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: 2023-03-25

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

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

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 **5**
1.1. GFS2 サポート制限 5
1.2. 新機能と変更点 6
1.2.1. Red Hat Enterprise Linux 7.0 の新機能と変更された機能 6
1.2.2. Red Hat Enterprise Linux 7.1 の新機能および変更された機能 7
1.2.3. Red Hat Enterprise Linux 7.2 の新機能および変更された機能 7
1.2.4. Red Hat Enterprise Linux 7.4 の新機能および変更された機能 7
1.3. GFS2 を設定する前に 7
1.4. GFS2 のインストール 8
1.5. RED HAT ENTERPRISE LINUX 7 での GFS2_TOOL の代替機能 8

第2章 GFS2 の設定および操作における考慮事項 **11**
2.1. フォーマットに関する考慮事項 11
2.1.1. ファイルシステムサイズ: 小さい方が好ましい 11
2.1.2. ブロックサイズ: デフォルト (4K) ブロックを推奨 11
2.1.3. ジャーナル数: マウントするノードにつき 1 つ 12
2.1.4. ジャーナルサイズ: 通常はデフォルト (128 MB) が最適 12
2.1.5. リソースグループのサイズおよび数 12
2.2. ファイルシステムの断片化 13
2.3. ブロック割り当てにおける問題 13
2.3.1. ファイルシステムに空き領域を確保する 13
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 13
2.3.3. 可能な場合には事前に割り当てる 14
2.4. クラスターに関する考慮事項 14
2.5. 使用に関する考慮事項 14
2.5.1. マウントオプション: noatime と nodiratime 14
2.5.2. VFS チューニングオプション: リサーチと実験 15
2.5.3. GFS2 上の SELinux 15
2.5.4. GFS2 上における NFS セットアップ 15
2.5.5. GFS2 上の Samba (SMB または Windows) ファイルサービス 17
2.5.6. GFS2 用仮想マシンの設定 17
2.6. ファイルシステムのバックアップ 17
2.7. ハードウェアに関する考慮事項 18
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 18
2.9. GFS2 のノードロック機能 18
2.9.1. POSIX ロックの問題 20
2.9.2. GFS2 によるパフォーマンスチューニング 20
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 21

第3章 GFS2 の管理 **25**
3.1. GFS2 ファイルシステムの作成 25
用途 25
例 26
全オプション 27
3.2. GFS2 ファイルシステムマウント 29
用途 29
例 29
完全な使用法 29
3.3. GFS2 ファイルシステムのアンマウント 32
用途 32
3.4. GFS2 のクォータ管理 33

3.4.1. ディスククォータの設定 33
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 33
3.4.1.2. クォータデータベースファイルの作成 34
3.4.1.3. ユーザーごとのクォータ割り当て 34
3.4.1.4. グループごとのクォータ割り当て 35
3.4.2. ディスククォータの管理 35
3.4.3. クォータの精度維持 36
3.4.4. quotasync コマンドを使用したクォータの同期 36
用途 37
例 37
3.4.5. リファレンス 38
3.5. GFS2 ファイルシステムの拡張 38
用途 38
コメント 38
例 39
完全な使用法 39
3.6. GFS2 ファイルシステムへジャーナルの追加 39
用途 40
例 40
完全な使用法 40
3.7. データジャーナリング 41
3.8. ATIME 更新の設定 42
3.8.1. relatime を使用したマウント 42
用途 43
例 43
3.8.2. noatime を使用したマウント 43
用途 43
例 43
3.9. GFS2 ファイルシステム上の動作の一時停止 43
用途 44
例 44
3.10. GFS2 ファイルシステムの修復 44
用途 45
例 46
3.11. GFS2 WITHDRAW 機能 46

第4章 GFS2 ファイルシステムに伴う問題の診断と修正 49
4.1. GFS2 ファイルシステムのパフォーマンス低下 49
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 49
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 49
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 50
4.5. 空のファイルシステムで使用中と表示される領域 51

第5章 クラスターでの GFS2 ファイルシステムの設定 52

付録A PERFORMANCE CO-PILOT による GFS2 パフォーマンスの分析 55
A.1. PERFORMANCE CO-PILOT の概要 55
A.2. PCP デプロイメント 56
A.3. PCP インストール 56
A.4. GFS2 パフォーマンスデータのトレース 57
A.5. メトリック設定 (PMSTORE の使用) 59
A.6. パフォーマンスデータのロギング (PMLOGGER の使用) 60
A.7. 視覚的なトレース (PCP-GUI および PMCHART を使用) 61

付録B GFS2 トレースポイントおよび DEBUGFS GLOCKS ファイル **63**
B.1. GFS2 トレースポイントのタイプ 63
B.2. トレースポイント 63
B.3. GLOCK 64
B.4. GLOCK DEBUGFS インターフェイス 65
B.5. GLOCK ホルダー 68
B.6. GLOCK のトレースポイント 69
B.7. BMAP トレースポイント 70
B.8. ログトレースポイント 70
B.9. GLOCK の統計 70
B.10. リファレンス 71

付録C 改訂履歴 **72**

索引 **72**

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、共有名前空間を提供し、共通のブロックデバイスを共有する複数のノード間での一貫性を管理します。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セットを提供すると同時に、ノード間でクラスターの完全な整合性を強制することを目的としています。一部のケースでは、Linux ファイルシステム API が、GFS2 のクラスター化した性質の完全な透明性を可能にしません。これは、たとえば、GFS2 で Posix ロックを使用しているプログラムが、**GETLK** 関数の使用を避けるべきということがあります。これは、クラスター化した環境では、プロセス ID がクラスターでの別のノードのものであることがあるためです。ただし、ほとんどの場合、GFS2 ファイルシステムの機能は、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 は GFS2 を提供します。GFS2 が必要とするクラスター管理の提供は RHEL High Availability Add-On により提供されます。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 では、ローカルファイルシステムと同様、ページキャッシュで、頻繁に使用されるデータのローカルキャッシングを行ってパフォーマンスを向上します。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。glock とそのパフォーマンスへの影響の詳細は、「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、GFS2 が現在対応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) 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ーに基づいています。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。

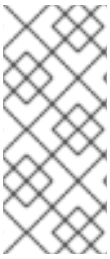
注記

GFS2 ファイルシステムはスタンドアロンシステム、またはクラスター設定で実装することが可能ですが、Red Hat Enterprise Linux 7 リリースでは、GFS2 を 1 ノードのファイルシステムとして使用することはサポートしていません。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多く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に限り、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が推奨されます。

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 (例: バックアップ) とした、単一ノードの GFS2 ファイルシステムを引き続きサポートします。

ファイルシステムのサイズを決定する際に、復旧のニーズを考慮する必要があります。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、ディスクまたはディスクサブシステムに障害が発生した場合、復旧時間はバックアップメディアの速度により異なります。**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、クラスター内のすべてのノードが共有ストレージにアクセスでき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サポート対象外のノードがある非対称クラスター設定はサポートされません。ただし、すべてのノードが実際に GFS2 ファイルシステム自体をマウントす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ます。

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

最初に作成する GFS2 ファイルシステムの数を指定します。(ファイルシステムは後で追加できます。)

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。名前は、クラスター内のすべての **lock_dlm** ファイルシステムで固有にする必要があります。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 ファイルシステムのジャーナル数を決定します。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。ジャーナルを GFS2 ファイルシステムに追加する方法は、「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) を使用することが推奨されます。

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。inode のタイムスタンプの更新を不要に行うと、クラスターのパフォーマンスに大きな影響が及びます。

注記

同じディレクトリー内の複数のノードで作成操作および削除操作が同時に多数発行すると、GFS2 でパフォーマンスの問題が発生することがあります。これによりシステムでパフォーマンスの問題が発生する場合は、ノードによるファイルの作成および削除を、可能な限りそのノード固有のディレクトリーに特定する必要があります。

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その他の推奨事項は、*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をまとめています。

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 ...* 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* 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* 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* 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。 ```# gfs2_edit -p journals /dev/clus_vg/lv1 Block #Journal Status: of 2620416 (0x27fc00) ------------------- Journal List ------------------ journal0: 0x14 128MB clean. journal1: 0x805b 128MB clean. ---```
lockdump *mountpoint* 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。 ```# cat /sys/kernel/debug/gfs2/*clustername:file_system_name*/glocks```

sb *device* proto [*newvalue*]
 	
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。
	
	```# tunegfs2 -l *device* | grep protocol```
	
	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。
	
	```# tunegfs2 -o lockproto=lock_dlm *device*```

sb *device* table [*newvalue*]
 	
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。
	
	```# tunegfs2 -l *device* | grep table```
	
	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。
	
