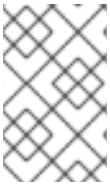
glock **debugfs** 接口允许 glocks 的内部状态和持有者的视觉化，并包括了在一些情况下被锁定的对象的一些概述详情。文件的每一行要么以 G: 开头，无缩进（指向 glock 本身），要么以不同的字母开头，以一个空格缩进，并指向文件中与其正上方的 glock 关联的结构(H: 是拥有者、I: inode ， R: 一个资源组)。下面是一个示例：

```
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]
```

上面的例子是在单个节点 GFS2 文件系统上运行 postmark 基准过程中，命令 **cat /sys/kernel/debug/gfs2/unity:myfs/glocks >my.locks >my.locks >my.lock** 所产生的一系列摘录（大约 18MB 文件）图中的 glocks 已选择，以便显示一些更值得关注的 glock 转储特性。

glock 状态是 EX（专用）、DF（推迟）、SH（共享）或 UN（未锁定）。这些状态与 DLM 锁定模式直接对应，但 UN 可能代表 DLM null 锁定状态，或者 GFS2 没有包含 DLM 锁定（取决于上面解释的 I 标记）。glock 的 s: 字段显示锁定的当前状态，拥有者中的同一字段指示请求的模式。如果准许了锁定，则拥有者将在标记（f: field）中带有 H 位。否则，它将设置 W wait 位。

n: 字段（数字）表示与每个项目关联的数字。对于 glocks，这是类型号，后接 glock 号，因此，在上例中第一个 glock 是 n:5/75320;，它表示一个 **iopen** glock，它与 inode 75320 相关。对于 inode 和 **iopen** glocks，glock 号始终与内节点的磁盘块号相同。



注意

debugfs glocks 文件中的 glock 号(n: field)以十六进制表示，而追踪点输出以十进制表示。这是因为历史原因；glock 号总是写成十六进制，但追踪点会选择十进制，因此号可以轻松地与其他追踪点输出（例如，来自 **blktrace** ）和来自 **stat**(1)的输出进行比较。

持有者和 glock 的所有标记的完整列表都在以下 "Glock Flags" 表和 Glock 持有者 中的 "Glock Holder Flags"表中设置。目前无法通过 glock **debugfs** 接口获取锁值块的内容。下表显示了不同 glock 类型的含义。

表 9.3. Glock 类型

类型号	锁定类型	使用
1	trans	事务锁定
2	inode	内节点元数据和数据
3	rgrp	源组群元数据
4	meta	超级块
5	iopen	内节点最近探测
6	flock	**flock**(2)syscall
8	quota	配额操作
9	journal	Journal mutex

更重要的 glock 标记之一就是 l (locked) 标记。这是执行状态更改时用于识别对 glock 状态的访问的位锁定。当状态机器要通过 DLM 发送远程锁定请求时设置它，只有在执行完操作时才会清除它。有时这意味着已发送了多个锁定请求，不同时间发生各种无效请求。

下表显示了不同的 glock 标记的含义。

表 9.4. Glock 标记

标记	名称	含义
d	Pending demote	推迟（远程）降级请求
D	Demote	降级请求（本地或者远程）
f	Log flush	释放这个 glock 前需要提交该日志
F	Frozen	回复忽略的远程节点 -- 恢复正在进行。

标记	名称	含义
i	Invalidate in progress	这个 glock 下无效页面的进程中
I	Initial	设定何时将 DLM 锁定与这个 glock 关联
l	Locked	这个 glock 处于更改状态中
L	LRU	设置什么时候 glock 出现在 LRU 列表中
o	Object	设置 glock 何时与对象关联（即类型为 2 glocks 的内节点，以及类型为 3 的 glock 的资源组）
p	Demote in progress	glock 正在响应降级请求
q	Queued	设定当拥有者排队到 glock 时，并在 glock 保留时清除，但没有剩余所有者。作为算法的一部分使用，计算 glock 的最小保留时间。
r	Reply pending	从远程节点接收的回复正在等待过程中
y	Dirty	释放这个 glock 前需要冲刷到磁盘中的数据

