# Red Hat Enterprise Linux 7

## Global File System 2

GFS2 ファイルシステムの設定および管理

Last Updated: 2020-10-02

# Red Hat Enterprise Linux 7 Global File System 2

GFS2 ファイルシステムの設定および管理

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

## 法律上の通知



## 概要

本ガイドでは、Red Hat Enterprise Linux 7 向けの Red Hat GFS2 (Red Hat Global File System 2) の設定と管理について説明します。

# 目次

# 第1章 GFS2 の概要

Red Hat GFS2 ファイルシステムは、 64 ビットの対称クラスターファイルシステムです。これは、一般的なブロックデバイスを共有している複数のノード間において共有の名前空間を利用でき、整合性を管理できます。GFS2 ファイルシステムは、ローカルファイルシステムに可能な限り近い機能を利用できるようにし、ノード間の完全なクラスターの完全性を確立するためのものです。場合によっては、Linux ファイルシステム API では、GFS2 のクラスター化の特性を完全に透明化できません。例えば、クラスター環境では指定のプロセス ID が別のノード用である場合もあるため、GFS2 で POSIX ロックを使用するプログラムは、**GETLK** 関数の使用を避けるべきです。ただし、多くの場合、GFS2 ファイルシステムの機能性は、ローカルファイルシステムのものと同じです。

Red Hat Enterprise Linux (RHEL) Resilient Storage Add-On では GFS2 を利用できます。GFS2 に必要なクラスター管理については、RHEL High Availability Add-On に依存します。High Availability Add-On の説明は、『Red Hat クラスターの設定と管理』 を参照してください。

**gfs2.ko** カーネルモジュールは GFS2 ファイルシステムを実装しており、GFS2 クラスターノードにロードされます。

GFS2 環境を最大限に利用するためにも、基礎となる設計に起因するパフォーマンス事情を考慮することが重要です。GFS2 はローカルファイルシステムのように、頻繁に使用されるデータのローカルキャッシングによりパフォーマンスを向上するため、ページキャッシュに依存します。クラスター内のノードにわたる一貫性を維持するため、キャッシュ制御は、*glock* 状態のマシンが行います。glocks やそのパフォーマンス状態の説明は、「GFS2 のノードロック機能」 を参照してください。

この章では、GFS2 に対する理解を深めるための予備知識となる、基本的な情報を簡潔にまとめています。

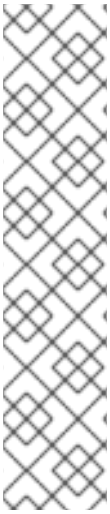
## 1.1. GFS2 サポート制限

表1.1「GFS2 サポート制限」 では、GFS2 が現在サポートしているファイルシステムの最大サイズと最大ノード数をまとめています。

**表1.1 GFS2 サポート制限**

| | |
|---|---|
| ノードの最大数 | 16 (x86、PowerVM 上の Power8)<br>4 (z/VM 下の s390x) |
| ファイルシステムの最大サイズ | すべての対応アーキテクチャー上の 100TB |

GFS2 は、理論的には 8 EB のファイルシステムに対応できる 64 ビットアーキテクチャに基づいています。お使いのシステムに、現在対応している以上の GFS2 ファイルシステムが必要な場合は、Red Hat サービス担当者までご連絡ください。



**注記**

GFS2 ファイルシステムはスタンドアローンシステムで実装したり、クラスター構成の一部として実装したりできますが、Red Hat Enterprise Linux 7 リリースでは Red Hat はシングルノードファイルシステムとしての GFS2 の使用をサポートしません。Red Hat は、シングルノード向けに最適化され、一般的にクラスターファイルシステムよりもオーバーヘッドが小さい複数の高パフォーマンスのシングルノードファイルシステムをサポートします。また、Red Hat は、シングルノードがファイルシステムをマウントする必要がある場合のみ、GFS2 ではなく高パフォーマンスのシングルノードファイルシステムを使用することをお勧めします。

Red Hat は、クラスターファイルシステムのスナップショットのマウントを目的とした (例: バックアップ)、単一ノード GFS2 ファイルシステムを引き続きサポートしています。

ファイルシステムのサイズを決定する際には、復旧時のニーズを考慮してください。非常に大規模なファイルシステムでは、**fsck.gfs2** コマンドの実行に時間がかかり、大容量のメモリーを消費する可能性があります。また、ディスクやディスクサブシステムの障害発生時には、使用するバックアップメディアの速度によって復旧時間が制限されます。**fsck.gfs2** コマンドに必要なメモリー容量については、「GFS2 ファイルシステムの修復」 を参照してください。

GFS2 ファイルシステムは LVM 外でも使用できますが、Red Hat は CLVM 論理ボリュームで作成された GFS2 ファイルシステムのみサポートします。CLVM は Resilient Storage アドオンに含まれます。これはクラスター全体での LVM の実装であり、クラスター内で LVM 論理ボリュームを管理する CLVM デーモン **clvmd** で有効になります。このデーモンにより、LVM2 を使用してクラスター全体で論理ボリュームの管理が可能となり、クラスター内のすべてのノードで論理ボリュームを共有できるようになります。LVM ボリュームマネージャーについては、『Logical Volume Manager Administration』 を参照してください。

**注記**

GFS2 ファイルシステムをクラスターファイルシステムとして設定する場合は、クラスター内の全ノードが共有ストレージにアクセス可能であることを確認する必要があります。共有ストレージにアクセスできるノードとできないノードが混在する、非対称型のクラスター構成はサポートしていません。この場合、全ノードが実際に GFS2 ファイルシステム自体をマウントしている必要はありません。

## 1.2. 新機能と変更点

本セクションでは、Red Hat Enterprise Linux 7 の初期以降のリリースに含まれる GFS2 ファイルシステムの新機能および変更された機能と、GFS2 に関するドキュメンテーションをリストします。

### 1.2.1. Red Hat Enterprise Linux 7.0 の新機能と変更された機能

Red Hat Enterprise Linux 7.0 には、以下のドキュメンテーションと機能の更新および変更が含まれます。

- Red Hat Enterprise Linux 7 では、GFS2 ファイルシステムを含むクラスターの場合に、*5章 クラスターでの GFS2 ファイルシステムの設定*に記載された手順に従って Pacemaker でクラスターを設定する必要があります。
- **gfs2_tool** コマンドは Red Hat Enterprise Linux 7 ではサポートされません。 **gfs2_tool** の代替機能の概要については、「Red Hat Enterprise Linux 7 での gfs2_tool の代替機能」 を参照してください。

### 1.2.2. Red Hat Enterprise Linux 7.1 の新機能および変更された機能

Red Hat Enterprise Linux 7.1 では、*付録A Performance Co-Pilot による GFS2 パフォーマンスの分析* が更新されました。

さらに、ドキュメント全体にわたり、技術的な内容の若干の修正と明確化を行いました。

### 1.2.3. Red Hat Enterprise Linux 7.2 の新機能および変更された機能

ドキュメントには若干の技術的な修正とわかりやすい記載にするための変更が加えられています。

### 1.2.4. Red Hat Enterprise Linux 7.4 の新機能および変更された機能

Red Hat Enterprise Linux 7.4 では、 Security Enhanced Linux (SELinux) は、GFS2 ファイルシステムで使用できます。SELinux と GFS2 の説明は、「GFS2 上の SELinux」 を参照してください。

## 1.3. GFS2 を設定する前に

GFS2 のインストールと設定を行う前に、以下に挙げる、GFS2 ファイルシステムの主要な特性を確認しておいてください。

**GFS2 ノード**

クラスター内のどのノードで GFS2 ファイルシステムをマウントするかを決定します。

**ファイルシステムの数**

初めに作成する GFS2 ファイルシステムの数を決定します (ファイルシステムは後で追加することができます)。

**ファイルシステム名**

各ファイルシステムの一意の名前を決定します。この名前は、クラスター上の全 **lock_dlm** ファイルシステムで一意でなければなりません。各ファイルシステム名は、パラメーター変数の形式にする必要があります。たとえば、本ガイドで例示した手順の中では、**mydata1** および **mydata2** というファイルシステム名を使用しています。

**ジャーナル**

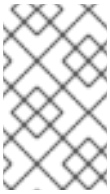
GFS2 ファイルシステム用のジャーナルの数を決定します。GFS2 ファイルシステムをマウントするノード毎に 1 つのジャーナルが必要になります。GFS2 では、後日、追加のサーバーがファイルシステムをマウントする時に、ジャーナルを動的に追加することができます。GFS2 ファイルシステムへのジャーナルの追加については、「GFS2 ファイルシステムへジャーナルの追加」 を参照してください。

**ストレージデバイスとパーティション**

ファイルシステム内に論理ボリュームを作成する際に使用するストレージデバイスとパーティションを決めます (CLVM を使用)。

**時間プロトコル**

GFS2 ノードのクロックが同期されていることを確認します。Red Hat Enterprise Linux ディストリビューションで提供されている Precision Time Protocol (PTP) または Network Time Protocol (NTP) ソフトウェア (設定に必要な場合) を使用することを推奨します。



**注記**

不要な inode 時間スタンプの更新を防ぐには、GFS2 ノード内のシステムクロックの時間差が数分以内になるように設定する必要があります。不要な inode 時間スタンプの更新はクラスターのパフォーマンスに深刻な影響を与えます。

**注記**

同じディレクトリー内で、複数のノードから多数の作成/削除の操作が同時に実行されると、パフォーマンス上の問題が発生する可能性があります。ご使用のシステムでこのようにパフォーマンス上の問題が生じた場合は、ノードによるファイルの作成と削除は、可能な限りそのノード固有のディレクトリーに局限すべきです。

GFS2 ファイルシステムの作成、使用、およびメンテナンスに関する更なる推奨事項は、*2章GFS2 の設定および操作における考慮事項* を参照してください。

## 1.4. GFS2 のインストール

Red Hat High Availability Add-On に必要なパッケージ以外に、GFS2 向けの **gfs2-utils** パッケージと CLVM (Clustered Logical Volume Manager) 向けの **lvm2-cluster** パッケージをインストールする必要があります。**lvm2-cluster** と **gfs2-utils** のパッケージは **ResilientStorage** チャンネルに含まれるため、パッケージをインストールする前にこのチャンネルを有効にする必要があります。

以下の **yum install** コマンドを使用して、Red Hat High Availability Add-On ソフトウェアのパッケージをインストールします。

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

Red Hat High Availability Add-On およびクラスター管理の一般情報については、『Cluster Administration』 を参照してください。

## 1.5. RED HAT ENTERPRISE LINUX 7 での GFS2_TOOL の代替機能

**gfs2_tool** コマンドは、Red Hat Enterprise Linux 7 ではサポートされません。 表1.2「Red Hat Enterprise Linux 7 での gfs2_tool の同等な機能」では、Red Hat Enterprise Linux 7 の **gfs2_tool** コマンドオプションの同等の機能について簡単に説明しています。

**表1.2 Red Hat Enterprise Linux 7 での gfs2_tool の同等な機能**

| gfs2_tool オプション | 代替機能 |
|---|---|
| **clearflag *Flag File1 File2 ...***<br>ファイルの属性フラグを消去する | Linux の標準的な **chattr** コマンド |
| **freeze *mountpoint***<br>GFS2 ファイルシステムをフリーズ (休止) する | Linux の標準的な **fsfreeze -f *mountpoint*** コマンド |

| gfs2_tool オプション | 代替機能 |
| --- | --- |
| **gettune *mountpoint***<br><br>チューニングパラメーターの現在の値を出力する | 多くの場合、**mount** (**get mount options**) によって置き換えられます。他のチューニングパラメーターは、各 **sysfs** ファイルから取得できます (**/sys/fs/gfs2/dm-3/tune/***)。 |
| **journals *mountpoint***<br><br>GFS2 ファイルシステムのジャーナルに関する情報を出力する | ジャーナルの情報は **gfs2_edit -p journals *device*** で取得できます。このコマンドは、ファイルシステムをマウントしている状態で使用できます。<br><br>`# gfs2_edit -p journals /dev/clus_vg/lv1`<br>`Block #Journal Status:  of 2620416 (0x27fc00)`<br>`---------- Journal List ----------`<br>`-`<br>`journal0: 0x14  128MB clean.`<br>`journal1: 0x805b  128MB clean.`<br>`----------` |
| **lockdump *mountpoint***<br><br>該当するファイルシステムに対してこのマシンが保持するロックに関する情報を出力する | GFS2 ロックの情報は、**debugfs** をマウントし、以下のようなコマンドを実行して取得できます。<br><br>`# cat /sys/kernel/debug/gfs2/`*clustername:file_system_name*`/glocks` |
| **sb *device* proto [*newvalue*]**<br><br>ロッキングプロトコルを表示 (場合によっては、さらに置換) する | ロッキングプロトコルの現在の値を取得するには、次のコマンドを使用します。<br><br>`# tunegfs2 -l `*device*` \| grep protocol`<br><br>ロッキングプロトコルの現在の値を置換するには、次のコマンドを使用します。<br><br>`# tunegfs2 -o lockproto=lock_dlm `*device* |
| **sb *device* table [*newvalue*]**<br><br>ロッキングテーブルの名前を表示 (場合によっては、さらに置換) する | ロッキングテーブル名の現在の値を取得するには、次のコマンドを使用します。<br><br>`# tunegfs2 -l `*device*` \| grep table`<br><br>ロッキングテーブル名の現在の値を置換するには、次のコマンドを使用します。<br><br>`# tunegfs2 -o locktable=`*file_system_name device* |
| **sb *device* ondisk [*newvalue*]**<br><br>**ondisk** フォーマット番号を表示 (場合によっては、さらに置換) する | このタスクは実行しないでください。 |

| gfs2_tool オプション | 代替機能 |
| --- | --- |
| **sb *device* multihost [*newvalue*]**<br><br>**multihost** フォーマット番号を表示 (場合によっては、さらに置換) する | このタスクは実行しないでください。 |
| **sb *device* uuid [*newvalue*]**<br><br>**uuid** 値を表示 (場合によっては、さらに置換) する | **uuid** の現在の値を取得するには、次のコマンドを使用します。<br><br># tunegfs2 -l *device* \| grep UUID<br><br>**uuid** の現在の値を置換するには、次のコマンドを使用します。<br><br># tunegfs2 -U *uuid device* |
| **sb *device* all**<br><br>GFS2 スーパーブロックを出力する | # tunegfs2 -l *device* |
| **setflag *Flag File1 File2 ...***<br><br>ファイルの属性フラグを設定する | Linux の標準的な **chattr** コマンド |
| **settune *mountpoint parameter newvalue***<br><br>チューニングパラメーターの値を設定する | 多くの場合、**mount** (**-o remount** とオプション) によって置き換えられます。他のチューニングパラメーターは、各 **sysfs** ファイルで設定できます (/**sys**/**fs**/**gfs2**/***cluster_name:file_system_name***/**tune**/*)。 |
| **unfreeze *mountpoint***<br><br>GFS2 ファイルシステムをフリーズ解除する | Linux の標準的な **fsfreeze -unfreeze *mountpoint*** コマンド |
| **version**<br><br>gfs2_tool コマンドのバージョンを表示する | 該当なし |
| **withdraw *mountpoint***<br><br>GFS2 により該当するファイルシステムを強制シャットダウンする | # echo 1 > /sys/fs/gfs2/*cluster_name:file_system_name*/tune/withdraw |

# 第2章 GFS2 の設定および操作における考慮事項

Global File System 2 (GFS2) ファイルシステムにより、単一クラスター内の複数のコンピューター (「ノード」) が同じストレージを協調的に共有することが可能となります。このような連携を実現して複数ノード間におけるデータの一貫性を維持するには、ノードで、ファイルシステムリソースを対象にクラスター全体にわたるロックスキームを採用する必要があります。このロッキングスキームは、TCP/IP などの通信プロトコルを使用してロッキング情報を交換します。

本章に記載の推奨事項にしたがって、パフォーマンスを向上させることができます。これには、GFS2 ファイルシステムの作成、使用、メンテナンスに関する内容が含まれます。

**重要**

Red Hat High Availability Add-On の導入がニーズを満たし、サポート可能であることを確認してください。Red Hat 認定担当者に相談して設定を確認してからデプロイするようにしてください。

## 2.1. フォーマットに関する考慮事項

本セクションでは、パフォーマンスを最適化するための GFS2 ファイルシステムのフォーマット方法についての推奨事項を取り上げます。

### 2.1.1. ファイルシステムサイズ: 小さい方が好ましい

GFS2 は、64 ビットアーキテクチャーをベースにしており、理論的には 8 EB の ファイルシステムに対応します。ただし、64 ビットのハードウェアで現在サポートしている GFS2 ファイルシステムの最大サイズは 100 TB です。

GFS2 の大型ファイルシステムは可能ですが、それが推奨されるということではない点に注意してください。GFS2 の経験則では、小さいほど好ましく、10 TB のファイルシステムを 1 つ使用するよりも、1 TB のファイルシステムを 10 使用した方がよいことになります。

GFS2 ファイルシステムのサイズを小さくとどめることが推奨される理由として、以下の点があげられます。

- 各ファイルシステムのバックアップ所要時間が短縮されます。
- **fsck.gfs2** コマンドでファイルシステムをチェックする必要がある場合の所要時間が短縮されます。
- **fsck.gfs2** コマンドでファイルシステムをチェックする必要がある場合は、必要なメモリーが少なくなります。

また、メンテナンス対象となるリソースグループが少なくなることにより、パフォーマンスが向上します。

当然ながら、GFS2 ファイルシステムのサイズを小さくしすぎた場合には、容量が不足してしまう可能性があり、それが影響をもたらすことになります。サイズを決定する前には、ご使用になる環境独自のユースケースを考慮すべきです。

### 2.1.2. ブロックサイズ: デフォルト (4K) ブロックを推奨

**mkfs.gfs2** コマンドは、デバイストポロジーに基づいて最適なブロックサイズの算出を試みます。一般的には、4K ブロックが推奨ブロックサイズです。これは、Linux のデフォルトのページサイズ (メモ

リー) が 4K であるためです。GFS2 は他のファイルシステムとは異なり、4K カーネルバッファーを使用して大半の操作を実行します。ブロックサイズが 4K の場合は、カーネルで行う必要のあるバッファー操作の作業が軽減されます。

パフォーマンスを最適化するには、デフォルトのブロックサイズを使用することを推奨します。デフォルト以外のブロックサイズを使用する必要があるのは、極めて小さなファイルを多数格納する効率性の高いストレージを必要としている場合のみでしょう。

### 2.1.3. ジャーナル数: マウントするノードにつき 1 つ

GFS2 では、クラスター内でファイルシステムをマウントする必要のあるノード 1 つにつき 1 ジャーナルが必要です。たとえば、16 ノードで構成されるクラスターで、2 つのノードのみからファイルシステムをマウントする必要が場合に必要なジャーナルは 2 つのみとなります。第 3 のノードからマウントする必要がある場合には、**gfs2_jadd** コマンドでジャーナルを追加することができます。GFS2 では、オンザフライでジャーナルを追加することが可能です。

### 2.1.4. ジャーナルサイズ: 通常はデフォルト (128 MB) が最適

**mkfs.gfs2** コマンドを実行して GFS2 ファイルシステムを作成する際には、ジャーナルのサイズを指定することができます。サイズを指定しなかった場合には、デフォルトで 128 MB に設定されます。これは大半のアプリケーションで最適な値です。

一部のシステム管理者は、128 MB は大きすぎると考えて、ジャーナルのサイズを最小限の 8 MB に縮小したり、より慎重となって 32 MB に指定しようとする場合があります。このような値に指定しても機能はするかもしれませんが、パフォーマンスに深刻な影響を及ぼす可能性があります。多くのジャーナリングシステムと同様に、GFS2 がメタデータの書き込みを行う際には毎回、メタデータは配置される前にジャーナルにコミットされます。これにより、ファイルがクラッシュしたり、電源が切断された場合でも、マウント時にジャーナルが自動再生される際に全メタデータが確実に復旧されます。しかし、ファイルシステムアクティビティーがそれほど大量でなくとも 8 MB ジャーナルは満たされてしまい、ジャーナルが満杯になると、GFS2 はストレージへの書き込みを待つ必要があるため、パフォーマンスが低下してしまいます。

通常は、128 MB のデフォルトジャーナルサイズを使用することを推奨します。使用するファイルシステムが極めて小型 (例: 5 GB) の場合には、128 MB のジャーナルは実用的でない可能性があります。使用するファイルシステムが大型で、十分な容量がある場合には、256 MB のジャーナルを使用するとパフォーマンスが向上するでしょう。