	```# tunegfs2 -o locktable=*file_system_name* *device*```

sb *device* ondisk [*newvalue*] **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*] **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*] **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。 # tunegfs2 -l *device* | grep UUID **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。 # tunegfs2 -U *uuid device*
sb *device* all GFS2 スーパーブロックを出力する	# tunegfs2 -l *device*
setflag *Flag File1 File2 ...* 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* 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* 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 gfs2_tool コマンドのバージョンを表示する	該当なし
withdraw *mountpoint* 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。このロックスキームは、TCP/IP などの通信プロトコルを使用して、ロック情報を交換します。

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 の導入がお客様のニーズを満たし、サポート対象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、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ーに基づいています。ただし、64 ビットハードウェア用で現在対応している GFS2 ファイルシステムの最大サイズは 100 TB です。

GFS2 のファイルシステムは大きくても構いませんが、推奨されているわけではありません。GFS2 の経験則から、小さい方がよいとされています。10 TB のファイルシステムを 1 つ用意するより、1 TB のファイルシステムを 10 個用意するほうがよいとされています。

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ると、容量が不足し、それ自体に影響が及ぶ可能性があります。サイズを決定する前に、どのように使用されるかを考慮してください。

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、4K が Linux のデフォルトのページサイズ (メモリー) であるためです。その他の一部のファイルシステムとは異なり、GFS2 は 4K カーネルバッ

ファーを使用して多くの操作を実行します。ブロックサイズが 4K であれば、カーネルによるバッファー操作の作業数が減ります。

パフォーマンスを最大にするためにも、デフォルトのブロックサイズを使用することが推奨されます。小さいファイルが大量にある効率的なストレージが必要な場合のみ、別のブロックサイズを使用してください。

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、ファイルシステムのマウントを必要とするクラスターの各ノードにジャーナルが必要になります。たとえば、16 ノードのクラスターがあり、2 つのノードからファイルシステムのみをマウントする必要がある場合は、2 つのジャーナルのみが必要になります。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。サイズを指定しないと、デフォルトの 128 MB になります。これは、ほとんどのアプリケーションに最適です。

一部のシステム管理者は、128 MB では過剰と考え、ジャーナルのサイズを最低レベルの 8 MB まで、あるいはより従来的な 32MB まで縮小することを望んでいます。動作に問題が起きない場合でも、パフォーマンスに重大な影響を与える可能性があります。多くのジャーナリングファイルシステムと同様に、GFS2 がメタデータを書き込むたびに、メタデータの配置前にジャーナルにコミットされます。これにより、システムがクラッシュしたり、停電したりした場合に、ジャーナルがマウント時に自動的に再生される時に、すべてのメタデータが復元します。ただし、ファイルシステムのアクティビティーでは 8 MB のジャーナルが簡単に埋まってしまいます。ジャーナルがいっぱいになると、GFS2 が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ます。

通常、デフォルトのジャーナルサイズである 128 MB を使用することが推奨されます。ファイルシステムが非常に小さい (例: 5GB) 場合、128 MB のジャーナルでは非現実的である可能性があります。より大きなファイルシステムがあり、容量に余裕があれば、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可能性があります。

2.1.5. リソースグループのサイズおよび数

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。最適なリソースグループサイズ (32MB から 2GB) の推定値を計算しようとします。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のサイズは、ファイルシステムの使用方法によって異なります。どの程度いっぱいになるか、または著しく断片化されるかどうかを考慮してください。

どのサイズが最適なパフォーマンスになるかを確認するには、異なるリソースグループのサイズで実験する必要があります。GFS2 を完全な実稼働環境にデプロイする前に、テストクラスターを試すのがベストプラクティスと言えます。

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。ファイルシステムが満杯になるほど、検索されるリソースグループが増え、そのすべてにクラスター全体のロックが必要になります。その結果、パフォーマンスが低下します。

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、2 GB の 5 つのリソースに分けられた 10GB のファイルがある場合、クラスター内のノードは、同じファイルシステムが 32 MB の 320 個のリソースグループに分けられている場合よりも頻繁に 5 つのリソースグループを奪おうとします。空きブロックを持つリソースグループを見つける前に、各ブロックの割り当てが複数のリソースグループを参照する必要があるため、ファイルシステムがほぼ満杯になると、この問題は悪化します。GFS2 は、次のいずれかの方法でこの問題を軽減しようとします。

- 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。ファイルを削除しなければ、競合は少なくなります。ただし、アプリケーションがブロックを継続的に削除し、ほぼ満杯のファイルシステムに新しいブロックを割り当てる場合は、競合が非常に高くなり、パフォーマンスに深刻な影響を及ぼします。

- 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、その中のノードがすべて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 用のデフラグツールはありませんが、個々のファイルを **filefrag** ツールで識別して一時ファイルにコピーし、一時ファイルの名前を変更してオリジナルを置き換えると、個々のファイルをデフラグできます。

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 ファイルシステムのブロック割り当てに関連する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は、割り当てる新しいブロックの領域の検索をうまく処理できなくなります。その結果、アロケータにより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ファイルの断片化が発生する可能性が非常に高い小さなスライスに絞り込まれる傾向があります。このファイルの断片化により、パフォーマンスの問題が発生することがあります。また、GFS2 ファイルシステムがほぼ満杯になると、GFS2 ブロックアロケーターは複数のリソースグループを検索するのにより多くの時間を費やし、十分な空き領域があるファイルシステムには必ずしも存在しないロック競合を追加します。また、パフォーマンスの問題が発生する可能性もあります。

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% 以上が使用済みのファイルシステムを実行しないことが推奨されます。

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 1 つのノードにより割り当てられ、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、ロック競合が多くなります。

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、リソースをロックする最初のノード (ファイルなど) が、そのロックのロックマスターになります。他のノードはそのリソースをロックしますが、最初にロックマスターからパーミッションを要求する必要があります。各ノードは、どのロックがロックマスターであるかを認識します。また、各ノードは、どのノードにロックを課しているのかを認識します。マスターノード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ノードのロックをロッキングするよりもはるかに高速です。

多くのファイルシステムと同様、GFS2 アロケーターは、ディスクヘッドの動きを抑え、パフォーマンスを向上させるために、同じファイルのブロックを互いに近づけようと試みます。ブロックをファイルに割り当てるノードは、新しいブロック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内のすべてのブロックが使用中の場合を除きます)。ファイルを含むリソースグループのロックマスターがデータブロックを割り当てると、ファイルシステムの実行が速くなります (ファイルを最初に開いたノードが新しいブロックの全書き込みを実行する方が速くなります)。

2.3.3. 可能な場合には事前に割り当てる

ファイルを事前に割り当てた場合は、ブロックの割り当てを完全に回避し、ファイルシステムをより効率的に実行できます。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

システムに含まれるノード数を決定する際には、高可用性とパフォーマンスにトレードオフがあることに注意してください。ノードの数が多いと、ワークロードのスケールがますます困難になります。このため、Red Hat は、ノードの数が 16 を超えるクラスターファイルシステムデプロイメントでの GFS2 の使用に対応していません。

クラスターファイルシステムをデプロイすることは、単一ノードデプロイメントのドロップインには代わりません。Red Hat では、システムをテストして必要なパフォーマンスレベルで機能させるためにも、新規インストールにおいて約 8 〜12 週のテストを行うことを推奨しています。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、**noatime** 引数および **nodiratime** 引数を使用して GFS2 ファイルシステムをマウントすることが推奨されます。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) と呼ばれるレイヤーの上にあります。**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、**dirty_background_ratio** および **vfs_cache_pressure** の値は、状況に応じて調整できます。現在の値を取得するには、次のコマンドを使用します。

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

次のコマンドは、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つけるには、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、強制モードで SELinux を動作させているシステム上であっても、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、man ページの **mount**(8) で説明されているように **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。SELinux は、ファイルシステムのすべての内容が、**context** マウントオプションによる **seclabel** 要素でラベル付けされると想定します。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。メモリー内に残り、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、**public_content_rw_t** ラベルまたは **public_content_t** ラベルを使用するか、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 ファイルシステムの管理には常に 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* の説明に従って、GFS2 ファイルシステムリソースを作成する際に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 ロッキングサブシステムとそのクラスターの複雑性を考慮し、GFS2 での NFS の設定には、多くの予防措置をとり、注意深く設定を行う必要があります。本セクションでは、GFS2 ファイルシステムで NFS サービスを設定する際に考慮すべき注意事項を説明します。