当从节点收到远程回调时（该节点希望在与本地节点上持有的锁冲突的模式下获得锁），则会设置两个标志 D（降级）或 d（降级待处理）中的一个。为了防止在特定锁竞争时出现饥饿的情况，可以为每个锁定分配一个最少持有时间。允许还没有获得锁的最小持有时间的节点保留该锁，直到时间间隔到期。

如果时间间隔已过期，那么将设置 D（demote）标志，并记录所需状态。在这种情况下，在拥有者队列中没有允许的锁定，锁定将被降级。如果时间间隔没有过期，则将设置 d（降级待处理）标记。这也会调度状态机器清除 d（降级待处理）并在最小的静默时间已过期时设置 D（降级）。

当为 glock 分配 DLM 锁定时，会设定 I（初始）标记。当 glock 首次被使用，而且 I 标签将会一直被设置，直到 glock 最终被释放（DLM 锁定被解锁）。

9.5. GLOCK 拥有者（HOLDER）

下表显示了不同的 glock 持有者标记的含义。

表 9.5. Glock 持有者标记

标记	名称	含义
a	Async	不要等待 glock 结果（以后轮询结果）

标记	名称	含义
A	Any	接受任意兼容锁定模式
c	No cache	取消锁定时，立即降级 DLM 锁定
e	No expire	忽略随后的锁定取消请求
E	Exact	必须有准确的锁定模式
F	First	设定赋予这个锁定的第一个拥有者
H	Holder	表示赋予请求的锁定
p	Priority	在队列头启用 ENQUEUE 拥有者
t	Try	"try" 锁定
T	Try 1CB	发送回调的 "try" 锁定
W	Wait	等待请求完成的设置

如前面提到的，最重要的拥有者标志是 H（拥有者）和 W（等待），因为它们分别被设置在赋予的锁定请求和锁定请求中。在队列中的拥有者的顺序非常重要。如果有被允许的所有者，则他们将总位于队列的前头，后接其他进入队列的所有者。

如果没有被允许的所有者，列表中的第一个拥有者就是触发下一个状态更改的拥有者。由于总是认为降级请求比文件系统的请求优先级高，因此可能并不总直接导致对请求的状态的更改。

glock 子系统支持两种类型的 "try" 锁定。这两类都比较有用，因为它们允许把锁定移出正常的顺序（使用适当的 back-off 和 retry），且它们可以被用来帮助避免资源被其他节点使用。正常的 t(try) 锁定如其名字所示，它是一个 "try"锁定，不会做任何特殊操作。另一方面，T（**try 1CB**）锁与 t 锁相同，除了 DLM 会向当前不兼容的锁持有者发送一个回调。T（**try 1CB**）锁的一个用处是带有 **iopen** 锁，其用于在 inode 的 **i_nlink** 计数为零时确定哪个节点负责释放 inode 。**iopen** glock 通常处于共享状态，但当 **i_nlink** 计数变为零且 →**evict_inode**（）被调用时，它将请求一个设置了 T (**try 1CB**) 的独占锁。如果允许了锁定，它将继续取消内节点的配置。如果锁没有授权，则会导致阻止锁授权的节点使用 D(降级)标记标记其 glock，该标记会在 →**drop_inode**（）时检查，以确保释放不会被忘记。

这意味着，具有零链接数但仍然处于打开状态的 inode 将由在其上发生最终 **close**() 的节点释放。另外，当内节点的链接计数降为 0 时，内节点被标记为处于具有零链路计数的特殊状态，但仍在资源组位映射中使用。此功能和 ext3 文件系统的孤儿列表类似，它允许位图的任何后续读取器知道有潜在的可能回收的空间，并尝试回收它。