### 2.1.5. リソースグループのサイズと数

GFS2 ファイルシステムが **mkfs.gfs2** コマンドで作成されると、ストレージが均一のサイズに分割されます。この分割された部分はリソースグループとして知られています。最適のリソースグループのサイズ (32MB から 2GB) を推測しようと試行します。**mkfs.gfs2** コマンドに**-r** オプションをつけることで、デフォルトより優先させることができます。

最適なリソースグループサイズは、ファイルシステムの使用法によって左右されます。使用率はどの程度になるか、極度に断片化されるかどうかなどの点を考慮してください。

どのサイズが最高のパフォーマスを達成できるか、様々なリソースグループサイズを試してみてください。テストクラスターで試してから、完全な実稼働環境に GFS2 をデプロイすることがベストプラクティスとなっています。

使用するファイルシステムのリソースグループ数が多過ぎる (各リソースグループが小さい) 場合には、ブロック割り当てで何万 (または何十万) にも及ぶリソースグループを対象に空きブロックを検索するため、多大な時間の無駄となる場合があります。ファイルシステムの使用率が高いほど、検索対象となる

リソースグループが多くなり、それらのすべてにクラスター全体のロックが必要となってしまい、パフォーマンスの低下をもたらします。

しかし、ファイルシステムの中のリソースグループが少なすぎる (各リソースグループが大きすぎる) 場合、頻繁に、同じリソースグループをロックしようとブロックの割り当てが競合する可能性があり、パフォーマンスにも影響を及ぼします。たとえば、10GB のファイルシステムで 5 つの 2GB リソースグループに分割されている場合、利用しているクラスターのノードは、32MB のリソースグループ 320 個に分割されている場合に比べて、リソースグループの競合が激しくなります。また、ファイルシステムの空きがほぼない場合など、さらに問題が悪化します。理由は、空きがあるブロックが見つかるまでに複数のリソースグループを検索する必要がある場合もあるからです。以下のような 2 種類の方法で、GFS2 はこの問題を軽減しようとしています。

- 第 1 の方法は、リソースグループが完全に満杯になった場合に、GFS2 はその情報を記憶し、以降の割り当てでは、(ブロックが解放されるまで) そのリソースグループをチェックしないようにします。ファイルは決して削除せず、競合度が低くなります。ただし、ご使用のアプリケーションが、ほぼ満杯のファイルシステムで絶えずブロックを削除して新規ブロックを割り当てている場合にはいる場合には、競合度が極めて高くなり、パフォーマンスに深刻な悪影響を及ぼします。

- 第 2 の方法は、新しいブロックが既存ファイルに追加されると (例：添付)、GFS2 は同じリソースグループ内で、ファイルとして新しいブロックを結合しようとします。こうすることで、パフォーマンスの向上を図ります。つまりディスクが回転している場合は、物理的に距離が短いほうが検索にかかる時間が短縮されます。

最悪のシナリオとしては、集約ディレクトリーが 1 つあり、その中のノードすべてがファイルを作成しようとする場合などです。これは、全ノードが同じリソースグループをロックしようと常に競合するためです。

## 2.2. ファイルシステムの断片化

Red Hat Enterprise Linux には GFS2 向けのデフラグツールはありませんが、**filefrag** ツールでファイルを識別して一時ファイルにコピーし、この一時ファイルの名前を変更することでオリジナルファイルを置き換えることにより、個別ファイルをデフラグすることができます。

## 2.3. ブロック割り当てにおける問題

本セクションでは、GFS2 ファイルシステムにおけるブロック割り当てに関連した問題の概要を説明します。データの書き込みだけを行うアプリケーションでは通常、ブロックの割り当て方法や割り当て先については問題になりませんが、ブロック割り当ての仕組みを多少理解しておくことにより、パフォーマンスを最適化することができます。

### 2.3.1. ファイルシステムに空き領域を確保する

GFS2 ファイルシステムがほぼ満杯になると、ブロックアロケーターが割り当てる新規ブロック用の領域を見つけるのが難しくなります。結果として、多くの場合、アロケーターによって割り当てられたブロックはリソースグループの最後または小さいスライス領域に詰め込まれるため、ファイルが断片化する可能性が非常に高くなります。このようなファイルの断片化によって、パフォーマンス上の問題が発生する場合があります。また、GFS2 がほぼ満杯になると、GFS2 のブロックアロケーターが複数のリソースグループを検索するため時間がかかり、十分な空き領域があるファイルシステムでは必ずしも発生しないロック競合が生じます。これにより、パフォーマンス上の問題が発生することがあります。

こういった理由で、 (ワークロードにより数値は変わってきますが) 85% 以上使用済みのファイルシステムを実行しないよう推奨しています。

### 2.3.2. 可能な場合は各ノードで独自のファイルを割り当てる

全ファイルが単一のノードによって割り当てられて、その他のノードがこれらのファイルにブロックを追加する必要がある場合には、分散ロックマネージャー (DLM) の動作の仕組みが原因となって、ロック競合の発生度が高くなります。

GFS (バージョン 1) では、クラスター全体のロックを制御する中央ロックマネージャーによって全ロックが管理されていました。この Grand Unified Lock Manager (GULM) は単一障害点であったため、問題がありました。GFS2 で置き換えられたロックスキーム DLM は、クラスター全体にロックを分散します。クラスター内のいずれかのノードが停止した場合、そのロックは他のノードによって復旧されます。

DLM では、リソース (ファイルなど) をロックする最初のノードがそのロックの “ロックマスター” となります。その他のノードもリソースをロックすることができますが、その前にロックマスターに許可を求める必要があります。各ノードは、ロックマスターのロックを認識しており、どのノードにロックを貸しているかを認識します。マスターノード以外のノード上のロックをロッキングするには、停止してロックのマスターに許可を求める必要があるため、マスターノード上のロックのロッキングの方がはるかに高速となります。

多くのファイルシステムと同様に、GFS2 のアロケーターは同じファイルのブロックを近くにまとめて、ディスクヘッドの移動を少なくし、パフォーマンスを向上させます。多くの場合、ファイルにブロックを割り当てるノードは、新規ブロック用に同じリソースグループを使用してロックする必要があります (そのリソースグループの全ブロックが使用中である場合を除く)。ファイルが含まれているリソースグループのロックマスターがそのデータブロックを割り当てると、ファイルシステムの動作が高速化します (つまり、ファイルを最初に開いたノードが新規ブロックの書き込みをすべて行うようにした方が動作が高速となります)。

### 2.3.3. 可能な場合には事前に割り当てる

ファイルが事前割り当て済みの場合、ブロック割り当てを完全に回避して、ファイルシステムの動作を効率化させることができます。GFS2 の新バージョンは、データブロックの事前割り当てに使用できる **fallocate**(1) システムコールを実装しています。

## 2.4. クラスターに関する考慮事項

システムを構成するノードの数を決定する際には、高可用性とパフォーマンスの間でトレードオフがある点に注意してください。ノード数が増えるにしたがって、ワークロードのスケーリングが困難となります。このため、Red Hat は 16 ノード以上のクラスターファイルシステムデプロイで GFS2 の使用はサポートしていません。

クラスターファイルシステムのデプロイは、単一ノードデプロイメントと単純に置き換えることはできません。新規インストールには、8〜12 週間程度のテスト期間を設けて、システムをテストし、必要とされるパフォーマンスレベルで動作することが推奨されます。この期間中に、パフォーマンス上または機能上の問題を解決することができ、Red Hat のサポートチームにもご相談いただけます。

クラスターのデプロイを検討中のお客様は、デプロイメントを行う前に、Red Hat サポートによる設定のレビューを受けておくことをお勧めします。これにより、後日にサポートの問題が発生するのを未然に防ぐことができます。

## 2.5. 使用に関する考慮事項

本セクションでは、GFS2 の使用に関する一般的な推奨事項について説明します。

### 2.5.1. マウントオプション: noatime と nodiratime

通常、GFS2 ファイルシステムは **noatime** および **nodiratime** の引数を使用してマウントすることを推奨します。これにより、アクセスする度に GFS2 がディスク inode を更新する時間を短縮することができます。GFS2 ファイルシステムパフォーマンスにおける、これら 2 つの引数の効果は、「GFS2 のノードロック機能」 を参照してください。

### 2.5.2. VFS チューニングオプション: リサーチと実験

すべての Linux ファイルシステムと同様に、GFS2 は仮想ファイルシステム (VFS) と呼ばれる層の最上部に位置します。**sysctl**(8) コマンドを使用して VFS 層をチューニングすることにより、配下の GFS2 パフォーマンスを向上させることができます。たとえば、状況に応じて **dirty_background_ratio** および **vfs_cache_pressure** の値を調整することができます。現在の値を取得するには、以下のコマンドを使用します。

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

以下のコマンドは、値を調整します。

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

**/etc/sysctl.conf** ファイルを編集すると、これらのパラメーターを永久的に変更することができます。

ユースケースに応じた最適な値を見出すには、完全な実稼働環境にデプロイする前に、さまざまな VFS オプションをリサーチして、テスト用クラスター上で実験を行ってください。

### 2.5.3. GFS2 上の SELinux

Red Hat Enterprise Linux 7.4 以降では、GFS2 ファイルシステムで Security Enhanced Linux (SELinux) を使用できます。

GFS2 で SELinux を使用すると、パフォーマンスに多少の影響が起こります。これを回避するには、Enforcing モードで SELinux を動作させているシステム嬢であっても GFS2 で SELinux を使用するべきではありません。GFS2 ファイルシステムをマウントする場合は、**mount**(8) man ページで説明されたように **context** オプションのいずれかを使用して SELinux が各ファイルシステムオブジェクトの **seclabel** 要素の読み取りを試行しないようにしてください。また、これにより、**seclabel** 要素を含む拡張属性ブロックの別のディスク読み取りが回避され、処理が高速化されます。

たとえば、SELinux が強制モードであるシステムでは、ファイルシステムに Apache のコンテンツを含める場合に、以下の **mount** コマンドを使用して GFS2 ファイルシステムをマウントできます。このラベルはファイルシステム全体に適用されます。メモリーに残り、ディスクには書き込まれません。

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

ファイルシステムに Apache のコンテンツが含まれるかどうかわからない場合は、ラベル **public_content_rw_t** または **public_content_t** を使用するか、新しいラベルを定義し、それに関するポリシーを定義できます。

Pacemaker クラスターでは、GFS2 ファイルシステムを管理するために常に Pacemaker を使用する必要があります。*5章 クラスターでの GFS2 ファイルシステムの設定* で説明されているように、GFS2 ファイルシステムリソースを作成するときはマウントオプションを指定できます。

### 2.5.4. GFS2 上における NFS セットアップ

GFS2 のロッキングサブシステムおよびクラスター化の特性は複雑度が高いため、GFS2 上で NFS をセットアップするにあたっては、数多くの予防措置を取り、細心の注意を払う必要があります。本セクションでは、GFS2 ファイルシステム上で NFS サービスを設定するにあたって考慮すべき注意事項について説明します。

**警告**

GFS2 ファイルシステムが NFS でエクスポートされている場合は、**localflocks** オプションを使用してファイルシステムをマウントする必要があります。**localflocks** オプションを使用すると、複数の場所から GFS2 ファイルシステムにアクセスすると安全ではなくなり、複数のノードから GFS2 を同時にエクスポートすることは現実的ではなくなるため、この設定を使用して同時に 1 つのノードに GFS2 ファイルシステムをマウントすることがサポート要件となります。これの目的は、各サーバからの POSIX ロックをローカルに強制することです。つまり、クラスタ化されておらず、互いに独立しています。なぜなら、GFS2 がクラスターのノード間で NFS から POSIX ロックを実装しようとすると、多くの問題が発生するためです。NFS クライアントで実行しているアプリケーションでは、別のサーバーから 2 台のクライアントがマウントしている場合に、ローカルの POSIX ロックにより、その 2 台のクライアントが同じロックを同時に保持することがあり、これによりデータが破損する場合があります。すべてのクライアントが 1 台のサーバーから NFS をマウントすると、複数のサーバーが同じロックを別々に許可するという問題が解消されます。**localflocks** オプションでファイルシステムをマウントするべきかどうかがわからない場合は、このオプションを使用しないでください。すぐに、データ損失を防ぐための適切な設定を Red Hat サポートにご相談ください。NFS を介した GFS2 のエクスポートは、状況によっては技術的にサポートされていますが、推奨はされません。

NFS を除くすべての GFS2 アプリケーションは、**localflocks** を使用してファイルシステムをマウントしないでください。これにより、GFS2 はクラスター (クラスター全体) におけるすべてのノード間で POSIX ロックと flocks を管理できます。**localflocks** を指定して NFS を使用しないと、クラスター内のその他のノードは相互の POSIX ロックと flocks を認識しないため、クラスター環境において不安定になります。

GFS2 システム上で NFS サービスを設定する際には、ロックについて考慮する以外に、以下のような点も検討する必要があります。

- Red Hat は、以下のような特性を持つアクティブ/パッシブ構成のロックを備えた NFSv3 を使用する Red Hat High Availability Add-On 設定のみをサポートしています。
    - バックエンドファイルシステムは、2〜16 のノードクラスターで稼働している GFS2 ファイルシステムです。
    - NFSv3 サーバーは、単一クラスターノードから GFS2 ファイルシステム全体を一度にエクスポートするサービスとして定義されます。
    - NFS サーバーは、一つのクラスターノードから他のクラスターノードへのフェイルオーバーが可能です (アクティブ/パッシブ構成)。
    - NFS サーバー経由 **以外の** GFS2 ファイルシステムへのアクセスは許可されません。これには、ローカルの GFS2 ファイルシステムアクセスと、Samba またはクラスター化された

Samba を介したアクセスの両方が含まれます。マウントしているクラスターノードからローカルのファイルシステムにアクセスすると、データが破損する場合があります。

- システム上で NFS クォータはサポートされていません。

この構成では、ノードで障害が発生しても、NFS サーバーをあるノードから別のノードへフェイルオーバーするときに **fsck** コマンドを実行する必要がないため、ファイルシステムに高可用性 (HA) が提供され、システムダウンタイムが削減されます。

- **fsid=** NFS オプションは、GFS2 の NFS エクスポートには必須です。
- クラスターに問題が生じた場合 (例: クラスタが定足数に達せず、フェンシングが失敗した場合) には、クラスター化された論理ボリュームと GFS2 ファイルシステムはフリーズされ、クラスターが定足数を満たすようになるまではアクセスできません。この手順で説明したような、単純なフェイルオーバー解決方法がご使用のシステムに最も適切かどうかを判断する際には、この可能性を考慮する必要があります。

### 2.5.5. GFS2 上の Samba (SMB または Windows) ファイルサービス

アクティブ/アクティブ構成が可能な CTDB を使用する GFS2 ファイルシステムから Samba (SMB または Windows) ファイルサービスを使用できます。

Samba 外からの Samba 共有内のデータへの同時アクセスには対応していません。GFS2 クラスターのリースは Samba ファイルサービスの処理速度を低下させるため現在、対応していません。Samba のサポートポリシーの詳細は、Red Hat カスタマーポータルの Support Policies for RHEL Resilient Storage - ctdb General Policies および Support Policies for RHEL Resilient Storage - Exporting gfs2 contents via other protocols を参照してください。

### 2.5.6. GFS2 用仮想マシンの設定

仮想マシンで GFS2 ファイルシステムを使用する場合は、キャッシュを強制的に無効にするために各ノードの VM ストレージ設定を適切に設定することが重要です。たとえば、**cache** と **io** に対するこれらの設定を **libvirt** ドメインに含めると、GFS2 が期待どおりに動作します。

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

また、デバイス要素内で **shareable** 属性を設定することができます。これは、デバイスがドメイン間で共有されることを意味します (ハイパーバイザーと OS でサポートされる場合)。**shareable** が使用される場合は、そのデバイスに対して **cache='no'** を使用する必要があります。

## 2.6. ファイルシステムのバックアップ

ファイルシステムのサイズに関わらず、問題が発生したときのために GFS2 ファイルシステムを定期的にバックアップするよう推奨しています。システム管理者の多くは、RAID、マルチパス、ミラーリング、スナップショット、その他の形式での冗長性などで保護されているため安心感を得ることができますが、確実な安全性などはありません。

単一または一式のノードをバックアップするには、通常全ファイルシステムを順番に読み取る必要があるため、バックアップ作成で問題が生じる可能性があります。これを単一のノードから実行した場合には、そのノードは、クラスター内の他のノードがロックの要求を開始するまで、全情報をキャッシュに保持します。クラスターが稼働中にこのようなタイプのバックアッププログラムを実行すると、パフォーマンスに悪影響を及ぼします。

バックアップが完了した後にキャッシュを削除すると、他のノードがクラスターのロック/キャッシュのオーナーシップを再取得する時間が短縮されます。ただし、バックアッププロセスが開始する前に他

のノードがキャッシュしていたデータのキャッシュが停止されることになるため、これでも理想的ではありません。以下のコマンドを実行すると、バックアップの完了後にキャッシュを削除することができます。

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

タスクがノード間で分割されるようクラスター内の各ノードがそれぞれのファイルをバックアップした方が処理が高速になります。これは、ノード固有のディレクトリーで **rsync** コマンドを使用するスクリプトにより実行することができます。

GFS2 バックアップは、SAN 上でハードウェアスナップショットを作成し、そのスナップショットを別のシステムに提供して、そこでバックアップすることにより作成することが推奨されます。スナップショットはクラスター内にないため、バックアップシステムが **-o lockproto=lock_nolock** でスナップショットをマウントする必要があります。

## 2.7. ハードウェアに関する考慮事項

GFS2 ファイルシステムをデプロイする際には、以下のようなハードウェア考慮事項を検討する必要があります。

- より高品質なストレージオプションを使用する

  GFS2 は、iSCSI や Fibre Channel over Ethernet (FCoE) などの低コストの共有ストレージオプションで稼働可能ですが、キャッシュ容量が大きな高品質のストレージを購入するとパフォーマンスが向上します。Red Hat は、ファイバーチャネル相互接続の SAN ストレージ上で品質、サニティー、パフォーマンスに関するほとんどのテストを実行します。原則として、まずテストを実施してからデプロイすることが推奨されます。

- デプロイ前にネットワーク機器をテストする

  高品質の高速ネットワーク機器を使用することにより、クラスターの通信および GFS2 が高速化され、信頼性が向上しますが、最も高額なハードウェアを購入する必要はありません。最も高価なネットワークスイッチの一部で **fcntl** ロック (flock) の受け渡しに使用されるマルチキャストパケットの受け渡しの問題が発生する一方で、安価なコモディティーネットワークスイッチの方が高速かつ信頼度が高い場合があります。完全な実稼働環境にデプロイする前に機器をテストすることが推奨されます。

## 2.8. パフォーマンス上の問題: RED HAT カスタマーポータルで確認する

High Availability Add-On および Red Hat Global File System 2 (GFS2) を使用する Red Hat Enterprise Linux クラスターのデプロイおよびアップグレードの推奨事項については、Red Hat カスタマーポータルの「Red Hat Enterprise Linux Cluster, High Availability, and GFS Deployment Best Practices」(https://access.redhat.com/articles/40051) というタイトルの記事を参照してください。

## 2.9. GFS2 のノードロック機能

GFS2 ファイルシステムでパフォーマンスを最適化するには、操作に関する基本的な理論をある程度理解しておくことが重要となります。単一ノードファイルシステムはキャッシュと共に実装され、頻繁に要求されるデータを使用する場合にディスクへのアクセスの待ち時間をなくすことを目的としています。Linux では、ページキャッシュ (および以前はバッファーキャッシュ) によりこのキャシング機能が提供されます。

GFS2 では各ノードに独自のページキャッシュがあり、オンディスクデータの一部が含まれていることがあります。 GFS2 は *glocks* (ジーロックスと発音) と呼ばれるロック機能のメカニズムを使用して

ノード間のキャッシュの整合性を維持します。glock サブシステムは、*分散型ロックマネージャー* (DLM) を下位の通信層として使用して実装される、キャッシュ管理機能を提供します。

glocks では、inode ごとにキャッシュの保護が提供されるため、ロックは 1 inode あたり 1 つとなり、キャッシング層の制御に使用されます。この glock が共有モード (DML ロックモード: PR) で許可されると、その glock 配下のデータは 1 ノードまたは複数のノードに同時にキャッシュすることができるため、全ノードがそのデータにローカルでアクセスできるようになります。

glock が排他モード (DLM ロックモード: EX) で許可された場合、その glock 配下では単一のノードしかデータをキャッシュすることができなくなります。このモードは、データを修正するすべての操作で使用されます (**write** システムコールなど)。