警告

GFS2 ファイルシステムが NFS でエクスポートされる場合は、**localflocks** 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。**localflocks** オプションを指定すると、複数の場所から GFS2 ファイルシステムに安全にアクセスできなくなるため、GFS2 を複数のノードから同時にエクスポートすることはできません。そのため、この設定を使用する際に、GFS2 ファイルシステムを一度に 1 つのノードにのみマウントすることが対応要件となります。この目的は、各サーバーからの POSIX ロックをローカル (つまり、クラスター化されず、相互に独立した状態) として強制的に設定することです。これは、GFS2 が NFS からクラスターのノード全体で POSIX ロックを実装しようとすると、複数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、2 台のクライアントが異なるサーバーからマウントしている場合は、ローカライズされた POSIX ロックにより、それら 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ます。これにより、データが破損する可能性があります。すべてのクライアントがあるサーバーから NFS をマウントすると、同じロックを個別サーバーに付与するという問題が発生しなくなります。**localflocks** オプションでファイルシステムをマウントするかどうか不明な場合は、このオプションを使用しないでください。データの損失を回避するためにも、すぐに Red Hat サポートに問い合わせ、適切な設定について相談してください。NFS 経由で GFS2 をエクスポートすることは、一部の状況において技術的には可能ですが推奨されません。

NFS を除くすべての GFS2 アプリケーションでは、**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 のロックと 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 ファイルシステムで 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/パッシブ設定のロックを備えた 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、1 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/パッシブ設定)。
  - NFS サーバー経由を **除いて**、GFS2 ファイルシステムへのアクセスは許可されていません。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 Samba によるアクセスの両方が含まれます。マウント元のクラスターノードからローカ

ルにファイルシステムにアクセスすると、データが破損する可能性があります。

- システムで NFS クォータはサポートされていません。

この設定では、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。

- GFS2 の NFS エクスポートには NFS オプション **fsid=** が必須です。
- クラスターで問題が発生した場合 (たとえば、クラスターが定足化し、フェンシングが成功しなくなるなど) は、クラスター化論理ボリュームと GFS2 ファイルシステムがフリーズし、クラスターが定足化されるまでアクセスできなくなります。この手順で定義されているような単純なフェイルオーバーソリューションがシステムにとって最も適しているかどうかを判断する際には、これを考慮してください。

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/アクティブ設定が可能な 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 の外部から Samba 共有のデータへの同時アクセスには対応していません。現在、GFS2 クラスターリースはサポート外で、Samba ファイル処理の速度が低下します。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オフにするために各ノードの仮想マシンの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、**libvirt** ドメインで **cache** および **io** の設定を追加すると、GFS2 が期待通りに動作するようになります。

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

代わりに、デバイス要素に **shareable** 属性を設定できます。これは、デバイスがドメイン間で共有される必要があることを示しています (ハイパーバイザーと OS が対応している場合)。**shareable** を使用すると、そのデバイスに **cache='no'** が指定されます。

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係なく、緊急時に備えて GFS2 ファイルシステムのバックアップを定期的に作成することが重要です。多くのシステム管理者は、RAID、マルチパス、ミラーリング、スナップショット、その他の形式の冗長性で保護されているため安全だと感じていますが、安全性は十分ではありません。

ノードまたはノードセットのバックアップを作成するプロセスには通常、ファイルシステム全体を順番に読み取る必要があるため、バックアップの作成に問題が生じる可能性があります。シングルノードからこれを行うと、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、そのノードはキャッシュ内のすべての情報を保持します。クラスターの稼働中にこのタイプのバックアッププログラムを実行すると、パフォーマンスが低下します。

バックアップの完了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/キャッシュの所有権を再取得するのに必要な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、キャッシュしていたデータのキャッシュを他のノードが停止するため、これは理想的ではありません。

バックアップ完了後に次のコマンドを実行すると、キャッシュを削除できます。

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

タスクがノード間で分割されるように、クラスターの各ノードがそれ自体のファイルのバックアップを作成する方が高速です。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。

Red Hat では、SAN 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にバックアップを行うことで、GFS2 バックアップを作成することを推奨しています。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 FCoE (Fibre Channel over Ethernet) などの安価な共有ストレージオプションで動作しますが、キャッシュ容量が大きい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の相互接続の SAN ストレージで、品質、健全性、パフォーマンスに関する多くのテストを行っています。原則として、最初にテストしたものは常に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

  高品質かつ高速のネットワーク機器により、クラスター通信と GFS2 の実行がより高速になり、信頼性が向上します。ただし、最も高価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中には、マルチキャストパケットの受け渡しに問題があるものがあります。これは、**fcntl** ロック (flock) に渡すために使用されます。一方、安価なコモディティーネットワークスイッチは、高速で信頼性が高い場合があります。Red Hat では、完全な実稼働環境にデプロイする前に機器を試すことを推奨しています。

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。単一ノードのファイルシステムはキャッシュと共に実装されます。キャッシュは、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 (および以前の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、ここにオンディスクデータの一部が含まれている場合があります。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。

glock は、inode ごとにキャッシュを保護するため、キャッシング層を制御するのに使用されるロックが各 inode に 1 つあります。glock が共有モード (DLM ロックモード: PR) で付与されると、その glock の下のデータは、同時に 1 つまたは複数のノードにキャッシュすることができるため、すべてのノードはそのデータへのローカルアクセスを有することができます。

glock が排他モード (DLM ロックモード: EX) で許可されると、単一ノードのみがその glock でデータをキャッシュできます。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 を要求すると、DLM は、その glock 現在を保持している 1 つまたは複数のノードに、新しい要求をブロックしてロックを解除するように依頼するメッセージを送信します。glock の削除の処理は、(ほとんどのファイルシステム操作の標準では) 長くなる可能性があります。共有 glock を削除するには、キャッシュを無効にすることだけが必要となりますが、それは比較的速く、キャッシュされたデータの量に比例します。

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は、共有 glock と同様に無効化を行う必要があります。

単一ノードのファイルシステムと GFS2 の違いは、単一ノードのファイルシステムにはキャッシュが 1 つだけあり、GFS2 には各ノードに個別のキャッシュがあることです。どちらの場合も、キャッシュされたデータへのアクセスのレイテンシーの長さは同じようになりますが、別のノードが以前同じデータをキャッシュしていると、キャッシュされていないデータにアクセスする場合のレイテンシーは、GFS2 の方がかなり長くなります。

共有 glock を必要とするのは、**read** (バッファー付き)、**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** (バッファー付き)、**mkdir**、**rmdir**、**unlink** などの操作のみです。ダイレクト I/O の読み書き操作では、割り当てが行われていない場合は DF (deffered) 状態の glock が必要です。また、書き込みに割り合てが必要な場合は (つまりファイルの拡張または穴埋めには)、EX (exclusive) 状態の glock が必要です。

この場合、パフォーマンスに関する主な考慮事項が 2 つがあります。まず、読み込み専用操作は各ノードで独立して実行できるため、クラスター全体で並列処理が極めてよく機能します。次に、同じ inode へのアクセスを競うノードが複数あると、排他 glock を必要とする操作によりパフォーマンスが低下する場合があります。たとえば 「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際に排他的なロックが必要なくなります。

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 は、1 つのノードからのみ書き込みまたは変更されます。

ファイルの作成中および削除中に、ディレクトリーにエントリーを挿入したりディレクトリーからエントリーを削除すると、ディレクトリーの inode への書き込みとしてカウントされます。

比較的頻度が低い場合は、このルールを無視できます。ただし、このルールを無視しすぎると、パフォーマンスが大幅に低下します。

読み書きのマッピングがある GFS2 のファイルに **mmap**() を行い、そこからのみ読み込む場合のみ、これは読み込みとしてのみカウントされます。GFS は書き込みとしてカウントされるため、GFS2 は **mmap**() I/O でスケーラビリティー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。すべての GFS2 ユーザーは、**atime** に特定の要件がない限り、**noatime** を使用してマウントすることが推奨されます。

2.9.1. POSIX ロックの問題

POSIX ロックを使用する場合は、以下の点を考慮する必要があります。

- flock を使用すると、POSIX ロックを使用するより処理が速くなります。
- GFS2 で Posix ロックを使っているプログラムは、**GETLK** の機能を使用しないようにする必要があります。これは、クラスター環境では、プロセス ID がクラスター内の別のノードに対するものである可能性があるためです。

2.9.2. GFS2 によるパフォーマンスチューニング

通常は、面倒な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。

面倒なアプリケーションの典型的な例は、メールサーバーです。このアプリケーションは多くの場合、各ユーザーの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のファイルを含む各ユーザーのディレクトリー (**maildir**) に配置されます。要求が IMAP 経由で到達する場合は、各ユーザーに特定のノードへのアフィニティーを与えることが理想的です。このようにして、電子メールメッセージの表示や削除の要求は、そのノードのキャッシュから提供される傾向があります。当然ながら、そのノードに障害が発生すると、セッションを別のノードで再起動できます。