9.6. GLOCK 追踪点

追踪点也被设计为能够通过将其与 **blktrace** 输出和磁盘布局知识相结合来确认缓存控制的正确性。然后可以检查是否在正确锁定下发布并完成任意给定的 I/O，并且没有竞争。

gfs2_glock_state_change 追踪点是要理解的最重要的一个。它跟踪 glock 的每次状态变化，从初始创建到最终降级，其以 **gfs2_glock_put** 和最终 NL 到解锁的转换结束。l (locked) glock 标志总是在状态更改发生前设置，且只有在完成更改后才会清除。在状态更改过程中，不会有未允许的拥有者（H glock 拥

有者标志）。如果有排队的拥有者，他们将总是处于 W（等待）状态。当完成了状态更改后，允许拥有者可能是清除 I glock 标签前的最后操作。

gfs2_demote_rq 追踪点会跟踪降级请求，包括本地和远程的。假设节点上有足够的内存，则很少看见本地降级请求，大多数情况下，它们通常由 **umount** 或偶尔的内存回收创建。远程降级请求数量是节点针对特定内节点或资源组间争用的指标。

gfs2_glock_lock_time 追踪点提供有关请求 DLM 所需时间的信息。阻止(**b**)标记被引入到 glock 中，特别是用于与这个追踪点结合使用。

当为持有者被授予锁时，会调用 **gfs2_promote**，这发生在状态变化的最后阶段，或者在请求锁时，由于 glock 状态已经缓存了合适模式的锁，因此可以立即授予该锁。如果拥有者是授予这个 glock 的第一个拥有者，那么就会在该拥有者中设定 f（第一个）标记。目前，这只由资源组使用。

9.7. BMAP 追踪点

块映射对任何文件系统都是一个核心任务。GFS2 使用传统的基于位图的系统，每个块有两个位。这个子系统中的追踪点的主要目的是，允许监控分配和映射块的时间。

对于每个 bmap 操作，**gfs2_bmap** 追踪点都会被调用两次：一次在开始显示 bmap 请求时，一次在结尾显示结果时。这样可轻松地将请求和结果匹配，并测量在文件系统的不同部分映射块、不同文件偏移甚至不同文件所需时间。也可以查看返回的平均扩展大小与请求的比较。

gfs2_rs 跟踪点跟踪块在块分配器中创建、使用和销毁时的块保留情况。

为了跟踪已分配的块，不仅会对分配，也会对块的释放调用 **GFS2_block_alloc** 。由于分配都是根据块所针对的 inode 的引用，因此这可用来跟踪哪些物理块属于活动文件系统中哪些文件。这在与 **blktrace** 相结合时特别有用，它将显示有问题的 I/O 模式，然后使用此追踪点所获取的映射将其引用回相关的 inode。

直接 I/O(**iomap**)是一个替代缓存策略，它允许在磁盘和用户缓冲区之间直接进行文件数据传输。当缓存的匹配率低时，这将非常有用。**gfs2_iomap_start** 和 **gfs2_iomap_end** 追踪点会跟踪这些操作，并可用于跟踪使用直接 I/O 的映射，直接 I/O 文件系统上的位置，以及操作类型。

9.8. 日志追踪点

该子系统中的追踪点跟踪添加到日志或从日志中删除的块(**gfs2_pin**)，以及将事务提交给日志所需的时间(**gfs2_log_flush**)。这在调试日志性能问题时非常有用。

gfs2_log_blocks 追踪点会跟踪日志中保留的块，例如，这可以帮助显示日志对于工作负载是否太小：

gfs2_ail_flush 追踪点与 **gfs2_log_flush** 追踪点相似，它跟踪 AIL 列表刷新的开始和结束。AIL 列表包含已通过日志，但尚未写回的缓冲区，这些缓冲区会定期刷新，以释放更多日志空间供文件系统使用，或者在进程请求 **sync** 或 **fsync** 时释放更多日志空间。