別のノードが即時に許可できない glock を要求した場合、DLM はそのノードまたは新規の要求をブロックしている glock を現在保持しているノードにメッセージを送り、ロックを削除するように要求します。glock を削除するプロセスは時間がかかる場合があります (大半のファイルシステム操作の基準で判断した場合)。共有 glock を削除する場合に必要となるのは、キャッシュの無効化のみで、比較的短時間で完了し、キャッシュされたデータ量に比例します。

排他的 glock を削除するには、ログをフラッシュして、変更されたデータをディスクに書き戻した後、共有 glock と同様に無効化を行う必要があります。

単一ノードファイルシステムと GFS2 で異なるのは、単一ノードファイルシステムのキャッシュは一つであるのに対して、GFS2 にはノードごとに別個のキャッシュがある点です。いずれの場合も、キャッシュ済みデータへのアクセスの待ち時間は同じ程度ですが、キャッシュされていないデータへのアクセスに関しては、別のノードが以前に同じデータをキャッシュしていた場合、GFS2 の方がはるかに長くなります。

共有 glock を必要とするのは、**read** (バッファ付き)、 **stat**、 **readdir** などの操作のみです。排他的な glock を必要とするのは**write** (バッファ付き)、 **mkdir**、 **rmdir**、**unlink** などの操作のみです。ダイレクト I/O の読み書き操作には、割り当てが行われていないくても DF (deffered) 状態の glock が必要です。または、書き込みに割り合てが必要な場合は、EX (exclusive) 状態の glock が必要です。

この場合、パフォーマンスについて 2 つ主なことを考慮する必要があります。まず、読み込み専用操作は各ノード上で独立して実行されるため、クラスターにわたり非常にうまく並列処理できます。次に、同じ inode へのアクセスを奪い合うノードが複数ある場合は、EX 状態の glock を必要とする操作によってパフォーマンスが低下することがあります。例えば「ファイルシステムのバックアップ」の説明にあるように、ファイルシステムのバックアップを行う場合などには、各ノードのワーキングセットを考慮することがGFS2 ファイルシステムパフォーマンスにおいて重要です。

これに加え、GFS2 では可能な限り、**noatime** と **nodiratime** マウントオプションを使用することが推奨されます。これにより、read が **atime** タイムスタンプを更新するために排他的なロックを必要としなくなります。

ワーキングセットやキャッシング効率を懸念している場合でも、GFS2 では、GFS2 ファイルシステムのパフォーマンスを監視できる Performance Co-Pilot や GFS2 トレースポイントというツールを利用できます。これらはそれぞれ、付録A *Performance Co-Pilot による GFS2 パフォーマンスの分析*、付録 B *GFS2 トレースポイントおよび debugfs glocks ファイル* で説明されています。

**注記**

GFS2 のキャッシング機能の実装方法により、次のいずれかの場合にパフォーマンスが最適となります。

- inode は全ノードにわたって読み取り専用で使用されます。
- inode が単一ノードからのみ書き込みまたは修正される

ファイル作成時および削除時に行われるディレクトリーへのエントリの挿入や削除は、ディレクトリー inode への書き込みと見なされる点に注意してください。

このルールを違反することは可能ですが、比較的頻度が低いことが条件となります。このルールを無視しすぎるとパフォーマンスに深刻な影響を及ぼすことになります。

読み込み/書き込みのマッピングを用いて GFS2 上のファイルを **mmap**() しても、読み取りしか行わない場合は、読み取りのみと見なされますが、GFS 上では書き込みとして見なされるため、**mmap**() I/O では、GFS2 の方が拡張性がはるかに高くなります。

**noatime mount** パラメーターを指定していない場合は、読み取りによって、ファイルのタイムスタンプを更新するための書き込みも発生します。GFS2 を使用する場合、**atime** を特に必要としない限りは、**noatime** を指定してマウントすることを推奨します。

### 2.9.1. POSIX ロックの問題

POSIX ロックを使用する場合、以下の点を考慮する必要があります。

- フロックを使用すると、POSIX ロックを使用するより処理が早い
- クラスター環境では指定のプロセス ID が別のノード用である場合もあるため、GFS2 で POSIX ロックを使用するプログラムは、**GETLK** 関数の使用を避けるべきです。

### 2.9.2. GFS2 によるパフォーマンスチューニング

通常は、問題を引き起こすアプリケーションのデータ格納方法を変更すると、パフォーマンスを大幅に向上させることができます。

問題を引き起こすアプリケーションの典型的な例が Email サーバーです。各ユーザー用ファイルを含むスプールディレクトリー (**mbox**)、または各メッセージ用のファイルを格納する各ユーザー用のディレクトリー (**maildir**) などで構成されているのが一般的です。IMAP 経由で要求が到着した場合、特定のノードに対するアフィニティーを各ユーザーに提供するのが理想的な設定となります。 これにより、Email メッセージの表示と削除の要求は、その単一ノードのキャッシュから提供される傾向になります。そのノードに障害が発生した場合は、当然、別のノードでセッションを再起動させることができます。

メールが SMTP 経由で到着した場合も、デフォルトで特定のユーザーのメールが特定のノードに渡されるようノードを個別に設定することが可能です。デフォルトのノードが稼働していない場合、メッセージは受信ノードによりユーザーのメールスプールに直接保存されます。これも、通常の場合は特定のファイルセットが単一のノードにキャッシュされ、そのノードの障害時には直接アクセスができるようにすることを目的として設計されています。

この設定により GFS2 のページキャッシュを最大限に活用し、**imap** または **smtp** にかかわらず、アプリケーションに対して障害の発生を透過的にすることができます。

バックアップは、難しいことが多いもう一つの領域です。可能であれば、特定の inode セットをキャッシュしているノードから直接、各ノードの作業セットをバックアップしておくことを強く推奨します。

定時に実行するバックアップスクリプトを使用していて、それが GFS2 で実行しているアプリケーションの応答時間の急増と同時に発生している場合には、クラスターがページキャッシュを効率的に使用していない可能性が高くなります。

当然ながら、バックアップを実行するためにアプリケーションを停止することができるような優位な立場にある場合は問題はありません。しかし、バックアップが 1 つのノードだけで実行される場合は、そのバックアップが完了するとファイルシステムの大部分がそのノードにキャッシュされ、他のノードからアクセスする際のパフォーマンスが低下します。この問題は、次のコマンドでバックアップ完了後にバックアップノードにある VFS ページキャッシュを削除することにより、ある程度軽減することができます。

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

ただし、この方法は、各ノードの作業セットが、大半は読み取り専用としてクラスター全体で共有される、もしくは大部分が単一のノードからアクセスされるかのいずれかとなるように注意するほどは、よい解決策ではありません。

### 2.9.3. GFS2 ロックダンプを使用した GFS2 パフォーマンスのトラブルシューティング

GFS2 キャシングの使用効率が悪いためにクラスターのパフォーマンスが影響を受けている場合、I/O の待機時間が長く延長される可能性があります。GFS2 のロックダンプ情報を利用すると、この問題の原因を究明することができます。

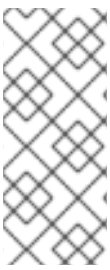
本セクションには、GFS2 ロックダンプの概要を記載しています。GFS2 ロックダンプについての詳しい説明は、付録B *GFS2 トレースポイントおよび debugfs glocks ファイル* を参照してください。

GFS2 ロックダンプ情報は **debugfs** ファイルから収集することができます。このファイルのパス名は以下のとおりです (**debugfs** が **/sys/kernel/debug/** にマウントされていることが前提です)。

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

ファイルの内容は、一連の行から構成されています。G: で始まる行は、それぞれひとつの glock を表し、その下の 1 文字分字下げしてある行はファイル内の直前の行の glock に関連する情報アイテムを表しています。

**debugfs** ファイルの最適な使用方法は、アプリケーションに問題の発生中に **cat** コマンドを使ってファイルの全内容のコピーをとり (RAM が大容量で、かつキャッシュされた inode が多数ある場合には長時間がかかる可能性があります)、後日に結果データを確認する方法です。



**注記**

**debugfs** ファイルのコピーを 2 つ作成すると、役に立つことがあります (2 つ目のコピーを最初のコピーの数秒後または数分後に作成します)。同じ glock 番号に関連する 2 つのトレースでホルダー情報を比較すると、ワークロードが処理されているか (単に処理速度が遅い)、または処理が停止しているか (この場合は必ずバグであり、Red Hat サポートに直ちに報告してください) がわかります。

**debugfs** ファイルの H: (ホルダー) で始まる行は、承認済みまたは承認待機中のロック要求を表します。ホルダー行のフラグフィールド f: には、待機中の要求を示す「W」フラグ、または承認済み要求を示す「H」フラグが表示されます。待機中の要求が多数ある glocks は特定の競合が発生している可能性が高くなります。

各 glock フラグの意味は 表2.1「glock フラグ」 に、また各 glock ホルダーフラグの意味は 表2.2「Glock ホルダーフラグ」 にまとめています。

**表2.1 glock フラグ**

| フラグ | 名前 | 意味 |
|---|---|---|
| b | Blocking | ロックされたフラグが設定されている場合に有効であり、DLM から要求された操作がブロックされる可能性があることを示します。このフラグは、降格の操作と「try」ロックに対して消去されます。このフラグの目的は、ロックを降格するために他のノードでかかる時間とは別の DLM 応答時間に関する統計情報の収集を許可することです。 |
| d | Pending demote | 遅延された (リモートの) 降格要求 |
| D | Demote | 降格要求 (ローカルまたはリモート) |
| f | Log flush | この glock を解放する前にログをコミットする必要があります |
| F | Frozen | リモートノードからの返信が無視されます (復元は進行中)。このフラグは、異なるメカニズムを使用するファイルシステムのフリーズとは関係がなく、復元中にしか使用されません。 |
| i | Invalidate in progress | この glock のページの無効化が進行中です |
| I | Initial | DLM がこの glock と関連付けられる場合に指定します |
| l | Locked | glock は状態の変更中です |
| L | LRU | glock が LRU リストにあるときに設定されます |
| o | オブジェクト | glock がオブジェクト (タイプ 2 glock の場合は inode、タイプ 3 glock の場合はリソースグループ) に関連付けられた場合に設定されます。 |
| p | Demote in progress | glock は降格要求に応答中です |
| q | キュー待ち | glock に対してホルダーがキューに格納された場合に設定され、glock が保持された場合に消去されます (ただし、残りのホルダーはありません)。glock の最小保持時間を計算するアルゴリズムの一部として使用されます。 |
| r | Reply pending | リモートノードから受信した返信の処理の待機中です |
| y | Dirty | この glock を解放する前にデータをディスクにフラッシュする必要があります |

**表2.2 Glock ホルダーフラグ**

| フラグ | 名前 | 意味 |
|---|---|---|
| a | Async | glock の結果を待ちません (結果を後でポールします) |
| A | Any | 互換性のあるロックモードはすべて受け入れ可能です |
| c | No cache | ロック解除時に DLM ロックを即時に降格します |
| e | No expire | 後続のロック取り消し要求を無視します |
| E | exact | 完全一致するロックモードでなければなりません |
| F | First | ホルダーがこのロックに最初に許可される場合に指定します |
| H | Holder | 要求したロックが許可されたことを示します |
| p | Priority | キューの先頭にある待機ホルダー |
| t | Try | 「try」ロックです |
| T | Try 1CB | コールバックを送信する「try」ロックです |
| W | Wait | 要求完了の待機中にセットされます |

問題の原因となっている glock を特定したら、次はそれに関連している inode を見つけます。これは、glock 番号 (G: 行にある n:) に示されています。 *type*/*number* の形式で記載されており、*type* が 2 の場合は glock は inode glock で *number* は inode 番号になります。 **find -inum *number*** のコマンドを実行すると、inode をトラッキングすることができます。glocks ファイルにある 16 進形式から 10 進形式に変換した inode 番号が *number* になります。

**警告**

ロックの競合が発生しているときにファイルシステムで **find** を実行すると、事態が悪化する可能性が高くなります。競合している inode を探している場合は、まずアプリケーションを停止してから **find** を実行することが推奨されます。

表2.3「glock のタイプ」 には、異なる glock のタイプの意味をまとめています。

**表2.3 glock のタイプ**

| タイプ番号 | ロックタイプ | 用途 |
|---|---|---|
| 1 | Trans | トランザクションのロック |

| タイプ番号 | ロックタイプ | 用途 |
|---|---|---|
| 2 | Inode | Inode のメタデータとデータ |
| 3 | Rgrp | リソースグループのメタデータ |
| 4 | Meta | スーパーブロック |
| 5 | Iopen | 最後に inode をクローズしたプロセスの検出 |
| 6 | Flock | **flock**(2) syscall |
| 8 | Quota | クォータ操作 |
| 9 | Journal | ジャーナルミューテックス |

特定された glock が異なるタイプだった場合に、最も可能性が高いのはタイプ 3: (リソースグループ) です。通常の負荷であるときに別のタイプの glock を待機しているプロセスが大量にある場合は、Red Hat サポートにご報告ください。

リソースグループロックのキューに格納された待機要求が複数ある場合は、この状況に対して複数の理由があることが考えられます。その 1 つの理由は、ファイルシステム内のリソースグループの数よりもノード数が多いことです。別の理由は、ファイルシステムがほぼ完全に満杯であることです (平均すると空きブロックの検索に時間がかかる)。いずれの状況も、ストレージを追加し **gfs2_grow** コマンドを使用してファイルシステムを拡張することで改善することができます。

# 第3章 GFS2 の管理

この章では、GFS2 を管理するためのタスクとコマンドについて説明します。この章は以下のような セクションで構成されます。

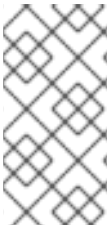
- 「GFS2 ファイルシステムの作成」
- 「GFS2 ファイルシステムマウント」
- 「GFS2 ファイルシステムのアンマウント」
- 「GFS2 のクォータ管理」
- 「GFS2 ファイルシステムの拡張」
- 「GFS2 ファイルシステムへジャーナルの追加」
- 「データジャーナリング」
- 「**atime** 更新の設定」
- 「GFS2 ファイルシステム上の動作の一時停止」
- 「GFS2 ファイルシステムの修復」
- 「GFS2 withdraw 機能」

## 3.1. GFS2 ファイルシステムの作成

**mkfs.gfs2** コマンドを使用して GFS2 ファイルシステムを作成します。 また **mkfs** コマンドに **-t gfs2** オプションを指定して使用することもできます。 ファイルシステムは起動中の LVM ボリューム上で作成されます。 **mkfs.gfs2** コマンドを実行するには 以下の情報が必要になります。

- プロトコル/モジュールのロック名（クラスター用の lock protocol は **lock_dlm**）
- クラスター名 (***LockTableName*** パラメーターを指定する際に必要)
- ジャーナルの数（ファイルシステムをマウントするノード 1 つにつき、ジャーナルが 1 つ必要）

GFS2 ファイルシステムを作成する場合は、直接 **mkfs.gfs2** コマンドを使用できます。または、**mkfs** コマンドに ***-t*** パラメーターを付けてタイプ **gfs2** のファイルシステムを指定し、その後に GFS2 ファイルシステムのオプションを指定できます。



**注記**

**mkfs.gfs2** コマンドで GFS2 ファイルシステムを作成した後には、そのファイルシステムのサイズは縮小できません。 ただし、 「GFS2 ファイルシステムの拡張」 に記載されている通り、**gfs2_grow** コマンドを使って既存のファイルシステムのサイズを拡大することは可能です。

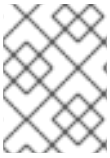
### 使用方法

クラスター化された GFS2 ファイルシステムを作成する場合、以下の形式のいずれかを使用できます:

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

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

ローカルの GFS2 ファイルシステムを作成する場合、以下の形式のいずれかを使用できます:



**注記**

Red Hat Enterprise Linux 6 では、Red Hat は GFS2 のシングルノードファイルシステムとしての使用をサポートしません。

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

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

**警告**

***LockProtoName*** と ***LockTableName*** のパラメーターの使用に精通していることを確認してください。***LockProtoName*** と ***LockTableName*** のパラメーターを不適切に使用をすると、ファイルシステムまたはロックスペースが破損する可能性があります。

***LockProtoName***

使用するロックプロトコルの名前を指定します。クラスター用のロックプロトコルは **lock_dlm** です。

***LockTableName***

このパラメーターはクラスター設定の GFS2 ファイルシステム用に指定されます。これは、次にあげるように、コロンで区切られた 2 つの部分 (空白なし) で構成されます: ***ClusterName:FSName***

- ***ClusterName***: クラスター名。このクラスターに GFS2 ファイルシステムが作成されます。
- ***FSName***: ファイルシステムの名前。1 文字から 16 文字までの長さに指定することができます。この名前は、クラスター上のすべての **lock_dlm** ファイルシステムと各ローカルノード上のすべてのファイルシステム（**lock_dlm** および **lock_nolock**）にわたって一意である必要があります。

***Number***

**mkfs.gfs2** コマンドで作成するジャーナルの数を指定します。ファイルシステムをマウントするノード毎に 1 つのジャーナルが必要です。「GFS2 ファイルシステムへジャーナルの追加」 で説明しているように、GFS2 ファイルシステムではファイルシステムを拡張することなくジャーナルを後で追加することができます。

***BlockDevice***

論理ボリュームまたは物理ボリュームを指定します。

### 例

この例では、**lock_dlm** は ファイルシステムが使用するロッキングプロトコルです (ファイルシステムはクラスター化ファイルシステム)。クラスター名は **alpha** であり、ファイルシステム名は **mydata1** です。このファイルシステムは 8 つのジャーナルを含み、**/dev/vg01/lvol0** 上に作成されます。

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

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

以下の例では、２つ目の **lock_dlm** ファイルシステムが作成されて、それがクラスター **alpha** 内で使用できます。ファイルシステム名は **mydata2** です。このファイルシステムには ８つのジャーナルが 含まれており、**/dev/vg01/lvol1** 上に作成されます。

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

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

### 全オプション

表3.1「コマンドオプション: **mkfs.gfs2**」 では **mkfs.gfs2** コマンドのオプションを説明しています (フラグとパラメーター)。

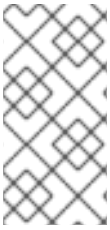
**表3.1 コマンドオプション: mkfs.gfs2**

| フラグ | パラメーター | 説明 |
|---|---|---|
| **-c** | ***Megabytes*** | 各ジャーナルのクォータ変更ファイルの初期サイズを ***Megabytes*** に設定します。 |
| **-D** | | デバッグの出力を有効にします。 |
| **-h** | | ヘルプ。使用可能なオプションを表示します。 |
| **-J** | ***Megabytes*** | ジャーナルのサイズをメガバイトで指定します。デフォルトのジャーナルサイズは 128 メガバイトです。最低サイズは 8 メガバイトです。ジャーナルを大きくするとパフォーマンスが向上しますが、小さいジャーナルよりもメモリーを多く消費します。 |
| **-j** | ***Number*** | **mkfs.gfs2** コマンドで作成されるジャーナルの数を指定します。ファイルシステムをマウントするノード毎に 1 つのジャーナルが必要になります。このオプションが指定されていない場合、作成されるジャーナルは 1 つとなります。GFS2 ファイルシステムでは ファイルシステムを拡張することなく、ジャーナルを後で追加することができます。 |
| **-O** | | **mkfs.gfs2** コマンドでファイルシステムへの書き込み前に確認プロンプトを表示しないようにします。 |

| フラグ | パラメーター | 説明 |
| --- | --- | --- |
| **-p** | ***LockProtoName*** | 使用するロックプロトコルの名前を指定します。認められているロックプロトコルには、以下のようなプロトコルがあります。<br>**lock_dlm** – 標準のロックモジュール。クラスター化ファイルシステムに必要です。<br>**lock_nolock** – GFS2 がローカルファイルシステムとして機能している場合に使用します (1 ノードのみ)。 |
| **-q** | | Quiet モード。何も表示しません。 |
| **-r** | ***Megabytes*** | リソースグループのサイズをメガバイト単位で指定します。リソースグループの最低サイズは 32 メガバイトです。リソースグループの最大サイズは 2048 メガバイトです。リソースグループのサイズが大きいと、非常に大規模なファイルシステムのパフォーマンスが向上することがあります。リソースグループのサイズを指定しない場合は、**mkfs.gfs2** がファイルシステムのサイズに基いてリソースグループのサイズを選択します。平均的なサイズのファイルシステムでは 256 メガバイトのリソースグループとなり、大きなファイルシステムではパフォーマンスを向上させるためにさらに大きなリソースグループサイズとなります。 |
| **-t** | ***LockTableName*** | **lock_dlm** プロトコルを使用している時に、ロックテーブルのフィールドを指定する一意識別子。**lock_nolock** プロトコルは、このパラメーターを使用しません。<br>このパラメーターは、次のようにコロンで区切られた（空白なし）2 つの部分で構成されます:<br>***ClusterName:FSName***.<br>***ClusterName*** は、GFS2 ファイルシステムが作成されているクラスターの名前です。このクラスターのメンバーだけが、このファイルシステムを使用できます。<br>***FSName***, ファイルシステム名です。長さは 1文字から 16文字までで、名前はクラスター内のすべてのファイルシステムで一意でなければなりません。 |
| **-u** | ***Megabytes*** | 各ジャーナルのリンクのないタグファイルの初期サイズを指定します。 |

| フラグ | パラメーター | 説明 |
| --- | --- | --- |
| **-V** | | コマンドのバージョン情報を表示します。 |

## 3.2. GFS2 ファイルシステムマウント



**注記**

「GFS2 ファイルシステムのアンマウント」 で説明されているように、システムのシャットダウン時に問題が発生する可能性があるため、mount コマンドでファイルシステムを手動でマウントするのではなく、常に Pacemaker を使用して運用環境で GFS2 ファイルシステムを管理する必要があります。