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。この設計は、通常のケースで 1 つのノードにキャッシュされた特定のファイルセットを維持することを目的としていますが、ノードに障害が発生した時にはダイレクトアクセスを許可します。

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。

バックアップも、扱いにくい分野です。繰り返しますが、可能であれば、各ノードのワーキングセットを、その特定の inode のセットをキャッシュしているノードから直接バックアップを作成することが強く推奨されます。通常の時点で実行するバックアップスクリプトがあり、GFS2 で実行しているアプリケーションの応答時間が急に増大した思われる場合は、クラスターがページキャッシュを最も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。一方、バックアップが 1 つのノードのみから実行する場合は、バックアップ完了後にそのノード上にファイルシステムの大部分がキャッシュされ、他のノードからの後続アクセスのパフォーマンスが低下します。次のコマンドを実行すると、バックアップ完了後にバックアップノード上の VFS ページキャッシュをドロップすることで、パフォーマンスの低下をある程度緩和できます。

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

ただし、この方法は、各ノードのワーキングセットが共有されているか、大半がクラスター全体で読み取り専用であるか、または 1 つのノードから大部分がアクセスされるようにするのと比べると、あまり良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は、I/O の待機時間が大幅に長くなることがあります。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。

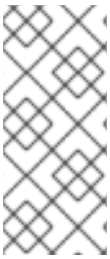
本セクションでは、GFS2 ロックダンプの概要を説明します。GFS2 ロックダンプの詳細は、付録B *GFS2 トレースポイントおよび debugfs glocks ファイル* を参照してください。

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
```

ファイルのコンテンツは一連の行になります。G: で始まる行はそれぞれ 1 つの glock を表し、それに続く 1 行のスペースでインデントされた行は、ファイル内で直前の glock に関する情報の項目を表します。

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 番号に関する 2 つのトレースの所有者情報を比較することで、ワークロードが進行中である (遅いだけ) か、それとも動かなくなったか (この場合は常にバグが原因であるため、Red Hat サポートに報告する必要があります) 判断できます。

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。ホルダーの行 f: の flags フィールドは、W フラグが待機要求を参照し、H フラグが許可された要求を参照しています。待機要求が多数ある glock は、特定の競合が発生している可能性があります。

表2.1「glock フラグ」 は、さまざまな glock フラグの意味を示しており、表2.2「glock ホルダーフラグ」 は、glock ホルダーのさまざまなフラグの意味を示します。

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、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、降格操作および try ロックに対して消去されます。このフラグの目的は、その他のノードがロックを降格する時間とは無関係に、DLM 応答時間の統計を収集できるようにすることです。
d	Pending demote	遅延している (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す (復旧が進行中です)。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 の下でページを無効にする過程です。
I	Initial	DLM ロックがこの glock と関連付けられる場合に指定します。
l	Locked	glock は、状態を変更中です。
L	LRU	glock が LRU 一覧にあるときに設定します。
o	Object	glock がオブジェクトに関連付けられている (つまり、タイプ 2 の glock の場合は inode、タイプ 3 の glock の場合はリソースグループ) ときに設定されます。
p	Demote in progress	glock は、降格要求に応答中です。
q	Queued	ホルダーが glock にキューイングされると設定され、glock が保持されるとクリアされますが、残りのホルダーはありません。アルゴリズムの一部として使用され、glock の最小保持時間を計算します。
r	Reply pending	リモートノードから受信した返信の処理を待機中です。
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 (結果は後でポーリングします)。
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。

問題の原因になっている glock を特定したら、それがどの inode に関連しているかを調べることが次のステップになります。glock 番号 (G: 行の n:) はこれを示します。*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 タイプ」 は、さまざまな glock タイプの意味を示しています。

表2.3 glock タイプ

タイプ番号	ロックタイプ	使用方法
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	使用方法
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	最後に閉じた inode 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

識別された glock のタイプが異なれば、それはタイプ 3: (リソースグループ) である可能性が最も高いです。通常の負荷下において他のタイプの glock を待機しているプロセスが多数ある場合は、Red Hat サポートに報告してください。

リソースグループロックでキューに置かれている待機要求が多く表示され場合は、複数の理由が考えられます。1 つは、ファイルシステム内のリソースグループ数と比べ、多くのノードが存在するためです。または、ファイルシステムがほぼ満杯になっている可能性もあります (平均して、空きブロックの検索時間が長い場合があります)。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 セクションで設定されます。

- 「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、アクティブ化された LVM ボリュームに作成されます。**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*** パラメーターの指定時に必要)
- 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 つに対してジャーナルが 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」 に記載されているとおり、**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。

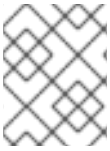
用途

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 および ***LockTableName*** パラメーターの使用方法をよく理解しておく必要があります。***LockProtoName*** および ***LockTableName*** パラメーターを不適切に使用すると、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、***ClusterName:FSName*** のように、コロンで区切られた 2 つの要素 (スペースなし) で設定されます。

- ***ClusterName***: 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。
- ***FSName***: ファイルシステムの名前。1 - 16 文字まで指定できます。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ごとに、ジャーナルが 1 つ必要になります。「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

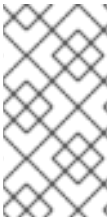
表3.1「コマンドオプション: **mkfs.gfs2**」 で、**mkfs.gfs2** コマンドオプション (フラグおよびパラメーター) を説明します。

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。最小サイズは 8 メガバイトです。ジャーナルのサイズが大きいほどパフォーマンスが向上しますが、使用するメモリーが、小さいジャーナルよりも多くなります。
-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ごとに、ジャーナルが 1 つ必要になります。このオプションを指定しない場合は、ジャーナルが 1 つ作成されます。GFS2 ファイルシステムでは、ファイルシステムを拡張せずに、後でジャーナルを追加できます。
-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。認識されるロックプロトコルには次のものがあります。 **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。 **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小サイズは 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くなると、非常に大きなファイルシステムで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。 このパラメーターは、***ClusterName:FSName*** のように、コロンで区切られた 2 つの要素 (スペースなし) で設定されます。 ***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。 ***FSName*** は、ファイルシステム名です。名前は、長さが 1 文字から 16 文字までで、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。
-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 を使用して実稼働環境で 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、かつクラスター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)。以上の要件を満たすと、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できます。

GFS2 ファイルシステムが適切に動作するには、GFS 2 ファイルシステムをマウントするすべてのノードに **gfs2-utils** パッケージをインストールする必要があります。**gfs2-utils** パッケージは、Resilient Storage チャンネルで提供されます。

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。

用途

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 ファイルシステムを置く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)、または使用できる標準の Linux **mount -o** オプション、もしくはその両方の組み合わせで設定されています。複数の ***option*** パラメーターはコンマで区切ります (スペースは使用しません)。

注記

mount コマンドは Linux のシステムコマンドです。本セクションで説明する GFS2 固有のオプションのほかにも、標準の **mount** コマンドオプション (**-r** など) を使用できます。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 に渡すことができる GFS2 固有の **-o option** 値を説明します。

注記

この表では、ローカルファイルシステムでのみ使用されるオプションを説明します。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、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 (例: バックアップ) とした、単一ノードの GFS2 ファイルシステム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 を操作できるようにします。 **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** が設定されていると、トランザクションにより変更し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。これにより、クラッシュ後に、初期化されていないブロックがファイルに表示されなくなります。**data=writeback** モードが設定されていると、そのユーザーのデータがダーティーになるとディスクに書き込まれます。ここでは、**ordered** モードと同じような一貫性保証は行わないため、ワークロードによって少し速くなります。デフォルト値は **ordered** モードです。
ignore_local_fs **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** フラグが自動的に有効になります。

オプション	説明
localflocks **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 (仮想ファイルシステム) レイヤーが flock および fcntl をすべて実行するように、GFS2 に命令します。**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*** を指定しない場合、ロックプロトコル名は、ファイルシステムのスーパーブロックから読み取られます。
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とオフを切り替えます。クォータを **account** 状態に設定すると、UID ごとまたは GID ごとの使用統計がファイルシステムにより正しく管理され、limit と warn の値は無視されます。デフォルト値は **off** です。
errors=panic|withdraw	**error=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発生します。**errors=withdraw** を指定 (デフォルトの動作) は、ファイルシステムのエラーによりシステムはファイルシステムから撤退し、次にシステムを再起動するまでアクセスできなくなります。場合によっては、システムは稼働したままになります。
discard/nodiscard	GFS2 が、解放されたブロックの破棄 I/O 要求を生成します。これを適切なハードウェアで使用して、シンプロビジョニングや同様のスキームを実装できます。
barrier/nobarrier	ジャーナルのフラッシュ時に GFS2 が I/O バリアを送信します。デフォルト値は **on** です。基となるデバイスが I/O バリアに対応していないと、このオプションは自動的に **off** になります。ブロックデバイスが、書き込みキャッシュの内容を失わないように設計されていない場合 (UPS 上にある場合や、書き込みキャッシュがない場合など) は常に、GFS2 で I/O バリアを使用することが強く推奨されます。
quota_quantum=*secs*	クォータ情報の変更が、クォータファイルに書き込まれる前にノードに留まる秒数を設定します。これは、このパラメーターの設定に推奨される方法です。この値には、ゼロよりも大きい秒数 (整数) を設定します。デフォルトは 60 秒です。短く設定すると、遅延クォータ情報の更新が速くなり、そのクォータを超過する可能性が低くなります。長く設定すると、クォータに関連するファイルシステムの操作が速くなり、より効率的になります。

オプション	説明
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** 変更がマスターの **statfs** ファイルに同期されるまでの最大期間を設定します。速いけど正確さが劣るように、もしくは遅くて正確さが上げるように **statfs** の値を調整できます。このオプションを 0 に設定すると、**statfs** は、常に true 値を報告します。
statfs_percent=*value*	その期間が経過していなくても、マスターの **statfs** ファイルに同期するまでの、ローカルベースでの **statfs** 情報の最大変化率の上限を設定します。**statfs_quantum** の設定が 0 の場合は、この設定が無視されます。

3.3. GFS2 ファイルシステムのアンマウント

Pacemaker を介して自動的で行わずに、手動でマウントした 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のマウントが解除されても、システムにより認識されません。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。このマウントの解除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マウントを解除する際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際に常に Pacemaker を使用します。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。
- 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。

このような状況でのシステムのシャットダウンで、ファイルシステムのマウントを解除しているときにハングする場合は、ハードウェアを再起動します。ファイルシステムはシャットダウンプロセスの早い段階で同期されるため、データが失われることはほとんどありません。

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。

用途