9.9. GLOCK 统计

GFS2 维护可帮助您跟踪文件系统中正在运行的统计信息。这可让您发现性能问题。

GFS2 维护两个计数器：

- **dcount** ，其统计请求的 DLM 操作的数量。这显示了有多少数据被用于均值/方差计算。
- **qcount** ，其统计请求的 **syscall** 级别操作的数量。通常 **qcount** 等于或大于 **dcount**。

另外，GFS2 维护三个均值/方差对。均值/方差对是平滑度估算，使用的算法是用于计算网络代码中往返的时间。

GFS2 中维护的均值和方差对不能扩展，而是以整数纳秒为单位。

- srtt/srttvar:非阻止操作的平稳往返时间
- srttb/srttvarb:阻止操作的平稳往返时间
- irtt/irttvar:请求间的时间（例如，DLM 请求之间的时间）

非阻塞请求是一个马上完成的请求，无论有疑问的 DLM 锁定状态如何。这当前意味着，请求的条件为 (a) 当前的锁定状态为专用（ exclusive）；(b) 请求的状态为 null 或者非锁定（unlocked）；或 (c) 设定了"try lock"标记。其他锁定请求都为阻塞请求。

对于 IRTTs 来说，较大的时间比较好；对于 RTTs 来说，比较小的时间更好。

统计数据保存在两个 **sysfs** 文件中：

- **glstats** 文件。这个文件与 **glocks** 文件类似，但它包含统计信息，一行一个 glock。数据是从那个为其创建 glock 的 glock类型 的"per cpu"数据初始化的，数据会被创建 glock（归零的计数器除外）。这个文件可能非常大。
- **lkstats** 文件。它为每个 glock 类型包含 "per cpu" 统计信息。它每行包含一个统计数据，每列都是一个 cpu 内核。每个 glock 类型有 8 行，一个类型跟随另一个类型。

9.10. 参考信息

有关追踪点和 GFS2 **glocks** 文件的详情，请查看以下资源：

- 有关 glock 内部锁定规则的详情，请参考 https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystem glocks.rst。
- 有关事件追踪的详情，请参考 https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/trace/eve
- 有关 **trace-cmd** 工具的详情，请参考 http://lwn.net/Articles/341902/。

第 10 章 使用 PERFORMANCE CO-PILOT(PCP)监控和分析 GFS2 文件系统

Performance Co-Pilot(PCP)可帮助监控和分析 GFS2 文件系统。PCP 中 GFS2 文件系统的监控是由 Red Hat Enterprise Linux 中的 GFS2 PMDA 模块提供的，该模块可通过 **pcp-pmda-gfs2** 软件包获得。

GFS2 PMDA 提供了很多由 **debugfs** 子系统中提供的 GFS2 统计给出的指标。安装后，PMDA 会公开 **glocks**、**glstats** 和 **sbstats** 文件中给出的值。报告每个挂载的 GFS2 文件系统的统计组。PMDA 还使用 Kernel Function Tracer(**ftrace**)公开的 GFS2 内核追踪点。

10.1. 安装 GFS2 PMDA

为了正确操作，GFS2 PMDA 要求 **debugfs** 文件系统已挂载。如果 **debugfs** 文件系统没有挂载，请在安装 GFS2 PMDA 前运行以下命令：

```
# mkdir /sys/kernel/debug
# mount -t debugfs none /sys/kernel/debug
```

在默认安装中不启用 GFS2 PMDA。要通过 PCP 使用 GFS2 指标监控，您必须在安装后启用它。

运行以下命令来安装 PCP 并启用 GFS2 PMDA。请注意，PMDA 安装脚本必须以 root 用户身份运行。

```
# yum install pcp pcp-pmda-gfs2
# cd /var/lib/pcp/pmdas/gfs2
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check gfs2 metrics have appeared ... 346 metrics and 255 values
```