GFS2 ファイルシステムをマウントできるようにするには、そのファイルシステムが存在しており (「GFS2 ファイルシステムの作成」 参照)、そのファイルシステムが存在するボリュームがアクティブな状態で、かつクラスタリングシステムとロッキングシステムのサポートが起動している必要があります (**Red Hat Cluster の設定と管理** を参照)。これらの要件を満たした上で、Linux ファイルシステムと同様に GFS2 ファイルシステムをマウントすることができます。

ファイル ACL を操作するには、**-o acl** マウントオプションを指定して、ファイルシステムをマウントする必要があります。**-o acl** マウントオプションを指定せずにファイルシステムをマウントすると、ユーザーは (**getfacl** で) ACL を表示できますが、(**setfacl**で) それらを設定することができません。

### 使用方法

**ACL 操作なしのマウント**

```
mount BlockDevice MountPoint
```

**ACL 操作が可能なマウント**

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

**-o acl**
ファイル ACL の操作を可能にする GFS2 固有のオプション 。

***BlockDevice***
GFS2 ファイルシステムが常駐するブロックデバイスを指定します。

***MountPoint***
GFS2 ファイルシステムがマウントされるディレクトリーを指定します。

### 例

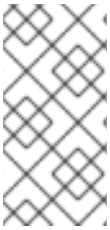
この例では、**/dev/vg01/lvol0** の GFS2 ファイルシステムは **/mygfs2** ディレクトリーにマウントされます。

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

完全な使用法

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

**-o option** 引数は GFS2 固有のオプション (表3.2「GFS2 固有のマウントオプション」 を参照)、使用可能な標準の Linux **mount -o** オプション、あるいはその両方の組み合わせで構成されます。複数の ***option*** パラメーターは、空白を 入れずにコンマで区切ります。



**注記**

**mount** コマンドは Linux のシステムコマンドです。このセクションに 説明してある GFS2 固有のオプションの他にも、標準の **mount** コマンドの オプション（たとえば、**-r**）を使用することができます。Linux **mount** コマンドのオプションに関する情報は、Linux **mount** man ページでご覧ください。

表3.2「GFS2 固有のマウントオプション」 ではマウント時に GFS2 へ渡すことのできる使用可能な GFS2 固有の **-o option** 値を説明します。

**注記**

この表には、ローカルのファイルシステムで使用するオプションの説明のみが記載されています。ただし、Red Hat Enterprise Linux 6 リリースでは、Red Hat はシングルノードファイルシステムとしての GFS2 の使用をサポートしないことに注意してください。Red Hat は、シングルノード GFS2 ファイルシステムでクラスターファイルシステムのスナップショットのマウント (たとえば、バックアップ目的) を引き続きサポートします。

**表3.2 GFS2 固有のマウントオプション**

| オプション | 説明 |
|---|---|
| **acl** | ファイル ACL の操作を可能にします。 **acl** マウントオプションを指定せずにファイルシステムをマウントした場合、ユーザーは ACL の表示 (**getfacl**) はできますが、設定 (**setfacl**) はできません。 |
| **data=[ordered\|writeback]** | **data=ordered** を設定すると、トランザクションにより変更されたユーザーデータは、トランザクションがディスクにコミットされる前にディスクにフラッシュされます。これは、クラッシュの後にファイル内で初期化されていないブロックがユーザーに表示されるのを防ぎます。**data=writeback** モードが設定されている場合は、ユーザーデータはダーティとなった後でも、随時ディスクに書き込まれます。これは、**ordered** モードと同じ一貫性保証は提供しませんが、ワークロードによっては若干速くなります。デフォルト値は **ordered** モードです。 |

| オプション | 説明 |
| --- | --- |
| **ignore_local_fs**<br><br>**注意:** このオプションは、GFS2 ファイルシステムが共有されている場合には **使用すべきではありません**。 | GFS2 がファイルシステムをマルチホストファイルシステムとして扱うように強制します。デフォルトでは、**lock_nolock** を使用すると **localflocks** のフラグが自動的に有効になります。 |
| **localflocks**<br><br>**注意:** このオプションは、GFS2 ファイルシステムが共有されている場合には使用すべきではありません。 | VFS (virtual file system) レイヤーですべての flock と fcntl を実行するよう、GFS2 に指示します。**localflocks** フラグは **lock_nolock** により自動的に有効になります。 |
| **lockproto=*LockModuleName*** | ユーザーが、ファイルシステムで使用するロックプロトコルを指定できるようにします。***LockModuleName*** が指定されていない場合は、ファイルシステムのスーパーブロックからロックプロトコル名が読み込まれます。 |
| **locktable=*LockTableName*** | ユーザーがファイルシステムで使用するロックテーブルを指定できるようにします。 |
| **quota=[off/account/on]** | ファイルシステムのクォータのオン/オフを切り替えます。**account** の状態となるようにクォータを設定すると、UID/GID 毎の使用状況の統計はファイルシステムにより正しく維持され、上限と警告の値は無視されます。デフォルト値は **off** です。 |
| **errors=panic\|withdraw** | **errors=panic** を指定すると、ファイルシステムのエラーによりカーネルパニックが起こります。**errors=withdraw** を指定すると (これはデフォルトの動作)、ファイルシステムのエラーにより、システムがファイルシステムから切り離され、次回の再起動までシステムにアクセスできなくなります。場合によっては、システムが稼働し続けることがあります。 |
| **discard/nodiscard** | これにより、GFS2 は、解放されたブロックに対する「破棄」 I/O 要求を生成します。これらのブロックは、適切なハードウェアがシンプロビジョニングおよび同様のスキームを実装するのに使用可能です。 |

| オプション | 説明 |
| --- | --- |
| **barrier/nobarrier** | これにより、GFS2 は、ジャーナルをフラッシュする際に、I/O バリアを送信します。デフォルト値は **on** です。このオプションは配下のデバイスが I/O バリアに対応していない場合は自動的に **off** になります。書き込みキャッシュの内容の消失が不可能となるようにブロックデバイスが設計されていない限り (UPS 上にある場合や書き込みキャッシュがない場合)、GFS2 では I/O バリアを常に使用することを強く推奨します。 |
| **quota_quantum=*secs*** | クォータ情報に関する変更がクォータファイルに書き込まれるまでに一つのノードに留めることができる秒数を設定します。パラメーターの設定には、この方法が推奨されます。値はゼロより大きい整数になります。デフォルトは 60 秒です。これより短く設定すると遅延クォータ情報の更新速度が速くなるため、そのクォータを越えてしまう可能性が少なくなります。長く設定すると、クォータに伴うファイルシステムの動作速度が高速化され、効率性が向上します。 |
| **statfs_quantum=*secs*** | **statfs** の遅いバージョンを設定する場合は **statfs_quantum** を 0 に設定するのが推奨の方法です。デフォルト値は 30 秒で、**statfs** の変更が **statfs** のマスターファイルに同期されるまでの最大時間を設定します。速度を高くして **statfs** の正確性を低くしたり、速度を低くして正確性を高めたりするなどの調整が可能です。このオプションを 0 に設定すると **statfs** は常に true の値を報告するようになります。 |
| **statfs_percent=*value*** | 有効期間が切れていない場合でも、 **statfs** のマスターファイルに戻って同期するまでにローカルベースで **statfs** 情報の最大変更率の上限を提供します。 **statfs_quantum** の設定が 0 の場合はこの設定は無視されます。 |

## 3.3. GFS2 ファイルシステムのアンマウント

Pacemaker により自動的にマウントされたのではなく手動でマウントされた GFS2 ファイルシステムは、システムのシャットダウン時にファイルシステムがアンマウントされたときに、システムにより認識されなくなります。結果として、GFS2 スクリプトは GFS2 ファイルシステムをアンマウントしません。GFS2 シャットダウンスクリプトの実行後に、標準のシャットダウンプロセスにより、クラスターインフラストラクチャーを含む残りのユーザープロセスがすべて強制終了され、ファイルシステムのアンマウントが試行されます。このアンマウントは、クラスターインフラストラクチャーがないと失敗し、システムがハングします。

GFS2 ファイルシステムのアンマウント時にシステムをハングさせないようにするには、次のいずれかを行ってください。

- GFS2 ファイルシステムを管理するには、必ず Pacemaker を使用してください。Pacemaker クラスターでの GFS2 の設定については、*5章 クラスターでの GFS2 ファイルシステムの設定*を参照してください。

- GFS2 ファイルシステムを **mount** コマンドを使って手作業でマウントした場合はシャットダウンまたは再起動を行う前に必ず **umount** コマンドを使って手作業でファイルシステムをアンマウントします。

このような状況下で、ファイルシステムがシステムのシャットダウン時にアンマウントするとハングしてしまう場合は、ハードウェアの再起動を行ってください。ファイルシステムはシャットダウンプロセスの早期に同期されるため、データが消失する可能性は少くなります。

GFS2 ファイルシステムはその他の Linux ファイルシステムと同じ方法でアンマウントできます。**umount** コマンドを使用します。

**注記**

**umount** コマンドは Linux のシステムコマンドです。 このコマンドに関する情報は、Linux **umount** コマンドの man ページをご覧ください。

使用方法

```
umount MountPoint
```

***MountPoint***

GFS2 ファイルシステムが現在マウントされているディレクトリーを指定します。

## 3.4. GFS2 のクォータ管理

ファイルシステムのクォータは、ユーザーまたはグループが使用できるファイルシステム容量のサイズを制限するために使用されます。クォータが設定されるまで、ユーザーまたはグループにはクォータ制限がありません。**quota=on** または **quota=account** のオプションで GFS2 ファイルシステムがマウントされると、制限が設定されていない場合でも、GFS2 は各ユーザーおよびグループが使用する領域を追跡します。GFS2 は、システムがクラッシュしてもクォータの使用状況を再構築する必要がないように、トランザクション形式でクォータ情報を更新します。

GFS2 ノードは、パフォーマンスの低下を防ぐために、クォータファイルへ更新の同期は定期的にしか行いません。ファジークォータアカウンティングでは、ユーザーやグループは設定上限を若干超過することができます。GFS2 はこれを最低限に抑えるために、クォータのハードリミットに近づくと、動的に同期の間隔を短縮します。

**注記**

GFS2 は標準の Linux クォータ機能をサポートします。この機能を使用するには、**quota** RPM をインストールする必要があります。これは、GFS2 で推奨されるクォータの管理方法であり、クォータを使用するすべての GFS2 デプロイメントで使用する必要があります。このセクションでは、これらの機能を使用した GFS2 クォータ管理について説明します。

### 3.4.1. ディスククォータの設定

ディスククォータを実装するには、以下の手順を用います。

1. 強制またはアカウンティングモードでクォータを設定します。
2. 現在のブロック使用状況の情報の入ったクォータデータベースファイルを初期化します。

3. クォータポリシーを割り当てます (アカウンティングモードでは、これらのポリシーは適用されません)。

これらの各ステップは、以下のセクションで詳しく解説しています。

#### 3.4.1.1. 強制またはアカウンティングモードでのクォータの設定

GFS2 ファイルシステムでは、デフォルトでクォータは無効になっています。ファイルシステムのクォータを有効にするには、**quota=on** オプションを指定してファイルシステムをマウントします。

制限および警告値を適用せずに、ディスク使用状況を追跡し、各ユーザーおよびグループのクォータアカウントを維持することができます。これを行うには、**quota=account** オプションを指定してファイルシステムをマウントします。

クォータが有効なファイルシステムをマウントするには、クラスターで GFS2 ファイルシステムリソースを作成するときに **options** 引数として **quota=on** を指定します。たとえば、以下のコマンドは、作成される GFS2 **Filesystem** リソースがクォータが有効な状態でマウントされることを示しています。

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

Pacemaker クラスターでの GFS2 ファイルシステムの設定については、*5章 クラスターでの GFS2 ファイルシステムの設定*を参照してください。

クォータの制限が適用されない場合でもクォータアカウントが維持された状態でファイルシステムをマウントするには、クラスターで GFS2 ファイルシステムリソースを作成するときに **options** 引数として **quota=account** を指定します。

クォータが無効なファイルシステムをマウントするには、クラスターで GFS2 ファイルシステムリソースを作成するときに **options** 引数として **quota=off** を指定します。

#### 3.4.1.2. クォータデータベースファイルの作成

クォータが有効化された各ファイルシステムがマウントされた後、システムはディスククォータを使用して作業することが可能となります。ただし、ファイルシステム自体は、クォータをサポートする準備は整っていない状態となります。次のステップとして、**quotacheck** コマンドを実行します。

**quotacheck** コマンドは、クォータが有効なファイルシステムを検証し、現在のディスク使用状況のテーブルをファイルシステムごとに構築します。このテーブルは、ディスク使用状況のオペレーティングシステム用コピーを更新するのに使用されます。また、ファイルシステムのディスククォータが更新されます。

クォータファイルをファイルシステム上に作成するには、**quotacheck** コマンドで **-u** および **-g** のオプションを使用してください。ユーザーおよびグループのクォータを初期化するには、これらの両オプションを指定する必要があります。たとえば、**/home** ファイルシステムにクォータが有効化されている場合、**/home** ディレクトリーにファイルを作成します。

```
quotacheck -ug /home
```

#### 3.4.1.3. ユーザーごとのクォータ割り当て

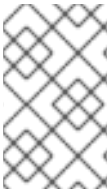
最後のステップは、**edquota** コマンドを使用したディスククォータ割り当てです。ファイルシステムをアカウンティングモードでマウントしている場合には (**quota=account** オプションを指定)、クォータは適用されない点に注意してください。

ユーザー用のクォータを設定するには、シェルプロンプトで root として以下のコマンドを実行してください。

```
# edquota username
```

クォータを必要とするユーザーごとにこの手順を実行します。たとえば、クォータが **/home** (以下の例では **/dev/VolGroup00/LogVol02**) パーティションに対して有効であり、コマンド **edquota testuser** を実行すると、システムでデフォルトとして設定されたエディターで以下のような出力が表示されます。

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



**注記**

**edquota** は、**EDITOR** 環境変数で定義されているテキストエディターを使用します。このエディターを変更するには、~/**.bash_profile** ファイルの **EDITOR** 環境変数を任意のエディターのフルパスに設定してください。

第 1 列は、クォータが有効化されているファイルシステムの名前です。第 2 列は、ユーザーが現在使用中のブロック数です。その後の 2 列は、ファイルシステム上のユーザーに対するソフトおよびハードのブロックリミットを設定するのに使用されます。

ソフトブロックリミットは、使用可能な最大ディスク容量を定義します。

ハードブロックリミットは、ユーザーまたはグループが使用可能な絶対最大ディスク容量です。この上限に達すると、それ以上のディスク容量を使用できなくなります。

GFS2 ファイルシステムは、inode 用のクォータは維持管理しないため、これらの列は GFS2 ファイルシステムには該当せず、空欄となります。

いずれかの値が 0 に設定されている場合、そのリミットは設定されていないことになります。これはテキストエディターで変更できます。例:

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

ユーザーのクォータが設定されていることを確認するには、以下のコマンドを使用します。

**quota testuser**

#### 3.4.1.4. グループごとのクォータ割り当て

クォータは、グループごとに割り当てることも可能です。ファイルシステムをアカウンティングモードでマウントしている場合は (**quota=account** オプションを指定)、クォータは適用されない点に注意してください。

**devel** グループにグループクォータを設定するには (グループクォータを設定する前にグループが存在している必要があります)、以下のコマンドを使用します。

```
edquota -g devel
```

このコマンドにより、グループの既存クォータがテキストエディターに表示されます。

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

GFS2 ファイルシステムは、inode 用のクォータは維持管理しないため、これらの列は GFS2 ファイルシステムには該当せず、空欄となります。上限を変更してファイルを保存します。

グループクォータが設定されていることを確認するには、以下のコマンドを使用します。

```
$ quota -g devel
```

### 3.4.2. ディスククォータの管理

クォータが実装されている場合には、若干の保守が必要となります – 大半は、クォータの超過監視および精度確認という形となります。

当然ながら、ユーザーが繰り返しクォータを超過したり、常にソフトリミットに達している場合には、ユーザーのタイプや、ユーザーの作業にディスク容量が及ぼす影響の度合に応じて、システム管理者には 2 つの選択肢があります。管理者は、ユーザーが使用するディスク領域を節約する方法をわかるようにするか、ユーザーのディスククォータを拡大するかのいずれかを行うことができます。

ディスク使用状況のレポートを作成するには、**repquota** ユーティリティーを使用します。たとえば、コマンド **repquota /home** により、以下のような出力が表示されます。

```
*** 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
```

クォータが有効化されたすべてのファイルシステム (オプション **-a**) のディスク使用状況レポートを表示するには、以下のコマンドを使用します。

```
# repquota -a
```

レポートは読みやすいですが、いくつか説明しておくべき点があります。各ユーザーの後ろに表示される **--** により、ブロック制限を超過しているかどうかが即時に判断できます。ブロックソフト制限を超過している場合は、出力で、最初の **-** の場所に **+** が表示されます。2 番目の **-** は、inode 制限を示しますが、GFS2 ファイルシステムは inode 制限をサポートしないため、この文字は **-** のままになります。GFS2 ファイルシステムは猶予期間をサポートしないため、**grace** の列は空欄のままになります。

NFS では、配下のファイルシステムにかかわらず、**repquota** コマンドはサポートされていない点に注意してください。

### 3.4.3. クォータの精度維持

しばらくクォータが無効な状態で稼働した後にファイルシステムでクォータを有効にする場合は、**quotacheck** コマンドを実行してクォータファイルを作成、確認、および修復する必要があります。また、クォータファイルが正確でないと思われる場合 (この問題は、システムのクラッシュ後に

ファイルシステムが正常にアンマントされていない場合に発生することがあります) は、**quotacheck** を実行することができます。

**quotacheck** コマンドについての詳しい情報は、**quotacheck** の man ページを参照してください。

**注記**

計算されるクォータ値は、ディスクアクティビティーによって影響を受ける可能性があるため、**quotacheck** は全ノード上でファイルシステムが比較的アイドル状態の時に実行してください。

### 3.4.4. quotasync コマンドを使用したクォータの同期

GFS2 は、ディスク上の独自の内部ファイル内にすべてのクォータ情報を格納します。GFS2 ノードは、ファイルシステムの書き込みのたびにこのクォータファイルを更新するのではなく、デフォルトで毎 60 秒ごとにクォータファイルを更新するようになっています。これは、クォータファイルに書き込むノード間の競合によるパフォーマンス低下を回避するために不可欠です。