```
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可能なファイルシステム領域のサイズを制限するために使用されます。ユーザーまたはグループには、クォータ制限が設定されないと、クォータ制限がありません。GFS2 ファイルシステムが **quota=on** オプションまたは **quota=account** オプションでマウントされていると、GFS2 は、制限がない場合でも、各ユーザーおよび各グループが使用してい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量を再構築しなくてもいいように、トランザクション形式でクォータ情報を更新します。

パフォーマンス低下を防ぐためにも、GFS2 ノードは定期的にクォータファイルに更新を同期します。ファジークォータアカウンティングでは、ユーザーまたはグループは、設定された制限を若干超えることができます。これを最小限に抑えるために、GFS2 はハードクォータの制限に近づくと同期期間を動的に短縮します。

注記

GFS2 は、標準の Linux クォータ機能に対応しています。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 でクォータを管理するのに推奨される方法であるため、クォータを使用した GFS2 の新規デプロイメントには必ず使用してください。本セクションでは、この機能を使用した GFS2 クォータ管理を説明します。

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、以下の手順を行います。

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、このポリシーは強制されません。)

この各ステップは、以下のセクションで詳しく説明します。

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、クォータはデフォルトで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次のようにマウントします。

limit と warn の値を適用せずに、ディスクの使用状況を追跡し、各ユーザーおよび各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** 引数に **quota=off** を指定します。たとえば、次のコマンドは、作成している GFS2 の **Filesystem** リソースが、クォータが有効になっている状態でマウントされることを指定します。

```
# 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** 引数に **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

クォータが有効なファイルシステムがマウントされると、システムがディスククォータを操作できるようになります。ただし、ファイルシステム自体がクォータに対応するようにするには、追加の設定が必要です。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。

ファイルシステムにクォータファイルを作成するには、**quotacheck** コマンドに **-u** オプションおよび **-g** オプションを指定します。ユーザーおよびグループのクォータを初期化するには、両方の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。(**quota=account** オプションを指定して) 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た場合は、クォータが適用されないことに注意してください。

ユーザーにクォータを設定するには、シェルプロンプトで、root で次のコマンドを実行します。

```
# edquota username
```

クォータが必要な各ユーザーに対して、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

EDITOR 環境変数により定義されたテキストエディターは、**edquota** により使用されます。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、使用するエディターのフルパスに設定します。

最初の列は、クォータが有効になっているファイルシステムの名前です。2 列目には、ユーザーが現在使用しているブロック数が示されます。その次の 2 列は、ファイルシステム上のユーザーのソフトブロック制限およびハードブロック制限を設定するのに使用されます。

ソフトブロック制限は、使用可能な最大ディスク容量を定義します。

ハードブロック制限は、ユーザーまたはグループが使用できる最大ディスク容量 (絶対値) です。この制限に達すると、それ以上のディスク領域は使用できなくなります。

GFS2 ファイルシステムは inode のクォータを維持しないため、この列は GFS2 ファイルシステムには適用されず、空白になります。

いずれかの値が 0 に設定されていると、その制限は設定されません。テキストエディターで制限を変更します。以下に例を示します。

```
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できます。(**account=on** オプションを指定して) 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た場合は、クォータが適用されません。

devel グループのグループクォータを設定するには (グループはグループクォータを設定する前に存在している必要があります)、次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 のクォータを維持しないため、この列は GFS2 ファイルシステムには適用されず、空白になります。この制限を変更して、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、次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
```

クォータが有効な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、次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。各ユーザーに続いて表示される **--** で、ブロック制限を超えたかどうかを簡単に判断できます。ブロックのソフト制限を超えると、出力の最初が **-** ではなく、**+** となります。2 番目の **-** は inode の制限を示していますが、GFS2 ファイルシステムは inode の制限に対応していないため、その文字はそのまま **-** となります。GFS2 ファイルシステムは猶予期間に対応していないため、**grace** (猶予) 列は空白になります。

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、すべてのクォータ情報をディスク上にある独自の内部ファイルに保存します。GFS2 ノードは、ファイルシステムの書き込みごと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 60 秒ごとにクォータファイルを更新します。これは、クォータファイルへの書き込みをノード間で行うことを避けるために必要です。このような場合は、パフォーマンスが低下します。

ユーザーまたはグループがクォータ制限に近づくと、GFS2 はクォータファイルの更新の間隔を動的に短縮し、制限を超えないようにします。クォータ同期の間の通常の期間は、調整可能なパラメーターである **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、各ノードごとと、ファイルシステムをマウントす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。

用途

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で、同期に対応するすべてのファイルシステムを同期します。-a を指定しない場合は、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 による通常のクォータファイル同期の新しい間隔を指定します。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可能性があります。

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。クラスター内の全ノードは、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には、そのファイルシステムのサイズは縮小できません。

用途

```
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ーの管理』 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 モード。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。基となる論理ボ

リュームを拡張しなくても、ジャーナルはいつでも動的に GFS2 ファイルシステムに追加できます。**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。その他のすべてのノードは、拡張が発生したことを検知します。

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はジャーナルが埋め込みメタデータではなくプレーンファイルであるため、基となる論理ボリュームを拡張するだけではジャーナル用の領域が確保されないためです。

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
```

用途

```
gfs2_jadd -j Number MountPoint
```

Number
新たに追加す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 モード。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。この呼び出しは、すべてのデータが安全に書き込まれたことをディスクに報告すると返されます。

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。ファイルサイズが増加すると、この利点が急速に低下します。中規模ファイルおよび大規模ファイルへの書き込みは、データジャーナリングが有効になっているとかなり遅くなります。

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 (およびそのすべてのサブディレクトリー) に作成されるすべての GFS2 ファイルに対して自動的に有効にできます。長さがゼロの既存のファイルも、データジャーナリングのオン/オフを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 に設定され、その後そのディレクトリー内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ル処理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。ディレクトリーにこのフラグを設定すると、その後にそのディレクトリー内に作成されたすべてのファイルとディレクトリーがジャーナリング処理されます。次の一連のコマンドは、**gfs2_dir** ディレクトリーに **j** フラグを設定し、フラグが正しく設定された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。ディレクトリーに **j** フラグが設定されているため、**newfile** のジャーナリングも有効にする必要があります。

```
# 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

各ファイルの inode と、ディレクトリーの inode には、3 つのタイムスタンプが関連付けられています。

- **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

ファイルシステムがマウントされるときに **relatime** (相対 atime) Linux マウントオプションを指定できます。これにより、以前の **atime** 更新が **mtime** または **ctime** 更新よりも古い場合は、**atime** が更新されたことを指定します。