10.2. 使用 PMINFO 工具显示可用性能指标的信息

pminfo 工具显示有关可用的性能指标的信息。以下示例显示了您可以使用这个工具显示的不同的 GFS2 指标。

10.2.1. 检查每个文件系统目前存在的 glock 结构数

GFS2 glock 指标可让您了解当前挂载的 GFS2 文件系统的 glock 结构数及其锁定状态。在 GFS2 中，glock 是一个数据结构，它将合并 DLM 并缓存到单个状态机器。每个 glock 都有一个与单个 DLM 锁的 1:1 映射，为锁定状态提供缓存，使得在单个节点中进行的重复操作不必重复调用 DLM，从而减少不必要的网络流量。

以下 **pminfo** 命令按其锁模式显示每个挂载的 GFS2 文件系统的 glocks 数的列表。

```
# pminfo -f gfs2.glocks

gfs2.glocks.total
    inst [0 or "afc_cluster:data"] value 43680
    inst [1 or "afc_cluster:bin"] value 2091

gfs2.glocks.shared
    inst [0 or "afc_cluster:data"] value 25
    inst [1 or "afc_cluster:bin"] value 25
```

```
gfs2.glocks.unlocked
    inst [0 or "afc_cluster:data"] value 43652
    inst [1 or "afc_cluster:bin"] value 2063

gfs2.glocks.deferred
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0

gfs2.glocks.exclusive
    inst [0 or "afc_cluster:data"] value 3
    inst [1 or "afc_cluster:bin"] value 3
```

10.2.2. 根据类型检查每个文件系统目前存在的 glock 结构数

GFS2 glstats 指标给出了每种 glock 类型的计数，这些 glock 每个 ystem 文件都存在，它们通常属于内节点（inode 和 metadata）或者资源组（资源组元数据）类型。

以下 **pminfo** 命令显示了每个挂载的 GFS2 文件系统的每种 Glock 类型数的列表。

```
# pminfo -f gfs2.glstats

gfs2.glstats.total
    inst [0 or "afc_cluster:data"] value 43680
    inst [1 or "afc_cluster:bin"] value 2091

gfs2.glstats.trans
    inst [0 or "afc_cluster:data"] value 3
    inst [1 or "afc_cluster:bin"] value 3

gfs2.glstats.inode
    inst [0 or "afc_cluster:data"] value 17
    inst [1 or "afc_cluster:bin"] value 17

gfs2.glstats.rgrp
    inst [0 or "afc_cluster:data"] value 43642
    inst [1 or "afc_cluster:bin"] value 2053

gfs2.glstats.meta
    inst [0 or "afc_cluster:data"] value 1
    inst [1 or "afc_cluster:bin"] value 1

gfs2.glstats.iopen
    inst [0 or "afc_cluster:data"] value 16
    inst [1 or "afc_cluster:bin"] value 16

gfs2.glstats.flock
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0

gfs2.glstats.quota
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0
```

```
gfs2.glstats.journal
    inst [0 or "afc_cluster:data"] value 1
    inst [1 or "afc_cluster:bin"] value 1
```

10.2.3. 检查处于等待状态的 glock 结构数量

最重要的拥有者标志是 H（holder：表示赋予请求的锁定）和 W（wait：等待请求完成时设置）。这些标记分别在赋予锁定请求和锁定请求队列中设置。

以下 **pminfo** 命令显示每个挂载的 GFS2 文件系统的具有 Wait(W)持有者标签的 glocks 数的列表。

```
# pminfo -f gfs2.holders.flags.wait

gfs2.holders.flags.wait
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0
```

如果您看到在资源组锁定中排队了大量等待的请求，那么可能有很多原因。一个原因可能是，在文件系统中，相对于资源组有大量的节点。另一个可能的原因是，文件系统可能接近完全被使用（平均来讲，需要较长的搜索时间）可以通过添加更多存储并使用 **gfs2_grow** 命令扩展文件系统来改善这两种情况。