ユーザーまたはグループのクォータが上限に近づくと、GFS2 は、クォータファイルの更新間隔を動的に短縮して上限の超過を防止します。クォータ同期の通常の間隔は、調整可能なパラメーター **quota_quantum** です。表3.2「GFS2 固有のマウントオプション」 で説明しているように、このパラメーターは **quota_quantum=** マウントオプションを使用してデフォルト値の 60 秒から変更することができます。**quota_quantum** パラメーターは、各ノードで、ファイルシステムがマウントされるたびに設定する必要があります。**quota_quantum** への変更は、アンマウント後には永続されません。**quota_quantum** 値は **mount -o remount** を使用して更新することができます。

**gfs2_quota sync** コマンドを使用すると、GFS2 によって実行される自動的な更新と更新の間にクォータ情報をノードからオンディスククォータファイルに同期することができます。

#### 使用方法

**クォータ情報の同期**

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

***u***

ユーザーのクォータファイルを同期します。

***g***

グループのクォータファイルを同期します。

***a***

現在クォータが有効化された、同期に対応する全ファイルシステムを同期します。-a を使用していない場合には、ファイルシステムのマウントポイントを指定する必要があります。

***mntpnt***

設定が適用される GFS2 ファイルシステムを指定します。

**同期間隔の調整**

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

***MountPoint***

設定が適用される GFS2 ファイルシステムを指定します。

***secs***

GFS2 による定期的なクォータファイル同期の間隔を新たに指定します。値を小さくすると、競合が増え、パフォーマンスが低下する場合があります。

**例**

以下の例では、コマンドが実行されるノードのキャッシュ済みダーティクォータすべてをファイルシステム **/mnt/mygfs2** のクォータファイルに同期します。

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

以下の例では、ファイルシステム **/mnt/mygfs2** を論理ボリューム **/dev/volgroup/logical_volume** に再マウントする時に、そのファイルシステムのクォータファイル定期更新間隔をデフォルト値から 1 時間 (3600 秒) に変更します。

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

### 3.4.5. リファレンス

ディスククォータに関する更なる情報は、以下にあげるコマンドの **man** ページを参照してください。

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

## 3.5. GFS2 ファイルシステムの拡張

**gfs2_grow** コマンドを使用すると、ファイルシステムが 存在するデバイスが拡張された後に、GFS2 ファイルシステムを拡張することができます。既存の GFS2 ファイルシステム上で **gfs2_grow** コマンドを実行すると、ファイルシステムの現在の最後とデバイスの最後との間の空の領域に、新しく初期化された GFS2 ファイルシステム拡張が書き込まれます。この書き込み操作が 終了すると、ファイルシステムのリソースインデックスが更新されます。この結果、クラスター内のすべてのノードは、追加されたストレージ領域を使用できます。

**gfs2_grow** コマンドはマウント済みのファイルシステムで実行する必要がありますが、この作業を行う必要があるのはクラスター内の 1 つのノードのみです。他のノードはすべて、ファイルシステムが拡張されたことを自動的に認識して新規領域を使い始めます。

**注記**

**mkfs.gfs2** コマンドで GFS2 ファイルシステムを作成した後はそのファイルシステムのサイズを縮小することはできません。

使用方法

```
gfs2_grow MountPoint
```

***MountPoint***
設定が適用される GFS2 ファイルシステムを指定します。

### コメント

**gfs2_grow** コマンドを実行する前に:

- ファイルシステム上の重要なデータをバックアップします。
- **df *MountPoint*** コマンドを実行して、拡張されるファイルシステムで使用するボリュームを決定します。
- LVM で配下のクラスターボリュームを拡張します。LVM ボリュームの管理についての情報は『論理ボリュームマネージャの管理』 を参照してください。

**gfs2_grow** コマンドを実行した後に、**df** コマンドを使用して、新しい領域がファイルシステムで現在利用できることをチェックします。

### 例

以下の例では、**/mygfs2fs** ディレクトリー上のファイルシステムを拡張します。

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

### 完全な使用法

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

***MountPoint***
GFS2 ファイルシステムがマウントされているディレクトリーを指定します。

***Device***
ファイルシステムのデバイスノードを指定します。

表3.3「ファイルシステムを拡張している間に利用できる GFS2 固有のオプション」 では、GFS2 ファイルシステムを拡張する際に使用できる GFS2 固有のオプションについて説明しています。

**表3.3 ファイルシステムを拡張している間に利用できる GFS2 固有のオプション**

| オプション | 説明 |
| --- | --- |
| **-h** | ヘルプ。使用法について短いメッセージを表示します。 |
| **-q** | Quiet。詳細レベルを下げます。 |
| **-r Megabytes** | 新規のリソースグループのサイズを指定します。デフォルトサイズは 256 メガバイトです。 |
| **-T** | テスト。すべての計算をしますが、ディスクへの書き込みとファイルシステムの拡張は行いません。 |
| **-V** | コマンドのバージョン情報を表示します。 |

## 3.6. GFS2 ファイルシステムへジャーナルの追加

**gfs2_jadd** コマンドを使用してジャーナルを GFS2 ファイルシステムに追加します。配下の論理ボリュームを拡張せずに随時、動的に GFS2 ファイルシステムにジャーナルを追加することができます。**gfs2_jadd** コマンドは、マウント済みのファイルシステム上で実行する必要がありますが、この作業を行う必要があるのはクラスター内の 1 つのノードのみです。他のノードはすべて、ファイルシステムが拡張されたことを認識します。

**注記**

GFS2 ファイルシステムが満杯の場合は、そのファイルシステムを含む論理ボリュームが拡張され、ファイルシステムより大きいサイズであっても **gfs2_jadd** コマンドが失敗します。これは、GFS2 ファイルシステムでジャーナルが組み込みメタデータではなくプレーンなファイルとなるためです。これにより、基礎となる論理ボリュームを単に拡張してもジャーナルには領域が提供されません。

ジャーナルを GFS2 ファイルシステムに追加する前に、以下の例のように **gfs2_edit -p jindex** コマンドを使用して現在 GFS2 ファイルシステムに含まれるジャーナルの数を確認できます。

```
# 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
```

### 使用方法

```
gfs2_jadd -j Number MountPoint
```

***Number***

新規に追加されるジャーナルの数を指定します。

***MountPoint***

GFS2 ファイルシステムがマウントされているディレクトリーを指定します。

### 例

以下の例では、1つのジャーナルが **/mygfs2** ディレクトリーのファイルシステムに追加されます。

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

以下の例では、2 つのジャーナルが **/mygfs2** ディレクトリーのファイルシステムに追加されます。

```
gfs2_jadd -j 2 /mygfs2
```

### 完全な使用法

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

***MountPoint***
GFS2 ファイルシステムがマウントされているディレクトリーを指定します。

***Device***
ファイルシステムのデバイスノードを指定します。

表3.4「ジャーナル追加時に利用できる GFS2 固有のオプション」 ではジャーナルを GFS2 ファイルシステムに追加する際に使用できる GFS2 固有のオプションを示します。

**表3.4 ジャーナル追加時に利用できる GFS2 固有のオプション**

| フラグ | パラメーター | 説明 |
|---|---|---|
| **-h** | | ヘルプ。使用法について短いメッセージ表示 |
| **-J** | ***Megabytes*** | 新規ジャーナルのサイズをメガバイトで指定します。デフォルトのジャーナルサイズは、128 メガバイトです。最小サイズは 32 メガバイトです。異なるサイズのジャーナルをファイルシステムに追加するには、各サイズのジャーナル毎に **gfs2_jadd** コマンドを実行する必要があります。指定するサイズは端数を切り捨て、ファイルシステムが 作成された時に指定してあるジャーナルセグメントサイズの倍数になるようにします。 |
| **-j** | ***Number*** | **gfs2_jadd** コマンドにより追加される新規ジャーナルの数を 指定します。デフォルト値は 1 です。 |
| **-q** | | Quiet。詳細レベルを下げます。 |
| **-V** | | コマンドのバージョン情報を表示します。 |

## 3.7. データジャーナリング

通常、GFS2 はそのジャーナルにメタデータのみを書き込みます。ファイルの内容は、ファイルシステムバッファーをフラッシュするカーネルの定期的な同期によって、最終的にディスクに書き込まれま

す。ファイルに対して **fsync()** を呼び出すと、そのファイルのデータはディスクに即時に書き込まれます。このコールは、ディスクがすべてのデータが安全に書き込まれたことを報告した時に戻ります。

データのジャーナリングでは、メタデータに加えて更にファイルデータがジャーナルに書き込まれるため、極めて小さなファイルでは **fsync()** にかかる時間を縮小することができます。ファイルサイズが大きくなると、この利点は急激に少くなります。データジャーナリング機能をオンにすると、中型および大型のファイルへの書き込み速度は大幅に低減します。

**fsync()** に依存してファイルデータを同期するアプリケーションは、データジャーナリングを使用することでパフォーマンスが向上する可能性があります。データジャーナリングは、フラグ付きディレクトリー（および、その全サブディレクトリー）内に作成された GFS2 ファイル用に、自動的に有効にされます。長さゼロの既存のファイルもデータジャーナリングをオン/オフに切り替えることができます。

１つのディレクトリー上でデータジャーナリングを有効にすると、そのディレクトリーは「inherit jdata」にセットされ、そのディレクトリー内に以後作成されるファイルやディレクトリーはすべてジャーナル化されることを示します。ファイルのデータジャーナリング機能は **chattr** コマンドで有効にしたり無効にしたりすることができます。

以下のコマンドでは、 **/mnt/gfs2/gfs2_dir/newfile** ファイルに対するデータジャーナリングを有効にしてからフラグが正しくセットされているかどうかをチェックします。

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

以下のコマンドは、**/mnt/gfs2/gfs2_dir/newfile** ファイルに対するのデータジャーナリングを無効にして、次にフラグが正しくセットされていることをチェックします。

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

また、**chattr** コマンドを使用してディレクトリーに **j** フラグを設定することもできます。このフラグをディレクトリーに設定すると、そのディレクトリーで以降に作成されるファイルやディレクトリーはすべてジャーナリングされます。以下のコマンドのセットは **gfs2_dir** ディレクトリーに **j** フラグを設定し、そのフラグが正しくセットされているかどうかを確認します。この後、コマンドにより **/mnt/gfs2/gfs2_dir** ディレクトリーに **newfile** と言う新しいファイルが作成され、そのファイルに **j** フラグが正しく設定されていることをチェックします。**j** フラグはディレクトリーが設定してあるので、**newfile** もジャーナリングが有効になっているはずです。

```
# 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
```

## 3.8. ATIME 更新の設定

ファイル inode とディレクトリー inode にはそれぞれ、関連付けられたタイムスタンプが 3 つあります。

- **ctime** – inode のステータスが最後に変更された時刻

- **mtime** – ファイル（またはディレクトリー）データが最後に修正された時刻
- **atime** – ファイル（またはディレクトリー）データが最後にアクセスされた時刻

GFS2 およびその他の Linux ファイルシステム上ではデフォルトで有効化されているように **atime** 更新が有効となっている場合には、ファイルが読み込まれる度に inode が更新される必要があります。

**atime** によって提供される情報を使用するアプリケーションはほとんどないため、これらの更新は、相当な量の不要な書き込みトラフィックとファイルロッキングトラフィックを伴う場合があります。そのようなトラフィックはパフォーマンスを低下させるため、 **atime** 更新はオフにするか、頻度を低くしたほうが良いでしょう。

**atime** 更新の影響を低減するには、2 つの方法があります。

- **relatime** (relative atime) でマウントすると、以前の **atime** 更新が **mtime** または **ctime** の更新より古い場合に、**atime** が更新されます。
- **noatime** でのマウントは、ファイルシステム上の **atime** 更新を無効にします。

### 3.8.1. relatime を使用したマウント

**relatime** (relative atime) Linux マウントオプションは、ファイルシステムのマウント時に指定することができます。これは以前の **atime** 更新が **mtime** または **ctime** の更新よりも古い場合に **atime** が更新されるように指定します。

#### 使用方法

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

***BlockDevice***

GFS2 ファイルシステムが常駐するブロックデバイスを指定します。

***MountPoint***

GFS2 ファイルシステムがマウントされるディレクトリーを指定します。

#### 例

この例では、GFS2 ファイルシステムは **/dev/vg01/lvol0** に存在し、ディレクトリー **/mygfs2** にマウントされます。**atime** 更新は、以前の **atime** 更新が **mtime** または **ctime** 更新よりも古い場合にのみ実行されます。

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

### 3.8.2. noatime を使用したマウント

Linux マウントオプション **noatime** は、ファイルシステムのマウント時に指定できます。これはファイルシステムでの **atime** 更新を無効にします。

#### 使用方法

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

***BlockDevice***

GFS2 ファイルシステムが常駐するブロックデバイスを指定します。

***MountPoint***

GFS2 ファイルシステムがマウントされるディレクトリーを指定します。

**例**

この例では、GFS2 ファイルシステムは **/dev/vg01/lvol0** に存在し、**atime** 更新が無効な状態でディレクトリー **/mygfs2** にマウントされます。

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

## 3.9. GFS2 ファイルシステム上の動作の一時停止

**dmsetup suspend** コマンドを使用すると、ファイルシステムへの書き込み動作を一時停止することができます。書き込み動作を一時停止することにより、ハードウェアベースデバイスのスナップショットを使用して ファイルシステムを一貫した状態でキャプチャーすることができます。一時停止を終了するには、**dmsetup resume** コマンドを実行します。

### 使用方法

**一時停止の開始**

```
dmsetup suspend MountPoint
```

**一時停止の終了**

```
dmsetup resume MountPoint
```

***MountPoint***

ファイルシステムを指定します。

### 例

以下の例では、ファイルシステム **/mygfs2** への書き込みを一時停止します。

```
# dmsetup suspend /mygfs2
```

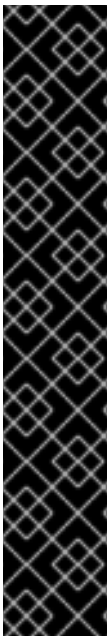
この例では、ファイルシステム **/mygfs2** への書き込みの一時停止を終了します。

```
# dmsetup resume /mygfs2
```

## 3.10. GFS2 ファイルシステムの修復

ファイルシステムをマウントした状態でノードに障害が発生した場合、ファイルシステムジャーナリングにより、迅速な復旧が可能になります。ただし、ストレージデバイスの電源が切れたり、物理的に切断されると、ファイルシステムの破損が発生する可能性があります（ジャーナリングはストレージサブ

システムの障害からのリカバリには使用できません）。この種の破損が発生した場合は、 **fsck.gfs2** コマンドを使用して GFS2 ファイルシステムのリカバリを行うことができます。



**重要**

**fsck.gfs2** コマンドは、すべてのノードからアンマウントされた ファイルシステム上でのみ実行する必要があります。ファイルシステムが Pacemaker クラスターリソースとして管理される場合は、ファイルシステムリソースを無効にすることができます (これによりファイルシステムはアンマウントされます)。ファイルシステムリソースは、**fsck.gfs2** コマンドの実行後に再び有効にします。**pcs resource disable** の **--wait** オプションで *timeout* 値を指定すると、値が秒単位で示されます。

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

**fsck.gfs2** コマンドが GFS2 ファイルシステム上でブート時に実行されないようにするには、クラスターで GFS2 ファイルシステムリソースを作成するときに **options** 引数の **run_fsck** パラメーターを設定します。**"run_fsck=no"** と指定すると、**fsck** コマンドが実行されません。

**注記**

以前に GFS ファイルシステムで **gfs_fsck** コマンドを使用した経験がある場合は、**fsck.gfs2** コマンドが以下の点で **gfs_fsck** の以前の一部のリリースと異なることに注意してください。

- **fsck.gfs2** コマンドの実行中に **Ctrl**+**C** を押すと処理が中断され、コマンドを中止するかどうか、現在の残りのパスを省略するかどうか、または処理を続行するかどうかを尋ねるプロンプトが表示されます。
- **-v** フラグを使用すると、詳細度を高くすることが出来ます。第 2 の **-v** フラグを追加すると、レベルが更に高くなります。
- **-q** フラグを使用すると、詳細度を低くすることができます。第 2 の **-q** フラグを追加すると、レベルがさらに低くなります。
- **-n** オプションを使用すると、ファイルシステムが読み取り専用として開き、すべての質問に対して自動的に **no** と回答します。このオプションでは、**fsck.gfs2** コマンドを実際に有効にせずに使用してエラーを見つけることができます。

その他のコマンドオプションについては **gfs2.fsck** の man ページを参照してください。

**fsck.gfs2** コマンドの実行には、オペレーティングシステムとカーネルに使用するメモリー以上のシステムメモリーが必要です。GFS2 ファイルシステム自体の各メモリーブロックには約 5 ビットまたは 5/8 バイトの追加メモリーが必要になります。 このため、ファイルシステムで **fsck.gfs2** を実行するために必要なメモリーのバイト数を判断するには、ファイルシステムに含まれているブロック数に 5/8 を乗算します。

たとえば、1 ブロックサイズが 4K の 16TB の GFS2 ファイルシステムで **fsck.gfs2** コマンドを実行するのに必要なメモリー容量を概算する場合は、最初に 16TB を 4K で割ってファイルシステムに含まれるメモリーのブロック数を計算します。

```
17592186044416 / 4096 = 4294967296
```

このファイルシステムに含まれているブロック数は 4294967296 なので、この値に 5/8 を乗算して、必要なメモリーのバイト数を求めます。

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

**fsck.gfs2** コマンドを実行するには、 このファイルシステムに約 2.6 GB の空きメモリーが必要になります。ブロックサイズが 1K の場合は **fsck.gfs2** コマンドの実行に 4 倍のメモリーまたは 11 GB の空きメモリーが必要になります。

### 使用方法

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

**-y**

**-y** フラグを設定すると、すべての質問に対して **yes** と返します。**-y** フラグが指定した場合には、その設定を変更するまで **fsck.gfs2** コマンドは答えを要求するプロンプトは表示しません。

***BlockDevice***

GFS2 ファイルシステムが常駐するブロックデバイスを指定します。

### 例

この例では、ブロックデバイス **/dev/testvol/testlv** に存在する GFS2 ファイルシステムが修復されます。修復に関するすべての質問には自動的に **yes** と回答されます。

```
# 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
```

## 3.11. GFS2 WITHDRAW 機能

GFS2 の *withdraw* 機能は、GFS2 ファイルシステムのデータ整合性機能であり、ハードウエアまたはカーネルソフトウェアの不具合によるファイルシステムの損傷を防ぎします。指定したクラスターノードで GFS2 ファイルシステムを使用している場合に、GFS2 カーネルが非整合性を検出すると、マウント解除して再マウントするまでそのノードで利用できなくなります (または問題を検出したマシンが再起動します)。マウントしたその他の GFS2 ファイルシステムは、そのノードで完全に機能し続けます。GFS2 の無効機能は、ノードをフェンスする原因となるカーネルパニックよりも厄介なものではありません。

以下は、GFS2 を無効にする可能性のある非整合の種類です。

- inode 整合性エラー
- リソースグループの整合性エラー
- ジャーナル整合性エラー
- マジックナンバーのメタデータの整合性エラー
- メタデータ型の整合性エラー

GFS2 を無効にさせる可能性のある非整合の例としては、ファイルの inode に対する誤ったブロック数があります。GFS2 がファイルを削除すると、そのファイルが参照しているデータとメタデータブロックはすべて体系的に削除されます。削除が完了すると、inode のブロック数が確認されます。ブロック数が 1 ではない場合 (つまり、残っているのはディスクの inode 自体であることを意味します)、inode のブロック数は、実際にファイルに使用されているブロックと一致しなかったため、ファイルシステムの整合性が取れていないと示されます。

多くの場合、問題はハードウェアの不具合 (メモリー、マザーモード、HBA、ディスクドライブ、ケーブルの故障) が原因である可能性があります。また、カーネルのバグ (別のカーネルモジュールが GFS2 のメモリーを誤って上書きする) や、実際のファイルシステムの損傷 (GFS2 バグにより発生する) により発生した可能性もあります。

ほとんどの場合、GFS2 の不整合は、クラスターノードを再起動すると解決します。クラスターノードを再起動する前に、Pacemaker からシステムの「クローン」システムを無効にします。これにより、そのノードでのみファイルシステムのマウントが解除されます。

```
# pcs resource disable --wait=100 mydata_fs_clone
# /sbin/reboot
```

**警告**

**umount** と **mount** コマンドを使用してファイルシステムのマウントを解除して再マウントしないでください。**pcs** コマンドを使用してください。このコマンドを使用しないと、ファイルシステムサービスが消えたことを Pacemaker が検出し、ノードを隔離します。