用途

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

BlockDevice

GFS2 ファイルシステムを置く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

ファイルシステムのマウント時に **noatime** Linux マウントオプションを指定できます。これにより、そのファイルシステムの **atime** 更新が無効になります。

用途

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

BlockDevice

GFS2 ファイルシステムを置く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

ファイルシステムへの書き込み動作を一時停止するには、**dmsetup suspend** コマンドを使用します。書き込みアクティビティーを停止することで、ハードウェアベースのデバイスのスナップショットを使

用して、一貫性のある状態でファイルシステムをキャプチャーできます。**dmsetup resume** コマンドは、一時停止を終了します。

用途

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

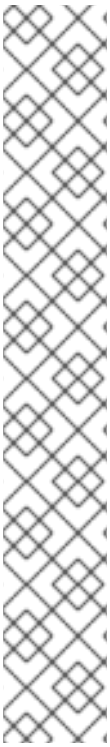
ファイルシステムがマウントされている状態でノードに障害が発生すると、ファイルシステムのジャーナリングにより迅速な復元が可能になります。ただし、ストレージデバイスの電源が切れたり、物理的に切断されていたりすると、ファイルシステムの破損が発生することがあります。(ジャーナリングは、ストレージサブシステムの障害からの復旧には使用できません。) そのような破損が発生した場合は、**fsck.gfs2** コマンドを使用して、GFS2 ファイルシステムを復旧できます。

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ている場合は、ファイルシステムリソースを無効にしてファイルシステムのマウントを解除できます。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** フラグを使用して、詳細のレベルを増やすことができます。2 番目の **-v** フラグを追加すると、レベルが再度増加します。
- **-q** フラグを使用して、詳細のレベルを減らすことができます。2 番目の **-q** フラグを追加すると、再びレベルが減ります。
- **-n** オプションは、ファイルシステムを読み取り専用として開き、すべてのクエリーに **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。

用途

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

-y

-y フラグにより、すべての質問の回答が **yes** となります。**-y** フラグを指定すると、**fsck.gfs2** コマンドは、変更を行う前に回答を求めるプロンプトを出しません。

BlockDevice

GFS2 ファイルシステムを置く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。修復するすべての質問に、自動的に **yes** と答えます。

```
# 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、マウントを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すべて、そのノードで完全に機能し続けます。GFS2 の withdraw 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 の無効を引き起こす (撤回) 可能性がある不一致の例は、ファイルの inode に対するブロック数が間違っています。GFS2 がファイルを削除すると、そのファイルが参照するすべてのデータおよびメタデータブロックがシステムにより削除されます。完了すると、inode のブロック数を確認します。ブ

ロック数が 1 でない場合 (つまり、残りのすべてがディスクの inode 自体である場合)、inode のブロック数はファイルに使用されている実際のブロックと一致しないため、ファイルシステムが不整合であることを示します。

多くの場合、この問題の原因は、ハードウェアの障害 (メモリー、マザーボード、HBA、ディスクドライブ、ケーブルなど) にある可能性があります。また、カーネルのバグ (GFS2 のメモリーを誤って上書きする別のカーネルモジュール) や、実際のファイルシステムの損傷 (GFS2 のバグによる) が原因で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 GFS2 ファイル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。

撤回の原因になっ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、ファイルシステムサービスを停止して、クラスター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確認を実行します。

1. 影響を受けるノードを再起動します。
2. Pacemaker でクローン以外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に **-o errors=panic** 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ことで、GFS2 の withdraw 機能を無効にできます。

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

このオプションが指定されていると、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、監視や介入がなく長期間にわたり無人状態になるクラスターに役に立ちます。

内部的には、GFS2 の withdraw 機能は、ロック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、withdraw が発生すると、通常は、システムログに報告されたデバイスマッパーデバイスからの I/O エラーが多数表示されるようになります。

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

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くても、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 ロックダンプの場合:

  ```
  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作成して、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ために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は、以下の問題を確認してください。

- フェンスに失敗した可能性があります。GFS2 ファイルシステムはフリーズし、フェンスが失敗した場合にデータの整合性を確保します。メッセージログを確認して、ハング時に失敗したフェンスがあるかどうかを確認します。フェンシングが正しく設定されていることを確認します。

- GFS2 ファイルシステムがクラッシュした可能性があります。メッセージログで **withdraw** という単語を確認し、GFS2 のメッセージおよびコールトレースで、ファイルシステムが撤回されたことを示すメッセージがないかどうかを確認します。withdraw は、ファイルシステムの破損、ストレージ障害、またはバグを示します。ファイルシステムのマウントを解除するするのが都合が良い早い段階で、以下の手順を実行する必要があります。

  1. 撤回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。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 サポートに、サポート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 の撤回では、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もあります。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ドが必要です。

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

- このエラーは、ロックの問題またはバグを示している可能性があります。「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」 に従ってこの問題の発生時にデータを収集し、Red Hat サポートにサポートチケットを開きます。

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

新しいノードをクラスターに追加し、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 ファイルシステムのジャーナル数が、GFS2 ファイルシステムへのアクセスを試行しているノードよりも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、ジャーナルが 1 つ必要です (ただし、**spectator** マウントオプションを設定してマウントした GFS2 ファイルシステムはジャーナルを必要としないため除外されます)。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 ファイルシステムのジャーナルがディスク上で領域 (ジャーナルの数 * 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。大量のジャーナルや大規模なジャーナルを指定していない場合でも、小さな GFS2 ファイルシステム (1GB 以下の範囲) には、デフォルトの GFS2 ジャーナルサイズで、使用中の大容量の領域が表示されます。

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。クラスターにはフェンス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

デフォルトでは、**no-quorum-policy** の値は **stop** に設定されます。クォーラムが失われると、残りのパーティションのリソースがすべてすぐに停止します。通常、このデフォルト設定は最も安全なオプションで最適なおプションですが、ほとんどのリソースとは異なり、GFS2 が機能するにはクォーラムが必要です。クォーラムが失われると、GFS2 マウントを使用したアプリケーション、GFS2 マウント自体の両方が正しく停止できません。クォーラムなしでこれらのリソースを停止しようとすると失敗し、最終的にクォーラムが失われるたびにクラスター全体がフェンスされます。

この状況に対処するには、GFS2 の使用時の **no-quorum-policy** を **freeze** に設定します。この設定では、クォーラムが失われると、クォーラムが回復するまで残りのパーティションは何もしません。

```
# 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** で起動中には、起動する必要はあり

ません。さらに、起動手順の一部として **ocf:odbc:clvm リソース** エージェントでは、**/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設定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設定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ノードごとに、ジャーナルが 1 つ必要になります。クラスター内の各ノードに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	使用方法
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)。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の種別を概説します。各メトリックでは、**-T** フラグと共に **pminfo** ツールを使用すると、追加情報を確認できます。

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 (各 GFS2 トレースポイントのいずれか) は、制御メトリックを使用してオンまたはオフを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、付与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の可能性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の可能性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、ユーザーは選択した 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** を使用するとアーカイブログ全体をダンプしたり、コマンドラインから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 インターフェイ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** インターフェイスおよび GFS2 トレースポイントについて記載しています。本セクションの対象者は、GFS2 の設計や GFS2 固有の問題のデバッグ方法の確認を希望し、ファイルシステムの内部に精通している上級ユーザーです。

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。上記のトレースポイントタイプを使用して、実行中の GFS2 ファイルシステムを監視し、以前のリリースの Red Hat Enterprise Linux でサポートされていたデバッグオプション以外の追加オプションを取得できます。トレースポイントは、ハングやパフォーマンスの問題が再現可能で、問題のある操作中にトレースポイント出力を取得できる場合に特に役立ちます。GFS2 では、glock は主要なキャッシュ制御メカニズムで、GFS2 のコアのパフォーマンスを理解するときの鍵となります。bmap (ブロックマップ) トレースポイントを使用して、ブロック割り当てとブロックマッピング (ディスク上のメタデータツリーで既に割り当てられているブロックのルックアップ) を監視し、アクセスの局所性に関する問題を確認できます。ログトレースポイントは、ジャーナルに書き込まれ、ジャーナルから公開されるデータを追跡し、GFS2 の対象部分に関する有用な情報を提供できます。

トレースポイントは、できるだけ汎用性が保たれ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。このインターフェイスのユーザーは、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イ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イ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ており、**blktrace** トレースポイントを GFS2 のトレースポイントと組み合わせて使用して、システムパフォーマンスの全体像を把握できます。トレースポイントの動作レベルでは、非常に短い時間で大量のデータを生成できます。トレースポイントは、有効になったときにシステムにかかる負荷が最小限になるように設計されていますが、何らか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ーリングすることで、データ量が減り、特定の状況を理解するのに有用な情報だけを取得することに集中できます。

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。**gfs2** モジュールが読み込まれると、各 GFS2 イベントごとにサブディレクトリーを含む **gfs2** サブディレクトリーが現れます。**/sys/kernel/debug/tracing/events/gfs2** ディレクトリー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
```