10.2.4. 使用基于内核追踪点的指标检查文件系统操作延迟

GFS2 PMDA 支持从 GFS2 内核追踪点收集指标数据。默认情况下，禁用了对这些指标的读取。当收集指标时，激活这些指标会打开 GFS2 内核跟踪点来填充指标值。当启用了这些内核追踪点指标数据时，这可能会对性能吞吐量产生小的影响。

PCP 提供了 **pmstore** 工具，它允许您根据指标值修改 PMDA 设置。**gfs2.control.*** 指标允许切换 GFS2 内核追踪点。以下示例使用 **pmstore** 命令启用所有 GFS2 内核追踪点。

```
# pmstore gfs2.control.tracepoints.all 1
gfs2.control.tracepoints.all old value=0 new value=1
```

当运行这个命令时，PMDA 会对 **debugfs** 文件系统中的所有 GFS2 追踪点进行切换。PCP 中 GFS2 的可用指标的完整列表 中的"Complete Metric List"表解释了每个控制追踪点及其用法，对各个控制追踪点的作用及其可用选项的解释也可通过 **pminfo** 中的帮助开关获得。

GFS2 提升指标计算文件系统中提升请求的数量。这些请求由第一次尝试时发生的请求数和在初始提升请求后被批准的"others"请求数分开。第一次提升的减少与"other"提升的增加可表示文件争用的问题。

GFS2 降级请求指标（如提升请求指标）统计文件系统中的降级请求数。但是，这些也被分隔为来自当前节点的请求和来自系统中其他节点的请求。来自远程节点的大量降级请求可能会表示给定资源组在两个节点间出现竞争。

pminfo 工具显示有关可用的性能指标的信息。这个过程显示每个挂载的 GFS2 文件系统的 Wait(W)拥有者标签的 glocks 数。以下 **pminfo** 命令显示每个挂载的 GFS2 文件系统的具有 Wait(W)持有者标签的 glocks 数的列表。

```
# pminfo -f gfs2.latency.grant.all gfs2.latency.demote.all

gfs2.latency.grant.all
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0
```

```
gfs2.latency.demote.all
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0
```

做好的做法是，在工作负载运行时没有问题的情况下观察一般的值，因此当这些值不在正常的范围内时，可以注意到性能的变化。

例如，您可能会看到等待完成的提升请求数量的变化，而不是在第一次请求时就可以完成。您可以从以下命令的输出结果中看到相关的信息。

```
# pminfo -f gfs2.latency.grant.all gfs2.latency.demote.all

gfs2.tracepoints.promote.other.null_lock
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0

gfs2.tracepoints.promote.other.concurrent_read
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0

gfs2.tracepoints.promote.other.concurrent_write
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0

gfs2.tracepoints.promote.other.protected_read
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0

gfs2.tracepoints.promote.other.protected_write
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0

gfs2.tracepoints.promote.other.exclusive
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0
```

以下命令的输出允许您确定远程降级请求中的大幅增加（特别是如果来自其他集群节点）。

```
# pminfo -f gfs2.tracepoints.demote_rq.requested

gfs2.tracepoints.demote_rq.requested.remote
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0

gfs2.tracepoints.demote_rq.requested.local
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0
```

以下命令的输出可能会显示，出现无法解释的日志清除数量的增加。

```
# pminfo -f gfs2.tracepoints.log_flush.total]

gfs2.tracepoints.log_flush.total
    inst [0 or "afc_cluster:data"] value 0
    inst [1 or "afc_cluster:bin"] value 0
```