無効にした整合性の問題によりシステムがハングアップする可能性があるため、ファイルシステムのサービスを停止できなくなる可能性があります。

再マウントしても問題が解決しない場合は、ファイルシステムを停止して、クラスター内の全ノードからファイルシステムのマウントを削除し、以下の手順に従ってサービスを再起動する前に、fsck.gfs2 コマンドでファイルシステムのチェックを実行します。

1. 影響を受けるノードを再起動します。
2. Pacemaker で非クローンのファイルシステムサービスを無効にして、クラスター内のすべてのノードからファイルシステムのマウントを解除します。

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

3. クラスターの 1 つのノードから、ファイルシステムデバイスで **fsck.gfs2** コマンドを実行して、ファイルシステムの損傷を確認して修復します。

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

4. ファイルシステムサービスを再度有効にして、すべてのノードで GFS2 ファイルシステムを再マウントします。

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

ファイルシステムサービスで **-o errors=panic** オプションを指定したファイルシステムをマウントすると、GFS2 の無効機能をオーバーライドできます。

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

このオプションが指定されている場合、通常はシステムを無効にするようなエラーが発生すると、代わりにカーネルパニックが発生します。これによりノードの通信が停止し、ノードがフェンスされます。これは特に、長期間人による監視や介入がないクラスターに役に立ちます、

内部的には、GFS2 の無効機能は、ロッキングプロトコルを切断することで機能し、それ以降のすべてのファイルシステム操作で I/O エラーが発生するようにします。その結果、GFS2 の無効化が発生すると、デバイスマッパーデバイスから多数の I/O エラーがシステムログに報告されるのが普通です。

# 第4章 GFS2 ファイルシステムに伴う問題の診断と修正

この章では、GFS2 の一般的な問題と対処方法についての情報を提供します。

## 4.1. GFS2 ファイルシステムのパフォーマンス低下

ご使用の GFS2 ファイルシステムのパフォーマンスが EXT3 ファイルシステムよりも低下する場合があります。GFS2 のパフォーマンスは、多数の要因および特定のユースケースで影響を受ける場合があります。GFS2 のパフォーマンス問題の対処方法は、本ガイドの随所に記載しています。

## 4.2. GFS2 ファイルシステムがハングし、単一ノードのリブートが必要

ご使用の GFS2 ファイルシステムがハングし、それに対して実行したコマンドを返さないが、ある特定のノードをリブートするとシステムが正常な状態に戻る場合には、ロックの問題もしくはバグの兆候である可能性があります。このような事態が発生した場合には、以下のデータを収集してください。

- 各ノード上のファイルシステム用の GFS2 ロックダンプ:

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

- 各ノード上のファイルシステム用の DLM ロックダンプ: この情報は、**dlm_tool** を使用して確認することができます。

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

  このコマンドでは、*lsname* は、対象のファイルシステム用に DLM が使用するロックスペース名です。この値は、**group_tool** コマンドの出力で確認することができます。

- **sysrq -t** コマンドの出力

- **/var/log/messages** ファイルの内容

データを収集したら、Red Hat サポートのチケットを起票して、収集したデータを提出してください。

## 4.3. GFS2 ファイルシステムがハングし、全ノードのリブートが必要

ご使用の GFS2 ファイルシステムがハングし、それに対して実行したコマンドを返さず、使用できる状態にするにはクラスター内の全ノードをリブートする必要がある場合、以下の問題を確認してください。

- フェンスに障害が発生している可能性があります。GFS2 ファイルシステムがフリーズし、障害が発生したフェンスのイベントのデータ整合性が確保されます。メッセージログを確認し、ハング時にフェンスに障害が発生していたかどうかを調べます。フェンスが正しく設定されているかどうかも確認します。

- GFS2 ファイルシステムが無効な状態 (withdraw) になっている場合があります。メッセージログに **withdraw** という単語があるのを調べ、ファイルシステムが withdraw な状態になっていることを示す GFS2 からのメッセージおよびコールトレースの有無を確認します。この状態は、ファイルシステムのは存、ストレージの障害、またはバグがあることを示しています。早い時期に、ファイルシステムのマウントした方が便利な場合は、以下の手順を行ってください。

  1. 無効な状態 (withdraw) が発生したノードを再起動します。

```
# /sbin/reboot
```

2. ファイルシステムリソースを停止して、すべてのノードで GFS2 ファイルシステムをマウント解除します。

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

3. **gfs2_edit savemeta...** コマンドでメタデータを取得します。ファイルに十分な空きがあることを確認する必要があります。場合によってはサイズが大きくなる場合があります。この例では、メタデータは **/root** ディレクトリーのファイルに保存されています。

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

4. **gfs2-utils** パッケージを更新します。

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

5. 1つのノードで、システム上において **fsck.gfs2** コマンドを実行し、ファイルシステムの整合性を確保して損傷を修復します。

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

6. **fsck.gfs2** コマンドが完了したら、ファイルシステムのリソースを再度有効にして、サービスに戻します。

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

7. Red Hat サポートのチケットを作成します。GFS2 が無効になったことを伝え、**sosreports** コマンドおよび **gfs2_edit savemeta** コマンドで生成されたログとデバッグ情報を添付してください。

GFS2 の一部のインスタンスを無効にした場合は、ファイルシステムまたはそのブロックデバイスにアクセスしようとしているコマンドがハングすることがあります。このような場合は、クラスターを再起動するにはハードリブートが必要です。

GFS2 の withdraw 機能の説明は、「GFS2 withdraw 機能」 を参照してください。

- ロック関連の問題が発生したかバグの可能性があります。問題の発生中にデータを収集し、「GFS2 ファイルシステムがハングし、単一ノードのリブートが必要」 で説明されたように Red Hat サポートのチケットを作成してください。

## 4.4. 新たに追加されたクラスターノードに GFS2 ファイルシステムをマウントできない

クラスターに新たなノードを追加して、そのノードで GFS2 ファイルシステムをマウントできない場合は、GFS2 ファイルシステムにアクセスしようとしているノードよりも、GFS2 ファイルシステム上のジャーナルの方が少ない可能性があります。ファイルシステムをマウントする GFS2 ホストごとに 1 つのジャーナルが必要です (ただし、**spectator** マウントオプションが設定された状態でマウントされた GFS2 ファイルシステムはジャーナルが必要ないため、例外となります)。GFS2 ファイルシステムにジャーナルを追加するには、「GFS2 ファイルシステムへジャーナルの追加」で説明されているように、**gfs2_jadd** コマンドを使用します。

## 4.5. 空のファイルシステムで使用中と表示される領域

空の GFS2 ファイルシステムがある場合は、**df** コマンドを使用すると、使用領域が表示されます。これは、GFS2 ファイルシステムジャーナルが、ディスクの領域 (ジャーナル数 x ジャーナルサイズ) を消費するためです。多数のジャーナルとともに GFS2 ファイルシステムを作成した場合、または大きなジャーナルサイズを指定した場合は、**df** を実行した際に、(ジャーナル数 x ジャーナルサイズ) が既に使用中であることが示されます。大量のジャーナルまたは大きいジャーナルを指定しなかった場合でも、小さい GFS2 ファイルシステム (1 GB 以下の範囲) では、デフォルトの GFS2 ジャーナルサイズで大きな領域が使用中であることが示されます。

# 第5章 クラスターでの GFS2 ファイルシステムの設定

以下に、GFS2 ファイルシステムを含む Pacemaker クラスターの設定に必要な手順の概要を示します。

すべてのノードでのクラスターソフトウェアのインストールと起動が完了した後でクラスターを作成します。クラスターのフェンシングを設定する必要があります。Pacemaker クラスターの作成とクラスターのフェンシングの設定は、『High Availability Add-On の管理』 の 「Creating a Red Hat High-Availability Cluster with Pacemaker」を参照してください。その後は、以下の手順を実行します。

1. クラスターのすべてのノードで、Resilient Storage チャンネルから **lvm2-cluster** と **gfs2-utils** パッケージをインストールします。

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

2. グローバル Pacemaker パラメーター **no_quorum_policy** を **freeze** に設定します。

**注記**

**no-quorum-policy** の値は、デフォルトでは **stop** に設定され、定足数を失うと、残りのパーティション上の全リソースが直ちに停止されます。一般的には、このデフォルト値が最も安全で最適なオプションですが、ほとんどのリソースとは異なり、GFS2 が正しく機能するには定足数が必要です。定足数を失うと、GFS2 マウントを使用しているアプリケーションと GFS2 マウント自体の両方が正しく停止できなくなります。定足数がないままこうしたリソースを停止しようとすると、停止に失敗し、最終的に定足数を失う度にクラスター全体がフェンシングされます。

このような状況に対処するため、GFS2 を使用している場合は **no-quorum-policy=freeze** を **freeze** に設定できます。この場合は、定足数が失われると、定足数を取り戻すまで残りのパーティションで何の処理も行われなくなります。

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

3. **dlm** リソースをセットアップします。これは、**clvmd** および GFS2 に必要な依存関係です。

```
# pcs resource create dlm ocf:pacemaker:controld op monitor interval=30s on-fail=fence
clone interleave=true ordered=true
```

4. クラスター化ロックを有効にするために、クラスターの各ノードで以下のコマンドを実行します。このコマンドを実行すると、**/etc/lvm/lvm.conf** ファイルの **locking_type** パラメーターが 3 に設定されます。

```
# /sbin/lvmconf --enable-cluster
```

5. **clvmd** をクラスターリソースとしてセットアップします。

```
# pcs resource create clvmd ocf:heartbeat:clvm op monitor interval=30s on-fail=fence clone
interleave=true ordered=true
```

**clvmd** と **cmirrord** は、**ocf:heartbeat:clvm** リソースエージェントを使用して Pacemaker で起動と管理を行うことに注意してください。また、**systemd** でブートしているときは、起動する

必要はありません。さらに、開始手順の一環として **ocf:heartbeat:clvm** リソースエージェントは、**/etc/lvm/lvm.conf** ファイルの **locking_type** パラメーターを 3 に設定し、**lvmetad** デーモンを無効にします。

6. **clvmd** および **dlm**の依存関係をセットアップし、順番に起動します。**clvmd** は **dlm** の後に起動し、**dlm** と同じノードで実行する必要があります。

```
# pcs constraint order start dlm-clone then clvmd-clone
# pcs constraint colocation add clvmd-clone with dlm-clone
```

7. クラスター化論理ボリュームを作成します。

```
# pvcreate /dev/vdb
# vgcreate -Ay -cy sasbin_vg /dev/vdb
# lvcreate -L5G -n sasbin_lv sasbin_vg
```

**警告**

CLVM を使用して共有ストレージ上にボリュームグループを作成する際には、クラスター内のすべてのノードが、ボリュームグループを構成する物理ボリュームに確実にアクセスできるようにする必要があります。ストレージにアクセスできるノードとできないノードが混在する、非対称型のクラスター構成はサポートされていません。

複数のノードにわたるボリュームを同時に有効にできるようにするために、CLVMD を使用してボリュームグループを管理する場合は、そのボリュームグループでクラスター化フラグを有効している必要があります。このフラグにより、CLVMD は管理する必要のあるボリュームを識別できるようになり、CLVMD が LVM メタデータの連続性を維持することができるようになります。この設定を行わないと、Red Hat サポート対象外の環境となり、ストレージが破損したり、データが失われたりすることがあります。

8. GFS2 ファイルシステムで論理ボリュームをフォーマットします。ファイルシステムをマウントする各ノードにジャーナルが 1 つ必要です。クラスター内の各ノードには必ず十分なジャーナルを作成してください。

```
# mkfs.gfs2 -j2 -p lock_dlm -t rhel7-demo:sasbin /dev/sasbin_vg/sasbin_lv
```

**警告**

GFS2 ファイルシステムを作成する場合は、**-t *LockTableName*** オプションに対して正しい値を指定することが重要です。適切な形式は、*ClusterName:FSName* です。正しい値を指定しないと、ファイルシステムをマウントできなくなります。また、ファイルシステムの名前は固有である必要があります。**mkfs.gfs2** コマンドのオプションの説明は、「GFS2 ファイルシステムの作成」 を参照してください。

9. **clusterfs** リソースを設定します。

   このファイルシステムは Pacemaker のクラスターリソースとして管理されるため、 **/etc/fstab** ファイルには追加しないでください。マウントオプションは、**options=*options*** を使用してリソース設定の一部として指定できます。すべての設定オプションを確認する場合は、**pcs resource describe Filesystem** コマンドを実行します。

   このクラスターリソースの作成コマンドは、**noatime** マウントオプションを指定します。これは、アプリケーションが許可する GFS2 ファイルシステムで推奨されます。

   この例では、ファイルシステムの名前はマウントポイントと同じです。これは必須ではありませんが、ファイルシステムに問題が発生した場合のトラブルシューティングに役立つように、ファイルシステム名を実際の使用またはマウントポイントに関連付けることが推奨されます。

   ```
   # # pcs resource create clusterfs Filesystem device="/dev/sasbin_vg/sasbin_lv"
   directory="/usr/local/sasbin" fstype="gfs2" options="noatime" op monitor interval=10s on-
   fail=fence clone interleave=true
   ```

10. GFS2 と **clvmd** の依存関係をセットアップし、順番に起動します。GFS2 は **clvmd** の後に起動し、**clvmd** と同じノードで実行する必要があります。

    ```
    # pcs constraint order start clvmd-clone then clusterfs-clone
    # pcs constraint colocation add clusterfs-clone with clvmd-clone
    ```

11. 予想どおり GFS2 がマウントされていることを確認します。

    ```
    # mount |grep sas
    /dev/mapper/sasbin_vg-sasbin_lv on /usr/local/sasbin type gfs2 (rw,noatime,seclabel)
    ```

# 付録A PERFORMANCE CO-PILOT による GFS2 パフォーマンスの分析

Red Hat Enterprise Linux 7 は、GFS2 パフォーマンスメトリックとともに Performance Co-Pilot (PCP) をサポートします。このツールを使用すると、GFS2 ファイルシステムのパフォーマンスを監視できます。この付録では、GFS2 パフォーマンスメトリックとその使用方法について説明します。

## A.1. PERFORMANCE CO-PILOT の概要

Performance Co-Pilot (PCP) は、コンピューター、アプリケーション、およびサーバーのステータス、アクティビティー、およびパフォーマンスを監視、視覚化、記録、および制御するオープンソースのツールキットです。PCP を使用すると、リアルタイムデータの監視および管理と、履歴データのロギングおよび取得を行えます。履歴データは、ライブ結果とアーカイブデータを比較して問題のパターンを分析するために使用できます。

PCP は、クライアントサーバーアーキテクチャーに基づいて設計されています。PCP コレクターサービスは、Performance Metric Collector Daemon (PMCD) であり、サーバーにインストールして実行できます。PCP コレクターサービスが起動すると、PCMD はインストールされた Performance Metric Domain Agent (PMDA) からパフォーマンスデータの収集を開始します。PMDA は、システムで個別にロードまたはアンロードでき、同じホスト上の PMCD によって制御されます。PCP の GFS2 ファイルシステムのパフォーマンスメトリックデータを収集するには、デフォルトの PCP インストールの一部である GFS2 PMDA が使用されます。

表A.1「PCP ツール」 には、本章で説明する PCP Toolkit に含まれる一部の PCP ツールの簡潔なリストが示されています。その他の PCP ツールについては、**PCPIntro**(1) man ページとその他の PCP man ページを参照してください。

**表A.1 PCP ツール**

| ツール | 用途 |
|---|---|
| **pmcd** | Performance Metric Collector Service: PMDA からメトリックデータを収集し、PCP の他のコンポーネントでメトリックデータを利用できるようにする |
| **pmlogger** | 他の PCP ツールでプレイバックできるパフォーマンスメトリック値のアーカイブログを作成できる |
| **pmproxy** | PCP 監視クライアントが **pmproxy** により **pmcd** の 1 つまたは複数のインスタンスに接続することを可能にする **pmcd** のプロトコルプロキシー |
| **pminfo** | コマンドラインでパフォーマンスメトリックに関する情報を表示する |
| **pmstore** | パフォーマンスメトリック値を変更できる (カウンターの再初期化または新しい値の割り当て) |
| **pmdumptext** | パフォーマンスメトリックデータをライブで、またはパフォーマンスアーカイブから ASCII テーブルにエクスポートする |
| **pmchart** | パフォーマンスメトリック値を表にプロットするグラフィカルユーティリティー (**pcp-gui** パッケージ) |

## A.2. PCP デプロイメント

クラスター全体を監視するには、GFS2 PMDA が他の PCP サービスとともにクラスターの各ノードで有効になり、ロードされるよう PCP をインストールおよび設定することが推奨されます。これにより、ノードをローカルで監視したり、対応する PMDA が監視モードでロードされ、PCP がインストールされたマシンでリモートで監視したりできます。また、オプションの **pcp-gui** パッケージをインストールして、**pmchart** ツールでトレースデータを視覚的に表示することもできます。

詳細については、デフォルトで **/usr/share/doc/pcp-doc** にインストールされる **pcp-doc** パッケージを参照してください。PCP は、各ツールの man ページも提供します。

## A.3. PCP インストール

PCP のテスト済み最新バージョンは、Red Hat Enterprise Linux 7 リポジトリーからダウンロードできます。

GFS2 PMDA が正常に機能するには、**debugfs** ファイルシステムをマウントする必要があります。**debugfs** ファイルシステムがマウントされていない場合は、GFS2 PMDA をインストールする前に以下のコマンドを実行します。

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

GFS2 PMDA は、インストール時にデフォルトで有効になりません。PCP で GFS2 メトリック監視を使用するには、GFS2 ドメインエージェントを有効にする必要があります。以下のコマンドを使用して PCP と GFS2 PMDA モジュールをインストールし、GFS2 PMDA を有効にします。PMDA インストールスクリプトは root で実行する必要があることに注意してください。

```
# yum install pcp pcp-gui pcp-pmda-gfs2
# cd /var/lib/pcp/pmdas/gfs2
# ./Install
```

PMDA インストールスクリプトを実行すると、PMDA で使用するロールを指定するよう求められます。

- **collector** と指定すると、現在のシステムのパフォーマンスメトリックの収集が許可されます。
- **monitor** と指定すると、システムでローカルシステムまたはリモートシステム、あるいはローカルおよびリモートシステムの両方を監視できます。
- **both** と指定すると、**collector** 設定と **monitor** 設定の両方が有効になります。

ほとんどの場合、PMDA を正常に稼働させるにはデフォルトの選択 (collector と monitor) で十分です。

```
# ./Install
You will need to choose an appropriate configuration for installation of
the "gfs2" Performance Metrics Domain Agent (PMDA).

  collector collect performance statistics on this system
  monitor allow this system to monitor local and/or remote systems
  both  collector and monitor configuration for this system

Please enter c(ollector) or m(onitor) or b(oth) [b]
Updating the Performance Metrics Name Space (PMNS) ...
```

```
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Waiting for pmcd to terminate ...
Starting pmcd ...
Starting pmlogger ...
Check gfs2 metrics have appeared ... 316 metrics and 205 values
```

GFS2 PMDA のインストールでエラーまたは警告が発生した場合は、PMCD が起動し、稼働していることと **debugfs** がマウントされていることを確認してください (システムに GFS2 ファイルシステムが 1 つもロードされていない場合は、警告が発生することがあります)。

**注記**

クラスターノードで GFS2 PMDA をインストールする場合は、PMDA 設定のデフォルト値 (両方) を選択するだけで、PMDA が正常に機能します。ワークステーションマシンでリモート PCP インストールからのデータを監視するだけが目的の場合は、PMDA をモニターとしてインストールすることが推奨されます。

## A.4. GFS2 パフォーマンスデータのトレース

PCP がインストールされ、GFS2 PMDA が有効な場合、PCP と GFS2 に利用可能なパフォーマンスメトリックスを確認する最も簡単な方法は、**pminfo** ツールを使用することです。**pminfo** コマンドラインツールは、利用可能なパフォーマンスメトリックスに関する情報を表示します。通常、**pminfo** はローカルメトリックネームスペースを使用して機能しますが、**-h** フラグを使用してリモートホストでメトリックスを表示するよう変更できます。**pminfo** ツールの詳細については、**pminfo**(1) man ページを参照してください。

以下のコマンドを実行すると、GFS2 PMDA により提供された利用可能なすべての GFS2 メトリックスのリストが表示されます。

```
# pminfo gfs2
```