すべての GFS2 トレースポイントを有効にするには、次のコマンドを入力します。

```
[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。各イベントの意味は、以下で詳しく説明します。

トレースポイントからの出力は、ASCII またはバイナリーの形式で提供されます。現在、この付録ではバイナリーインターフェイスに関する説明は含まれません。ASCII インターフェイスは 2 つの方法で利用できます。リングバッファーの現在の内容を一覧表示するには、以下のコマンドを実行します。

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

このインターフェイスは、一定期間にわたって長時間実行しているプロセスを使用し、イベントの後にバッファー内の最新のキャプチャー情報を確認する必要がある場合に便利です。もう 1 つのインターフェイスは **/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、発生時にこのファイルから読み取られます。このインターフェイスを介して利用可能な履歴情報はありません。出力の形式は両方のインターフェイスで同じであり、各 GFS2 イベントはこの付録の後のセクションで説明します。

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ーを利用することができます。このユーティリティーの詳細は、「リファレンス」 のリンクを参照してください。**trace-cmd** ユーティリティーは **strace** ユーティリティー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. GLOCK

GFS2 を理解するために、理解する必要がある最も重要な概念で、他のファイルシステムと区別されるのは、glocks の概念です。ソースコードの観点から、glock は、DLM を統合して 1 台のマシンにキャッシュするデータ構造です。各 glock は、1 つの DLM ロックと 1:1 の関係を持ち、そのロック状態のキャッシュを提供します。これにより、ファイルシステムの 1 つのノードから実行される反復操作が DLM を繰り返し呼び出す必要がないため、不要なネットワークトラフィックを回避できます。glock には、メタデータをキャッシュするカテゴリーとキャッシュしない広範なカテゴリーが 2 つあります。inode の glock およびリソースグループの glock の両方がキャッシュメタデータをキャッシュし、他のタイプの glock はメタデータをキャッシュしません。inode の glock はメタデータに加えてデータのキャッシングにも関与し、すべての glock の中で最も複雑なロジックを持っています。

表B.1 Glock モードおよび DLM ロックモード

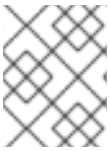
Glock モード	DLM ロックモード	備考
UN	IV/NL	ロック解除 (I フラグに依存する glock または NL ロックに関連付けられた DLM ロックがない)
SH	PR	共有 (保護された読み取り) ロック
EX	EX	排他ロック
DF	CW	ダイレクト I/O およびファイルシステムのフリーズに使用される遅延 (同時書き込み)

Glock は、(別のノードの要求または仮想マシンの要求で) ロックが解除されるまでメモリー内に残り、ローカルユーザーはありません。この時点で、glock ハッシュテーブルから削除され、解放されます。glock が作成されると、DLM ロックは glock に即座に関連付けられません。DLM ロックは、DLM への最初の要求時に glock に関連付けられます。この要求が成功すると、I(initial) フラグが glock に設定さ

れます。表B.4「glock フラグ」 は、さまざまな glock フラグの意味を示しています。DLM が glock に関連付けられていると、DLM ロックは、少なくとも解放されるまで常に NL (Null) ロックモードのままになります。NL からロック解除への DLM ロックの降格は、常に glock の有効期間時の最後の操作になります。

各 glock には多数のホルダーを関連付けることができ、それぞれが上位レイヤーからのロック要求を表します。コードの重要なセクションを保護するために、GFS2 キューに関連するシステムコールおよびホルダーを glock のキューからからキューから取り出します。

glock 状態のマシンはワークキューに基づいています。パフォーマンス上の理由から、タスクレットを使用することが推奨されます。ただし、現在の実装では、そのコンテキストから I/O を送信し、使用を禁止する必要があります。



注記

ワークキューには、GFS2 トレースポイントと組み合わせて使用できる独自のトレースポイントがあります。

表B.2「Glock モードおよびデータタイプ」 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を示しています。これは、inode ロックとリソースグループロックの両方に適用されますが、リソースグループロック用の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 モードおよびデータタイプ

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 インターフェイス

glock **debugfs** インターフェイスを使用すると、glock とホルダーの内部の状態を視覚化でき、場合によってはロックされたオブジェクトの概要情報も含まれます。ファイルの各行は、インデントなしで G: を開始する (glock 自体を参照する) か、シングルスペースでインデントされた別の文字で開始し、ファイル内の直前の glock に関連する構造を参照します (H: はホルダー、I: は inode、および R: はリソースグループ) です。このファイルの内容の例は、以下のようになります。

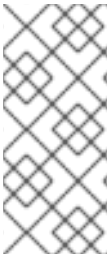
```
G:  s:SH n:5/75320 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:EX n:3/258028 f:yI t:EX d:EX/0 a:3 r:4
 H: s:EX f:tH e:0 p:4466 [postmark] gfs2_inplace_reserve_i+0x177/0x780 [gfs2]
 R: n:258028 f:05 b:22256/22256 i:16800
G:  s:EX n:2/219916 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。この図の glock は、glock ダンプのより興味深い機能のいくつかを示すために選択されています。

glock の状態は、EX (排他的)、DF (据え置き)、SH (共有)、または UN (ロック解除) になります。この状態は、DLM null ロック状態を表す可能性がある UN を除いて、または GFS2 が DLM ロックを保持しないことを除いて、DLM ロックモードに直接対応します (上記の I フラグにより異なります)。glock の s: フィールドはロックの現在の状態を示し、ホルダーの同じフィールドは要求されたモードを示します。ロックが許可されると、ホルダーのフラグに H ビットが設定されます (f: フィールド)。設定されていない場合は、W wait ビットが設定されます。

N: フィールド (数値) は、各項目に関連付けられた番号を示します。glock の場合、タイプ番号の後に glock 番号が続くため、上記の例では最初の glock は n:5/75320 となり、inode 75320 に関連する **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になります。



注記

debugfs glock ファイルの 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 16 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 の全フラグのリストは、表B.4「glock フラグ」 と 表B.5「glock ホルダーフラグ」 に記載されています。ロック値のブロックの内容は、現在 **debugfs** インターフェイスからは利用できません。

表B.3「glock の種類」 は、さまざまな glock タイプの意味を示しています。

表B.3 glock の種類

タイプ番号	ロックタイプ	使用方法
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
3	rgrp	リソースグループのメタデータ

タイプ番号	ロックタイプ	使用方法
4	meta	スーパーブロック
5	iopen	最後に閉じた inode 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 つは、l (locked) フラグです。これは、状態変更が行われるときに glock 状態へのアクセスを調整するのに使用されるビットロックです。これは、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。これは、複数のロック要求が送信され、その間にさまざまな無効化が発生することを意味します。

表B.4「glock フラグ」 は、さまざまな glock フラグの意味を示しています。

表B.4 glock フラグ

フラグ	名前	意味
d	Pending demote	遅延している (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す (復旧が進行中です)。
i	Invalidate in progress	この glock の下でページを無効にする過程です。
I	Initial	DLM ロックがこの glock と関連付けられる場合に指定します。
l	Locked	glock は、状態を変更中です。
L	LRU	glock が LRU 一覧にあるときに設定します。
o	Object	glock がオブジェクトに関連付けられている (つまり、タイプ 2 の glock の場合は inode、タイプ 3 の glock の場合はリソースグループ) ときに設定されます。
p	Demote in progress	glock は、降格要求に応答中です。

フラグ	名前	意味
q	Queued	ホルダーが glock にキューイングされると設定され、glock が保持されるとクリアされますが、残りのホルダーはありません。アルゴリズムの一部として使用され、glock の最小保持時間を計算します。
r	Reply pending	リモートノードから受信した返信の処理を待機中です。
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ノードと競合するモードでロックを取得する必要のあるノードからリモートコールバックを受け取ると、D (降格) または d (降格保留) のフラグのいずれかが設定されます。特定のロックの競合が発生した場合の枯渇状態を防ぐために、各ロックには最小保持時間が割り当てられます。最小保持時間の間にロックされていないノードは、その期間が経過するまでロックを保持できます。