10.3. PCP 中 GFS2 可用指标的完整列表

下表描述了 GFS2 文件系统的 **pcp-pmda-gfs2** 软件包给出的性能指标的完整列表。

表 10.1. 完整指标数据列表

指标名称	描述
gfs2.glocks.*	有关从 glock 统计文件(**glocks**)收集的信息的指标，该指标计算目前挂载在系统上的每个 GFS2 文件系统的每个状态中存在的 glock 数。
gfs2.glocks.flags.*	计算给定 glocks 标签已存在的 glock 数量的指标范围
gfs2.holders.*	有关从 glock 统计文件(**glocks**)收集的信息的指标，该指标计算目前挂载在系统上的每个 GFS2 文件系统的每个锁状态的持有者的 glocks 数。
gfs2.holders.flags.*	带有给定拥有者标记的 glocks 数量的指标范围
gfs2.sbstats.*	有关从挂载在系统上的每个 GFS2 文件系统的超级块统计文件(**sbstats**)中收集的时间指标。
gfs2.glstats.*	有关从 glock 统计文件(gl**stats**)收集到的信息的指标，该指标计算目前挂载在系统上的每个 GFS2 文件系统的每种 glock 的数量。
gfs2.latency.grant.*	一个利用 **gfs2_glock_queue** 和 **gfs2_glock_state_change** 追踪点中数据的派生指标，以计算针对每个挂载的文件系统完成 glock 授权请求的平均延迟，以微秒为单位。这个指标可用于发现，当授权延迟增加时降低文件系统性能的问题。
gfs2.latency.demote.*	一个利用 **gfs2_glock_state_state_change** 和 **gfs2_demote_rq** 追踪点中数据的派生指标，来为每个挂载的文件系统要完成的 glock 降级请求计算平均延迟，以微秒为单位。这个指标可用于发现，当降级延迟增加时降低文件系统性能的问题。
gfs2.latency.queue.*	一个利用 **gfs2_glock_queue** 追踪点中数据的派生指标，来为每个挂载的文件系统要完成的 glock 队列请求计算平均延迟。
gfs2.worst_glock.*	一个利用 **gfs2_glock_lock_time** 追踪点中数据的派生指标，来为每个挂载的文件系统计算感知到的 "current worst glock"。如果同一锁被建议多次，则这个指标用于发现潜在的锁竞争和文件系统减慢。
gfs2.tracepoints.*	有关当前挂载在系统上的每个文件系统的 GFS2 **debugfs** 追踪点的输出的指标。这些指标的每个子类型（每个 GFS2 追踪点中的一个）都可以被控制，无论是启用或关闭控制指标。
gfs2.control.*	配置用于在 PMDA 中打开或关闭指标记录的指标数据。通过 **pmstore** 工具控制切换指标。

10.4. 执行最小 PCP 设置来收集文件系统数据

此流程概述了如何安装最小 PCP 设置来收集关于 Red Hat Enterprise Linux 的统计信息的说明。这个设置涉及在产品系统中添加收集数据以便进一步分析所需的最小软件包数量。

可以使用其他 PCP 工具分析 **pmlogger** 输出生成的 **tar.gz** 存档，并可与其他性能信息源进行比较。

步骤

1. 安装所需的 PCP 软件包。

```
# yum install pcp pcp-pmda-gfs2
```

2. 为 PCP 激活 GFS2 模块。

```
# cd /var/lib/pcp/pmdas/gfs2
# ./Install
```

3. 启动 **pmcd** 和 **pmlogger** 服务。

```
# systemctl start pmcd.service
# systemctl start pmlogger.service
```

4. 在 GFS2 文件系统中执行操作。

5. 停止 **pmcd** 和 **pmlogger** 服务。

```
# systemctl stop pmcd.service
# systemctl stop pmlogger.service
```

6. 收集输出并将其保存到根据主机名和当前日期和时间命名的 **tar.gz** 文件中。

```
# cd /var/log/pcp/pmlogger
# tar -czf $(hostname).$(date+%F-%Hh%M).pcp.tar.gz $(hostname)
```

10.5. 其他资源

- GFS2 追踪点和 glock debugfs 接口。
- 使用 Performance Co-Pilot 监控性能
- Performance Co-Pilot(PCP)文章、解决方案、教程和白皮书的索引