各メトリックのヘルプ情報と説明を取得するために **-T** フラグを指定したり、各メトリックに対応するパフォーマンス値の現在の値を取得するために **-f** フラグを指定したりできます。これは、メトリックのグループまたは個別メトリックに対して行えます。ほとんどのメトリックデータは、プローブ時に、システムにマウントされた各 GFS2 ファイルシステムに対して提供されます。

```
# pminfo -t gfs2.glocks
gfs2.glocks.total [Count of total observed incore GFS2 global locks]
gfs2.glocks.shared [GFS2 global locks in shared state]
gfs2.glocks.unlocked [GFS2 global locks in unlocked state]
gfs2.glocks.deferred [GFS2 global locks in deferred state]
gfs2.glocks.exclusive [GFS2 global locks in exclusive state]

# pminfo -T gfs2.glocks.total

gfs2.glocks.total
Help:
Count of total incore GFS2 glock data structures based on parsing the contents
of the /sys/kernel/debug/gfs2/bdev/glocks files.

# pminfo -f gfs2.glocks.total
```

```
gfs2.glocks.total
    inst [0 or "testcluster:clvmd_gfs2"] value 74
```

6 つの GFS2 メトリックスグループがあり、各グループがルート GFS2 メトリックの新しいリーフノードになるよう配置されます (セパレーターとして '.' を使用)。これはすべての PCP メトリックスに該当します。表A.2「GFS2 向け PCP メトリックグループ」では、各グループで利用可能なメトリックスのタイプの概要について説明しています。各メトリックでは、**pminfo** ツールで **-T** フラグを使用して追加情報を見つけることができます。

**表A.2 GFS2 向け PCP メトリックグループ**

| メトリックグループ | 提供されるメトリック |
|---|---|
| **gfs2.sbstats.*** | システムで現在マウントされている各 GFS2 ファイルシステムのスーパーブロック統計ファイル (**sbstats**) から収集された情報に関するタイミングメトリック。 |
| **gfs2.glocks.*** | システムで現在マウントされている各 GFS2 ファイルシステムの glock の数をカウントする glock 統計ファイル (**glocks**) から収集された情報に関するメトリック。 |
| **gfs2.glstats.*** | システムで現在マウントされている各 GFS2 ファイルシステムの各タイプの glock の数をカウントする glock 統計ファイル (**glstats**) から収集された情報に関するメトリック。 |
| **gfs2.tracepoints.*** | システムで現在マウントされている各ファイルシステムの GFS2 **debugfs** トレースポイントからの出力に関するメトリック。これらのメトリックの各サブタイプ (いずれかの GFS2 トレースポイント) は、制御メトリックを使用してオンまたはオフに関係なく個別に制御できます。 |
| **gfs2.worst_glock.*** | マウントされた各ファイルシステムの「現在最も悪い glock」を計算するために **gfs2_glock_lock_time** トレースポイントからデータを使用する計算済みメトリック。このメトリックはロックの競合を検出するのに役立ちます。ファイルシステムは、同じロックが複数回提示される場合は低速になります。 |
| **gfs2.latency.grant.*** | マウントされた各ファイルシステムに対して glock 許可要求が完了するまでの平均レイテンシーをマイクロ秒単位で計算するために **gfs2_glock_queue** と **gfs2_glock_state_change** の両方のトレースポイントからデータを使用する計算済みメトリック。このメトリックは、許可レイテンシーが増加するときにファイルシステムの低速化を検出するのに役立ちます。 |
| **gfs2.latency.demote.*** | マウントされた各ファイルシステムに対して glock 降格要求が完了するまでの平均レイテンシーをマイクロ秒単位で計算するために **gfs2_glock_state_change** と **gfs2_demote_rq** の両方のトレースポイントからデータを使用する計算済みメトリック。このメトリックは、降格レイテンシーが増加するときにファイルシステムの低速化を検出するのに役立ちます。 |

| メトリックグループ | 提供されるメトリック |
| --- | --- |
| **gfs2.latency.queue.*** | マウントされた各ファイルシステムに対して glock キュー要求が完了するまでの平均レイテンシーをマイクロ秒単位で計算するために **gfs2_glock_queue** トレースポイントからデータを使用する計算済みメトリック。 |
| **gfs2.control.*** | **pmstore** ツールにより有効または無効にしたり、切り替えたりするトレースポイントメトリックを制御するために使用する設定メトリック。これらの設定メトリックは、「メトリック設定 (**pmstore** の使用)」で説明されています。 |

## A.5. メトリック設定 (PMSTORE の使用)

PCP の一部のメトリックでは、特にメトリックが制御変数として動作する場合に、メトリック値を変更できます。これは、GFS2 PMDA で **gsf2.control.*** メトリックを使用する場合に該当します。これは、**pmstore** コマンドラインツールを使用して実現されます。ほとんどの他の PCP ツールの場合と同様に、**pmstore** ツールは通常、ローカルシステムの指定されたメトリックに対する現在値を変更しますが、**-h** スイッチを使用して、指定されたリモートシステムでメトリック値を変更できます。詳細については、**pmstore**(3) man ページを参照してください。

例として、以下のコマンドは、GFS2 PMDA がインストールおよびロードされたシステムのローカルマシンですべての GSF2 トレースポイントを有効にします。このコマンドが実行されると、PMDA により、**debugfs** ファイルシステムのすべての GFS2 トレースポイントがオンになります。

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

表A.3「制御トレースポイント」では、各制御トレースポイントとその使用方法について説明しています。各制御トレースポイントと利用可能なオプションの効果の説明は、**pminfo** ツールで help スイッチを指定して確認できます。

**表A.3 制御トレースポイント**

| 制御メトリック | 用途と利用可能なオプション |
| --- | --- |
| **gfs2.contol.tracepoints.all** | GFS2 トレースポイント統計は、0 [オフ] または 1 [オン] を使用して手動で制御できます。メトリックの値を設定すると、トレースポイントメトリックからの収集が試行されるかどうかに関係なく、PMDA の動作が制御されます。 |
| **gfs2.control.tracepoints.*** | GFS2 トレースポイント統計は、0 [オフ] または 1 [オン] を使用して手動で制御できます。メトリックの値を設定すると、指定された各トレースポイントメトリックからの収集が試行されるかどうかに関係なく、PMDA の動作が制御されます。 |
| **gfs2.control.global_tracing** | グローバルトレースは、0 [オフ] または 1 [オン] を使用して制御できます。ほとんどの GFS2 メトリックが機能するために、これはオンである必要があります。 |

| 制御メトリック | 用途と利用可能なオプション |
| --- | --- |
| **gfs2.control.worst_glock** | オンまたはオフであるかに関係なく、コントロール metrics.0 [オフ] または 1 [オン] を使用して個別に制御できます。メトリックの値を設定すると、**lock_time** メトリックの収集が試行されるかどうかに関係なく、PMDA の動作が制御されます。**glock_lock_time** ベースのメトリックが機能するには、マシンで GFS2 トレースポイントが利用可能である必要があります。 |
| **gfs2.control.latency** | **gfs2.latency** 統計は、**pmstore gfs2.control.latency** 0 [オフ] または 1 [オン] を使用して手動で制御できます。メトリックの値を設定すると、**latency** メトリックの収集が試行されるかどうかに関係なく、PMDA の動作が制御されます。latency メトリックが機能するには、マシンで gfs2 トレースポイントが利用可能である必要があります。 |
| **gfs2.control.glock_threshold** | すべての **ftrace** 統計で処理され、受け入れられる glock の数。この数は、処理する glock の数を調整するために **pmstore** ツールを使用して手動で変更できます。この値は正である必要があります。 |

## A.6. パフォーマンスデータのロギング (PMLOGGER の使用)

PCP では、**pmlogger** ツールでシステムの選択されたメトリックのアーカイブログを作成して、後でリプレイできるパフォーマンスメトリック値をログに記録できます。これらのメトリックアーカイブは、過去のパフォーマンス分析を行うために後でプレイバックできます。

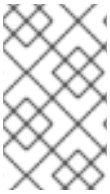
**pmlogger** ツールを使用すると、システムで記録されるメトリックと頻度を指定して、ログに記録されたメトリックを柔軟に制御できます。デフォルトでは、**pmlogger** の設定ファイルは **/var/lib/pcp/config/pmlogger/config.default** に格納されます。この設定ファイルには、プライマリーロギングインスタンスによりログに記録されるメトリックの概要が記載されています。

**pmlogger** がローカルマシンでメトリック値をログに記録するには、プライマリーロギングインスタンスを起動する必要があります。**systemctl** を使用すると、マシンの起動時に **pmlogger** をサービスとして起動するようにできます。

以下の例は、GFS2 パフォーマンスメトリックの記録を有効にする **pmlogger** 設定ファイルの一部を示しています。この部分は **pmlogger** が 10 秒ごとに PCP GFS2 レイテンシーメトリックのパフォーマンスメトリック値、30 秒ごとに最も悪い上位 10 の glock メトリック、1 分ごとにトレースポイントデータをログに記録し、10 分ごとに **glock**、**glstats**、および **sbstats** メトリックからデータをログに記録することを示しています。

```
# It is safe to make additions from here on ...
#

log mandatory on every 5 seconds {
    gfs2.latency.grant
    gfs2.latency.queue
    gfs2.latency.demote
    gfs2.glocks
}

log mandatory on every 10 seconds {
    gfs2.worst_glock
}
```

```
log mandatory on every 30 seconds {
    gfs2.tracepoints
}

log mandatory on every 5 minutes {
    gfs2.glstats
    gfs2.sbstats
}

[access]
disallow * : all;
allow localhost : enquire;
```



**注記**

PCP では、**pmlogger** が有効な場合にホストのログに記録されるデフォルトのメトリックセットが提供されます。ただし、このデフォルトの設定では、GFS2 メトリックのロギングは実行されません。

メトリックデータの記録後には、システムでの PCP ログアーカイブのリプレイに関して複数のオプションがあります。ログをテキストファイルにエクスポートし、スプレッドシートにインポートしたり、グラフを使用して PCP-GUI アプリケーションでログをリプレイして、システムのライブデータとともに過去のデータを視覚化したりできます。

ログファイルを表示するために PCP で利用可能なツールの 1 つは **pmdumptext** です。このツールを使用すると、ユーザーは選択された PCP ログアーカイブを解析し、値を ASCII テーブルにエクスポートできます。**pmdumptext** では、アーカイブログ全体をダンプしたり、コマンドラインで個別のメトリックを指定してログからメトリック値のみを選択したりできます。**pmdumptext** の使用の詳細については、**pmdumptext**(1) man ページを参照してください。

## A.7. 視覚的なトレース (PCP-GUI および PMCHART を使用)

PCP-GUI パッケージを使用すると、**pmchart** グラフィカルユーティリティーを使用してパフォーマンスメトリック値をグラフにプロットできます。**pmchart** では、複数の表を同時に表示できます。メトリックは 1 つまたは複数のライブホストから取得され、履歴データのソースとして PCP ログアーカイブからメトリックデータを使用する別のオプションが提供されます。

**pmchart** を開くと、PCP チャート GUI が表示されます。その GUI の最下部には **pmtime** の VCR に似たコントロールが表示されます。開始/一時停止ボタンを使用すると、メトリックデータをポーリングする間隔と、履歴データを使用している場合は、メトリックの日付と時刻を制御できます。

ツールバーの **File -> New Chart** オプションから、ホスト名またはアドレスを指定し、リモートホストからパフォーマンスメトリックを選択することにより、ローカルマシンとリモートマシンの両方からメトリックを選択できます。高度な設定オプションには、表の軸の値を手動で設定したり、プロットの色を手動で選択したりする機能が含まれます。

**pmchart** で作成されたイメージを取得したり、ビューを記録したりする複数のオプションがあります。現在のビューのイメージを保存するには、ツールバーの **File -> Export** オプションを選択します。記録を開始するにはツールバーの **Record -> Start** オプションを選択し、記録を終了するには **Record -> Stop** を選択します。記録が終了すると、記録されたメトリックはアーカイブされ、後で参照できるようになります。

**pmchart** インターフェースをカスタマイズして、パフォーマンスメトリックのデータを折れ線グラフ、

棒グラフ、使用量グラフなどの複数の方法で表示することができます。**pmchart** では、「ビュー」と呼ばれる主要な設定ファイルにより、1 つまたは複数のグラフに関連付けられたメタデータを保存できるようになります。このメタデータは、使用されるメトリックとグラフ列を含むグラフのすべての側面を定義します。カスタム「ビュー」設定は **File -> Save View** を選択して保存し、後で再びロードできます。ビュー設定ファイルとその構文の詳細については、**pmchart**(1) man ページを参照してください。

以下の **pmchart** ビュー設定の例では、**gfs2.glocks** メトリックを使用してマウント済みの GFS2 ファイルシステム **loop1** の glock の合計数を示す積み重ねグラフを定義しています。また、以下の例の下部では、同じファイルシステムインスタンス「loop1」に対する glock の許可、降格、およびキューへの格納の要求の平均レイテンシーをプロットするプロットグラフも定義しています。

```
#kmchart
version 1

chart title "Total number of Glocks /loop1" style stacking antialiasing off
    plot legend "Shared"   metric gfs2.glocks.shared    instance "loop1"
    plot legend "Unlocked" metric gfs2.glocks.unlocked  instance "loop1"
    plot legend "Deferred" metric gfs2.glocks.deferred  instance "loop1"
    plot legend "Exclusive"metric gfs2.glocks.exclusive instance "loop1"

chart title "Average Glock Latency (usecs) /loop1" style plot antialiasing off
    plot legend "Demote" metric gfs2.latency.demote.all instance "loop1"
    plot legend "Grant"  metric gfs2.latency.grant.all  instance "loop1"
    plot legend "Queue"  metric gfs2.latency.queue.all  instance "loop1"
```

# 付録B GFS2 トレースポイントおよび DEBUGFS GLOCKS ファイル

本付録は、glock **debugfs** インターフェースおよび GFS2 トレースポイントについて記載しています。内容は、ファイルシステムの内部について精通しており、GFS2 の設計および GFS2 固有の問題のデバッグ方法についての知識を更に深めたいとお考えの上級ユーザー向けとなっています。

## B.1. GFS2 トレースポイントのタイプ

GFS2 トレースポイントには、*glock* ("ジーロック" と発音) トレースポイント、*bmap* トレースポイント、および *log* トレースポイントの 3 つのタイプがあります。これらのトレースポイントは、実行中の GFS2 ファイルシステムのモニタリングに使用することができ、Red Hat Enterprise Linux の旧リリースでサポートされているデバッグオプションで取得可能な情報に対する追加情報を提供します。トレースポイントは、ハングやパフォーマンス上の問題が再現可能で、問題のある操作の実行中にトレースポイントの出力が取得可能である場合に、特に有用です。GFS2 では、glocks は主要なキャッシュ制御メカニズムであり、GFS2 の中核のパフォーマンスを理解するためのカギです。bmap (block map) トレースポイントは、ブロック割り当ておよびブロックマッピング (オンディスクメタデータツリーの割り当て済みブロックのルックアップ) を発生と同時にモニタリングし、アクセスのローカリティに関連した問題をチェックするのに使用することができます。log トレースポイントはジャーナルに書き込まれるデータおよびジャーナルからリリースされるデータを記録し、GFS2 のその部分に関する有用な情報を提供することができます。

トレースポイントは可能な限り汎用的となるように設計されています。これにより、Red Hat Enterprise Linux 7 では API を変更する必要はなくなるはずです。このインターフェースのユーザーは、これが通常の Red Hat Enterprise Linux 7 API セットの一部ではなく、デバッグ用のインターフェースであることに注意してください。したがって、Red Hat は、GFS2 トレースポイントインターフェースが変更されないことを保証しません。

トレースポイントは Red Hat Enterprise Linux 7 の汎用機能であり、その対象範囲は GFS2 に限定されません。特に、トレースポイントは **blktrace** インフラストラクチャーの実装に使用されます。また、**blktrace** トレースポイントを GFS2 のトレースポイントと併用してシステムパフォーマンスの詳細を把握することができます。トレースポイントが機能するレベルにより、非常に短時間で大量のデータを作成することができます。トレースポイントは、有効化された時のシステムへの負荷が最小限となるように設計されていますが、ある程度の影響は避けられません。さまざまな方法でイベントをフィルタリングすることにより、データ量を削減し、特定の状況を理解するにあたって有用な情報のみを取得できます。

## B.2. トレースポイント

トレースポイントは **/sys/kernel/debug/tracing/** ディレクトリーにあります (**debugfs** が標準の場所である **/sys/kernel/debug** ディレクトリーにマウントされていることを前提)。**events** サブディレクトリーには、指定可能なすべてのトレーシングイベントが格納されます。また、**gfs2** モジュールがロードされている場合は、**gfs2** サブディレクトリーに、下位サブディレクトリーが GFS2 イベントごとに 1 つずつ格納されます。多くの場合、**/sys/kernel/debug/tracing/events/gfs2** の内容は以下のようになります。

```
[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
```

GFS2 トレースポイントをすべて有効にするには、以下のコマンドを実行します。

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

特定のトレースポイントを有効化するために、各イベントサブディレクトリーに **enable** ファイルがあります。また、各イベントまたはイベントセットを対象にイベントフィルターを設定するのに使用できる **filter** ファイルの場合も同じです。各イベントの意味については、下記に詳しく説明しています。

トレースポイントからの出力は、ASCII またはバイナリー形式で提供されます。本付録では、現時点でバイナリーインターフェースについて説明しません。ASCII インターフェースは 2 つの方法で利用することができます。以下のコマンドを実行すると、現在のリングバッファーの内容を一覧表示することができます。

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

このインターフェースは、ある一定の期間に長時間実行されるプロセスを使用する場合や、何らかのイベントの後にバッファー内にキャプチャーされている最新の情報を確認したい場合に役立ちます。もう 1 つのインターフェースは **/sys/kernel/debug/tracing/trace_pipe** であり、すべての出力が必要な場合に使用することができます。イベントは、発生するときにこのファイルから読み取られます。このインターフェースでは履歴情報は提供されません。出力の形式は両インターフェースとも同じであり、 本付録の後半で GFS2 イベント別に説明しています。

トレースポイントのデータの読み取りには、**trace-cmd** と呼ばれるユーティリティを利用することができます。このユーティリティについての更なる詳しい情報は、「リファレンス」 に記載のリンクを参照してください。**trace-cmd** ユーティリティは **strace** ユーティリティと同様に使用することができ、様々なソースからトレースデータを収集している間にコマンドを実行することが可能です。

## B.3. GLOCKS

GFS2 を理解するにあたって、把握しておくべき最も重要なコンセプトであり、GFS2 と他のファイルシステムを差別化しているのは、glock の概念です。ソースコードの観点から言えば、glock は、DLM とキャッシュ機能を単一のステートマシンにまとめて組み込むデータ構造です。各 glock は、単一 DLM ロックとの間に 1:1 のリレーションシップがあり、そのロック状態用のキャッシュ機能を提供します。このため、ファイルシステムの単一ノードから繰り返し実行される操作で DLM を何度も呼び出す必要がなく、不要なネットワークトラフィックを回避するのに役立ちます。glock には大きく 2 つに分類され、メタデータをキャッシュするものと、そうでないものに分かれます。inode glocks およびリソースグループ glocks はいずれもメタデータをキャッシュしますが、それ以外のタイプの glock はメタデータをキャッシュしません。inode glock は、メタデータ以外にデータのキャッシュにも関与しており、全 glock の中でもロジックが最も複雑です。

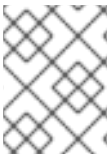
**表B.1 glock モードおよび DLM ロックモード**

| glock モード | DLM ロックモード | 注意事項 |
|---|---|---|
| UN | IV/NL | ロック解除されています (I フラグに依存する glock または NL ロックに関連付けされた DLM ロックはなし) |
| SH | PR | 共有 (読み取り保護) ロック |
| EX | EX | 排他ロック |
| DF | CW | ダイレクト I/O およびファイルシステムフリーズで使用される遅延 (同時書き込み) |

glock は (他のノードからの要求または仮想マシンから要求を受けて) ロックが解除されて、ローカルユーザーがいなくなるまでメモリーに残ります。その時点で、glock は glock ハッシュテーブルから削除されて解放されます。glock の作成時には、DLM ロックはその glock には即関連付けられません。DLM ロックは、DLM への初回要求時に関連付けられ、その要求が成功した場合には、その glock に 'I' (initial) フラグが設定されます。表B.4「glock フラグ」 には、各 glock フラグについての説明を記載しています。DLM が glock に関連付けられた後は、glock が解放されるまで、その DLM ロックは少なくとも NL (Null) ロックモードの状態を常時維持します。DLM ロックの NL から unlocked への降格は常に、glock の有効期間における最後の操作となります。

各 glock には多数の "ホルダー" を関連付けることができます。各ホルダーは、上位からのロック要求を表します。GFS2 に関するシステムコールは、glock からのホルダーをキュー/デキューして、コードの重要なセクションを保護します。

glock 状態のマシンはワークキューに基づきます。パフォーマンス上の理由により、タスクレットの方が望ましいですが、現行の実装ではタスクレットの使用を禁止するコンテキストから I/O を送信する必要があります。



**注記**

ワークキューには独自のトレースポイントがあり、GFS2 のトレースポイントと併用することができます。

表B.2「glock のモードとデータタイプ」 には、各 glock モードでキャッシュされる状態と、キャッシュされた状態がダーティーである可能性があるかどうかについてまとめています。これは、inode とリソースグループロックの両方に適用されます。ただし、リソースグループロックにはデータコンポーネントはなくメタデータのみです。

**表B.2 glock のモードとデータタイプ**

| glock モード | キャッシュデータ | キャッシュメタデータ | ダーティーデータ | ダーティーメタデータ |
|---|---|---|---|---|
| UN | いいえ | いいえ | いいえ | いいえ |
| SH | はい | はい | いいえ | いいえ |
| DF | いいえ | はい | いいえ | いいえ |
| EX | はい | はい | はい | はい |

## B.4. GLOCK DEBUGFS インターフェース

glock **debugfs** インターフェースは glock とホルダーの内部の状態を視覚化することができます。また、場合によっては、ロックされているオブジェクトのサマリー情報が含まれます。このファイルの各行は、インデントなしで G: (glock 自体のことを示す) から始まるか、1 文字分字下げされて他の文字で始まり、ファイル内の直前の行の glock に関連付けられた構造を示します (H: ホルダー、I: inode、R: リソースグループ)。以下は、このファイルの内容の一例です。

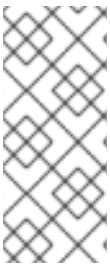
```
G:  s:SH n:5/75320 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:EX n:3/258028 f:yI t:EX d:EX/0 a:3 r:4
 H: s:EX f:tH e:0 p:4466 [postmark] gfs2_inplace_reserve_i+0x177/0x780 [gfs2]
```

```
 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]
```

上記の例は、単一ノードの GFS2 ファイルシステムで postmark benchmark を実行中に **cat /sys/kernel/debug/gfs2/unity:myfs/glocks >my.lock** のコマンドによって生成された (約 18 MB のファイルからの) 抜粋です。ここに示した glocks は、glock ダンプの興味深い特徴を示すために選択したものです。

glock の状態は EX (exclusive)、DF (deferred)、SH (shared)、UN (unlocked) のいずれかとなります。これらの状態は、DLM ロックモードに直接対応しています。ただし、DLM が null ロック状態であること、または GFS2 がロックを保持していないことのいずれかを示す UN を除きます (前述したように、I フラグによって異なります)。glock の s: フィールドは、ロックの現在の状態を示し、またホルダーの s: は要求されたモードを示します。ロックが許可された場合、ホルダーは H ビットがフラグ (f: フィールド) に設定されます。そうでない場合には、W (wait) ビットに設定されます。

n: フィールド (番号) は各アイテムに関連付けされている番号を示します。glock の場合、これはタイプの番号であり、その後に glock 番号が続きます。したがって、上記の例では、最初の glock は n:5/75320 であり、inode 75320 に関連付けられた **iopen** glock を示します。inode と **iopen** glock の場合、glock 番号は常に inode のディスクブロック番号と同じです。



**注記**

debugfs glocks ファイル内の glock 番号 (n: フィールド) は 16 進法ですが、トレースポイントの出力には 10 進法で表示されます。これは、glock 番号がかつてから常に 16 進法で記述されてきたためですが、トレースポイントの出力は、別のトレースポイント出力 (例: **blktrace**) および **stat**(1) からの出力と数値を容易に比較できるようにするために 10 進法 が選択されました。

ホルダーと glock 用のフラグの完全な一覧は 表B.4「glock フラグ」 と 表B.5「Glock ホルダーフラグ」 に示されています。LVB の内容は現在 **debugfs** インターフェースから取得できません。

表B.3「Glock のタイプ」 には、異なる glock のタイプの意味をまとめています。

**表B.3 Glock のタイプ**

| タイプ番号 | ロックタイプ | 用途 |
|---|---|---|
| 1 | trans | トランザクションのロック |
| 2 | inode | Inode のメタデータとデータ |

| タイプ番号 | ロックタイプ | 用途 |
|---|---|---|
| 3 | rgrp | リソースグループのメタデータ |
| 4 | meta | スーパーブロック |
| 5 | iopen | 最後に inode をクローズしたプロセスの検出 |
| 6 | flock | **flock**(2) syscall |
| 8 | quota | クォータ操作 |
| 9 | journal | ジャーナルミューテックス |

重要な glock フラグの 1 つに l (locked) フラグがあります。これは、glock の状態変更を実行する際に glock 状態へのアクセスを回避するために使用するビットロックです。ステートマシンが DLM を介してリモートロック要求を送信するときに設定され、完全な操作が実行された場合にのみ消去されます。場合によっては、複数のロック要求が送信されて、その間に様々な無効化が発生していることを意味します。

表B.4「glock フラグ」 には、異なる glock のフラグの意味をまとめています。

**表B.4 glock フラグ**

| フラグ | 名前 | 意味 |
|---|---|---|
| d | Pending demote | 遅延された (リモートの) 降格要求 |
| D | Demote | 降格要求 (ローカルまたはリモート) |
| f | Log flush | この glock を解放する前にログをコミットする必要があります |
| F | Frozen | リモートのノードからの返信が無視されました - リカバリが進行中です |
| i | Invalidate in progress | この glock のページの無効化が進行中です |
| I | Initial | DLM がこの glock と関連付けられる場合に指定します |
| l | Locked | glock は状態の変更中です |
| L | LRU | glock が LRU リストにあるときに設定されます |
| o | オブジェクト | glock がオブジェクト (タイプ 2 glock の場合は inode、タイプ 3 glock の場合はリソースグループ) に関連付けられた場合に設定されます。 |

| フラグ | 名前 | 意味 |
|---|---|---|
| p | Demote in progress | glock は降格要求に応答中です |
| q | キュー待ち | glock に対してホルダーがキューに格納された場合に設定され、glock が保持された場合に消去されます (ただし、残りのホルダーはありません)。glock の最小保持時間を計算するアルゴリズムの一部として使用されます。 |
| r | Reply pending | リモートノードから受信した返信の処理の待機中です |
| y | Dirty | この glock を解放する前にデータをディスクにフラッシュする必要があります |

ローカルノードで保持されているのと競合するモードでロックを要求するノードからリモートコールバックを受信すると、D (demote) または d (demote pending) のいずれか一方のフラグが設定されます。特定のロックに対する競合が発生している時にスターベーション状態を防ぐには、各ロックに最小保持時間を割り当てます。最小保持時間に達していないノードは、その期間が経過するまでロックを維持することができます。

期間が経過した場合には、D (demote) フラグが設定され、必要な状態が記録されます。その場合、次回にホルダーのキューに許可さえたロックがない場合には、そのロックは降格されます。期間が経過していない場合には、代わりに d (demote pending) フラグが設定されます。これにより、最小保持期間が経過した時にステートマシンが d (demote pending) をクリアし D (Demote) を設定するようにもスケジュールされます。

I (initial) フラグは、glock が DLM ロックに割り当てられている場合に設定されます。これは、glock が最初に使用されてから、最終的に解放される (DLM ロックが解除される) まで I フラグが設定された状態が続く場合に発生します。

## B.5. GLOCK ホルダー

表B.5「Glock ホルダーフラグ」 には、異なる glock ホルダーのフラッグの意味をまとめています。

**表B.5 Glock ホルダーフラグ**

| フラグ | 名前 | 意味 |
|---|---|---|
| a | Async | glock の結果を待ちません (結果を後でポールします) |
| A | Any | 互換性のあるロックモードはすべて受け入れ可能です |
| c | No cache | ロック解除時に DLM ロックを即時に降格します |
| e | No expire | 後続のロック取り消し要求を無視します |
| E | Exact | 完全一致するロックモードでなければなりません |
| F | First | ホルダーがこのロックに最初に許可される場合に指定します |

| フラグ | 名前 | 意味 |
|---|---|---|
| H | Holder | 要求したロックが許可されたことを示します |
| p | Priority | キューの先頭にある待機ホルダー |
| t | Try | 「try」ロックです |
| T | Try 1CB | コールバックを送信する「try」ロックです |
| W | Wait | 要求完了の待機中にセットされます |

前述したように、H (holder) および W (wait) はそれぞれ、許可されたロック要求と、キューに追加されたロック要求に設定されるので、最も重要なホルダーフラグです。一覧内でのホルダーの順序付けは重要です。許可されたホルダーがある場合は、必ずキューの先頭に配置され、キューに追加されたホルダーがその後に続きます。

許可されたホルダーがない場合には、一覧の最初のホルダーが次の状態変更をトリガーします。降格の要求は、ファイルシステムからの要求よりも常に優先度が高いとみなされるため、この要求によって必ずしも直接、要求された状態への変更が行われるわけではありません。

glock のサブシステムは、2 種類の "try" ロックをサポートします。これらのロックは、(適切な back-off および retry で) 通常の順序でのロックの取得を可能にし、他のノードが使用中のリソースを回避するために使用できるため、いずれも有用です。通常の t (try) ロックは、その名前が示すように "try" ロックであり、特別なことは何もしません。一方、T (**try 1CB**) ロックは、DLM が単一のコールバックを互換性のないロックホルダーに送信する以外は t ロックと同じです。T (**try 1CB**) ロックの使用例の 1 つに、**iopen** ロックとの併用があります。このロックは、inode の **i_nlink** 数がゼロのときにノードを判別して、どのノードが inode の割り当て解除を行っているかを判断するために使用されます。**iopen** glock は通常共有の状態で保持されますが、**i_nlink** 数がゼロになり、**->delete_inode**() が呼び出されると、T (**try 1CB**) が設定された状態で排他的ロックを要求します。ロックが許可された場合は、inode の割り当て解除を継続します。ロックが許可されない場合は、ロックの許可を阻止していたノードにより glock に D (降格) フラグが設定されます。このフラグは、割り当て解除が忘れられないように **->drop_inode**() が呼び出されたときにチェックされます。

これは、リンクカウントがゼロでありながら開いている inode が、最終の **close**() が発生するノードによって割り当て解除されることを意味します。また、inode のリンクカウントがゼロに減少するのと同時に、その inode は、リンクカウントがゼロでありながらもリソースグループビットマップで依然として使用中であるという特別の状態としてマークされます。この関数は、ext3 ファイルシステムの孤立アイテム一覧と同様に、後に続くビットマップのリーダーに、再使用可能な潜在的領域があることを知らせて、再利用を試みます。

## B.6. GLOCK のトレースポイント

トレースポイントは、**blktrace** の出力およびオンディスクレイアウトの知識と組み合わせることにより、キャッシュ制御の正確性を確認することができるようにも設計されています。これにより、任意の I/O が正しいロックで発行および完了され、競合が発生していないことをチェックできるようになります。

**gfs2_glock_state_change** トレースポイントは、最も重要な理解すべきトレースポイントです。このトレースポイントは初期作成から、**gfs2_glock_put** で終わる最終降格、最終の NL からロック解除に至るまでの遷移の全状態をトラッキングします。l (locked) glock フラグは、状態の変更が発生する前に必ず設定され、完了するまではクリアされません。状態の変更中には、許可されるホルダーはありませ

ん (H glock ホルダーのフラグ)。キューに配置されたホルダーがある場合には、常に W (waiting) 状態となります。状態の変更が完了すると、ホルダーが許可されることが可能となります。これは、l glock フラグがクリアされる前の最後の操作です。

**gfs2_demote_rq** トレースポイントは、ローカルおよびリモートの両方の降格要求を追跡します。ノードに十分なメモリーがある場合、ローカルの降格要求はほとんど発生しません。降格要求は多くの場合 **umount** を使用したり、メモリーをときどき再利用したりすることによって作成されます。リモートの降格要求数はノード間における特定の inode またはリソースグループに対する競合を示す尺度です。

**gfs2_glock_lock_time** トレースポイントは、DLM に対する要求の処理にかかった時間に関する情報を提供します。このトレースポイントと組み合わせて使用するために、ブロック (**b**) フラグが特別に glock に導入されました。

ホルダーにロックが許可されると、**gfs2_promote** が呼び出されます。これは、状態変更の最終段階もしくは glock 状態が適切なモードのロックをすでにキャッシュしているため、即時に許可できるロックが要求された場合に発生します。ホルダーがこの glock を最初に許可される場合、そのホルダーには f (first) フラグが設定されます。これは、現在リソースグループのみのよって使用されています。

## B.7. BMAP トレースポイント

ブロックマッピングとは、どのファイルシステムでも中核となるタスクです。GFS2 は、1 ブロックあたり 2 ビットの従来のビットマップベースのシステムを採用しています。このサブシステムにおけるトレースポイントの主要目的は、ブロックの割り当てとマッピングの所要時間をモニタリングすることです。

**gfs2_bmap** トレースポイントは、1 回の bmap 操作につき 2 回 (bmap 要求を表示するために起動時に 1 回と、結果を表示するために終了時に 1 回) 呼び出されます。これにより、要求と結果を容易に照合して、ファイルシステムの異なる部分や、異なるファイルオフセット、あるいは異なるファイルのブロックをマップするのに要した時間を簡単に測定することができます。また、返されたエクステントサイズの平均値と要求されたサイズを比較することもできます。

**gfs2_rs** トレースポイントは、ブロックアロケーターで作成、使用、および破棄されたブロック予約を追跡します。

割り当てられたブロックを追跡するために、割り当てに対してだけでなく、ブロックの解放に対しても **gfs2_block_alloc** が呼び出されます。割り当てはすべて、ブロックの対象の inode に応じて参照されるため、これを利用して、どの物理ブロックがライブファイルシステム内のどのファイルに属しているかを追跡できます。これは特に **blktrace** と併用した場合に有用です。これにより、問題のある I/O パターンが表示され、このトレースポイントによって取得したマッピングを使用して適切な inode を再び参照することができます。

## B.8. LOG トレースポイント

このサブシステムのトレースポイントは、ジャーナル (**gfs2_pin**) に追加された、もしくは削除されたブロックと、ログ (**gfs2_log_flush**) へのトランザクションのコミット所要時間をトラッキングします。このトレースポイントは、ジャーナリングパフォーマンスに関する問題をデバッグする際に非常に役立ちます。

**gfs2_log_blocks** トレースポイントは、ログ内で確保済みブロックを追跡します。これは、たとえばワークロードに対してログが小さすぎるかどうかを確認するのに役立ちます。

**gfs2_ail_flush** トレースポイントは、AIL リストのフラッシュの開始と終了を追跡するため、**gfs2_log_flush** トレースポイントと似ています。AIL リストには、ログに書き込まれ、まだ書き戻されていないバッファーが含まれます。このリストは、ファイルシステムが使用するログ領域をさらに解放するために定期的にフラッシュされます。また、プロセスが **sync** または **fsync** を要求した際にもフラッシュされます。

## B.9. GLOCK の統計

GFS2 ではファイルシステム内で何が起きているのか追跡する際に役立つ統計が維持されます。この統計を使用してパフォーマンス関連の問題を見つけることができます。

GFS2 では次の 2 つのカウンターが維持されます。

- **dcount**。要求された DLM 操作の回数をカウントします。平均/分散の計算に使用されたデータの量を示します。
- **qcount**。要求された **syscall** レベル操作の回数をカウントします。一般的に、**qcount** の数は **dcount** 以上になります。

また、GFS2 では 3 つの平均/分散の組み合わせが維持されます。平均/分散の組み合わせは平滑化指数関数推定値であり、使用されるアルゴリズムはネットワークコードで往復時間の計算に使用されるアルゴリズムです。GFS2 で維持される平均と分散の組み合わせはスケーリングされず、整数のナノ秒単位で表されます。

- srtt/srttvar: ブロック以外の動作の平滑化往復時間
- srttb/srttvarb: ブロック動作の平滑化往復時間
- irtt/irttvar: 内部要求時間 (例: DLM 要求間の時間)

ブロック以外の要求とは該当の DLM ロックの状態に関係なく直ちに完了する要求を指します。現在、(a) ロックの最新状態が排他的な場合の要求、(b) 要求された状態が null または unlocked のいずれかの場合の要求、(c) "try lock" フラグが設定されている場合の要求がブロック以外の要求に該当します。これ以外のすべての要求はブロック要求になります。

IRTT の場合は時間がかかる方がパフォーマンス関連の問題が発生する可能性が少なく、RTT の場合は時間が短い方がその可能性が少ないということになります。

統計は次の 2 つの **sysfs** ファイルに格納されます。

- **glstats** ファイル。1 行に 1 つの glock の統計値が格納される点以外は **glocks** ファイルと同じです。作成される glock の glock タイプの「per cpu」データでデータの初期化が行われます (ゼロとなるカウンターとは別)。このファイルは非常に大きくなる場合があります。
- **lkstats** ファイル。各 glock タイプの「1 cpu あたり」の統計値が含まれます。1 行あたり 1 つの統計が示され、各列は 1 cpu コアを示します。1 つの glock タイプは 8 行で示されます。

## B.10. リファレンス

トレースポイントおよび GFS2 **glocks** ファイルに関する詳細情報は、以下の参考情報を参照してください。

- glock 内部ロックルールの詳細は、https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystem glocks.rst を参照してください。
- イベントトレーシングの情報は、https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/trace/eve を参照してください。
- **trace-cmd** ユーティリティに関する情報は、 http://lwn.net/Articles/341902/ を参照してください。

# 付録C 改訂履歴

**改訂 4.1-1.1** Mon Sep 21 2020 Ludek Janda
翻訳ファイルを XML ソースバージョン 4.1-1 と同期

**改訂 4.1-1** Wed Aug 7 2019 Steven Levine
7.7 GA 公開用ドキュメントの準備

**改訂 3.1-1** Thu Oct 4 2018 Steven Levine
7.6 GA 公開用ドキュメントの準備

**改訂 2.1-1** Thu Mar 15 2018 Steven Levine
7.5 GA 公開用ドキュメントの準備

**改訂 2.1-0** Thu Dec 14 2017 Steven Levine
7.5 ベータ版公開用ドキュメントの準備

**改訂 1.3-6** Wed Nov 8 2017 Steven Levine
7.4 のバージョンを更新

**改訂 1.3-3** Thu Jul 27 2017 Steven Levine
7.4 GA 公開用ドキュメントバージョン

**改訂 1.3-1** Wed May 10 2017 Steven Levine
7.4 ベータ版公開用ドキュメントの準備

**改訂 1.2-5** Thu Mar 23 2017 Steven Levine
7.3 GA 公開用バージョンへの更新

**改訂 1.2-4** Mon Oct 17 2016 Steven Levine
7.3 GA 公開用バージョン

**改訂 1.2-3** Wed Aug 17 2016 Steven Levine
7.3 ベータ公開用ドキュメントの準備

**改訂 0.3-5** Mon Nov 9 2015 steven levine
7.2 GA 公開用ドキュメントの準備

**改訂 0.3-2** Wed Aug 19 2015 Steven Levine
7.2 ベータ公開用ドキュメントの準備

**改訂 0.2-13** Wed Feb 18 2015 Steven Levine
7.1 GA 公開用バージョン

**改訂 0.2-8** Thu Dec 11 2014 Steven Levine
7.0 ベータリリース向けバージョン

**改訂 0.1-29** Wed Jun 11 2014 Steven Levine
7.0 GA リリース向けバージョン

**改訂 0.1-11** Mon Dec 9 2013 Steven Levine
7.0 ベータリリース

**改訂 0.1-1** Wed Jan 16 2013 Steven Levine
ドキュメントの Red Hat Enterprise Linux 6 バージョンからのブランチ

# 索引

## シンボル

## M



## P



## Q



## W