期間が過ぎると、D (降格) フラグが設定され、必要な状態が記録されます。この場合、次にホルダーキューに許可されたロックがなくなると、ロックが降格になります。期間が過ぎていない場合は、代わりに d (降格保留) フラグが設定されます。また、これによりステートマシンが d (降格保留) を消去し、最小保持期間が過ぎた場合に D (降格) を設定します。

glock に DLM ロックが割り当てられている場合は、I (初期) フラグが設定されます。これは、glock が最初に使用され、最終的に glock が解放されるまで (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」 は、さまざまな glock ホルダーフラグの意味を示しています。

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 (結果は後でポーリングします)。
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。

フラグ	名前	意味
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。

前述のように、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ため、ホルダーフラグで最も重要となるのは H (ホルダー) および W (待機) です。一覧内でのホルダーの順序は重要です。許可されたホルダーがある場合は常にキューの先頭にあり、その後にキューに追加されているホルダーが続きます。

許可されたホルダーがない場合は、一覧の最初のホルダーが次の状態変更をトリガーするホルダーになります。降格要求は常にファイルシステムからの要求よりも優先度が高いとみなされるため、必ずしも要求された状態が直接変更されるとは限りません。

glock サブシステムは、2 種類の try ロックに対応しています。これらは、(適切なバックオフと再試行により) 通常の順序からロックを解除でき、他のノードで使用されているリソースを回避するために使用できるため役立ちます。通常の t (試行) ロックは、その名前が示すとおりです。特別なことは行わない試行ロックです。一方、T (**try 1CB**) ロックは、DLM が現在互換性のないロックホルダーにコールバックを 1 つ送信するのを除き、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。これは、inode の **i_nlink** カウントがゼロのときノード間での調整に使われます。また、inode の割り当てに対応するノードを決定します。**iopen** glock は通常共有状態で保持されますが、**i_nlink** 数がゼロになり、**->evict_inode**() が呼び出されると、T (**try 1CB**) が設定された排他的ロックが要求されます。ロックが許可されると、inode の割り当て解除が継続されます。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。次に、特定の I/O が発行され、正しいロックの下で完了したこと、および競合が存在しないことを確認できます。

gfs2_glock_state_change トレースポイントを理解することは、最も重要なものです。これは、glock の最初の作成から最後の降格までのすべての状態変化を追跡します。最後の降格は、**gfs2_glock_put** とロック解除の遷移までの最後の NL で終わります。glock フラグの l (locked) は、常に状態が変更される前に設定され、終了するまで削除されません。状態変更中は、許可されたホルダー (H glock ホルダーフラグ) が存在することはありません。キューに格納されたホルダーがある場合、それらは常に W (waiting) 状態になります。状態の変更が完了すると、glock の l フラグを削除する前の最後の操作で、ホルダーを付与できます。

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せず、多くの場合は **umount** または場合によってはメモリーの再取得によって作成されます。リモート降格要求の数は、特定の inode またはリソースグループのノード間の競合の測定値です。

gfs2_glock_lock_time トレースポイントは、DLM への要求に要し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 glock に導入されました。

ホルダーにロックが付与されると、**gfs2_promote** が呼び出されます。これは、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、状態変更の最終段階、または即時に付与できるロックが要求されたときに発生します。ホルダーがこの glock に付与される最初のホルダーである場合は、f (最初の) フラグがそのホルダーに設定されます。これは現在、リソースグループでのみ使用されます。

B.7. BMAP トレースポイント

ブロックマッピングは、どのファイルシステムでも中心となるタスクです。GFS2 は、ブロックごとに 2 ビットとなる従来のマテリアルベースのシステムを使用します。このサブシステムにおけるトレースポイントの主な目的は、ブロックの割り当ておよびマッピングにかかる時間の監視を可能にすることです。

gfs2_bmap トレースポイントは、各 bmap 操作で 2 回呼び出されます。まずは bmap リクエストを表示するとき、次に終了時に結果を表示するときです。これにより、要求と結果を容易に一致させ、ファイルシステムの異なる部分、異なるファイルオフセット、または異なるファイルのブロックをマッピングするのに要した時間を測定できます。返されたエクステントサイズの平均と、要求されるエクステント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、破棄されたブロック予約を追跡します。

割り当て済みブロックを追跡するには、割り当てだけでなく、ブロックの解放でも **gfs2_block_alloc** を呼び出します。割り当てはすべて、ブロックの対象となる inode に従って参照されるため、これを使用して、どの物理ブロックがライブファイルシステムのどのファイルに属しているかを追跡できます。これは **blktrace** と組み合わせると特に便利です。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で取得したマッピングを使用して、関連する inode に戻って参照されることがあります。

B.8. ログトレースポイント

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**) に追加されたブロックや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にかかった時間を追跡します。これは、ジャーナリングのパフォーマンス問題をデバッグしようとする際に非常に便利です。

gfs2_log_blocks トレースポイントは、ログ内の予約ブロックを追跡し、たとえば、ログがワークロードに対して小さすぎるかどうかを表示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点で **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 は、ファイルシステム内で何が発生しているかを追跡するのに役立つ統計を維持します。これにより、パフォーマンスの問題を特定できます。

GFS2 は、2 つのカウンターを維持します。

- **dcount** (要求された DLM 操作の回数をカウントします)。これは、平均/分散の計算にかかったデータ量を示します。
- **qcount** (要求された **syscall** レベルの操作をカウントします)。通常、**qcount** は **dcount** と同じか、それ以上になります。

また、GFS2 では、平均と分散のペアが維持されます。平均/分散ペアは平滑化された指数予測であり、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を計算するために使用されるものです。
GFS2 で維持される平均と分散のペアはスケーリングされておらず、整数のナノ秒で表されます。

- srtt/srttvar - 非ブロック操作の平滑化された往復時間
- srttb/srttvarb - ブロック操作の平滑化された往復時間
- irtt/irttvar - 要求間の時間 (例: DLM 要求間の時間)

ブロック以外の要求とは、問題の DLM ロックの状態に関係なく、すぐに完了する要求のことです。これは現在、(a) ロックの現在の状態が排他的、(b) 要求された状態が null またはアンロックである、または (c) try lock フラグが設定されている場合の要求を意味します。ブロックリクエストは、他のすべてのロックリクエストに対応します。

時間が長いほど IRTT に適していますが、時間が短いほど RTT に適しています。

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。データは、glock が作成される glock タイプの CPU あたりのデータから初期化されます (ゼロ化されたカウンターを除く)。このファイルは非常に大きくなる可能性があります。
- **lkstats** ファイル。これには、各 glock タイプの CPU ごとの統計が含まれます。1 行に 1 つの統計が含まれ、各列は CPU コアです。glock のタイプごとに 8 つの行があり、タイプは交互に続きます。

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** ユーティリティーに関する情報は、 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 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 リリースのバージョン

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

アンマウント、システムのハング, GFS2 ファイルシステムのアンマウント

アンマウントコマンド, GFS2 ファイルシステムのアンマウント

アンマウント時のシステムハング, GFS2 ファイルシステムのアンマウント

クォータの管理, GFS2 のクォータ管理, 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
  クォータの同期, quotasync コマンドを使用したクォータの同期

ジャーナル追加用の GFS2 固有のオプション表, 完全な使用法

テーブル
  mkfs.gfs2 コマンドオプション, 全オプション

  ジャーナル追加用の GFS2 固有のオプション, 完全な使用法

  ファイルシステムを拡張する GFS2 固有のオプション, 完全な使用法

  マウントオプション, 完全な使用法

ディスククォータ
  グループごとに割り当て, グループごとのクォータ割り当て

  ソフト制限, ユーザーごとのクォータ割り当て

  ハード制限, ユーザーごとのクォータ割り当て

  ユーザーごとに割り当て, ユーザーごとのクォータ割り当て

  有効化, ディスククォータの設定
    quotacheck、実行中, クォータデータベースファイルの作成

    クォータファイルの作成, クォータデータベースファイルの作成

  管理, ディスククォータの管理
    quotacheck コマンド、確認, クォータの精度維持

    レポート, ディスククォータの管理

  関連資料, リファレンス

データジャーナリング, データジャーナリング

トレースポイント, GFS2 トレースポイントおよび debugfs glocks ファイル

ノードロック機能, GFS2 のノードロック機能

パフォーマンスチューニング, GFS2 によるパフォーマンスチューニング

ファイルシステム
  atime、更新の設定, atime 更新の設定
    noatime でのマウント , noatime を使用したマウント

    relatime でのマウント , relatime を使用したマウント

  アクティビティーの一時停止, GFS2 ファイルシステム上の動作の一時停止

アンマウント, GFS2 ファイルシステムのアンマウント
クォータの管理, GFS2 のクォータ管理, 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
クォータの同期, quotasync コマンドを使用したクォータの同期

ジャーナルの追加, 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, データジャーナリング
マウントする, GFS2 ファイルシステムマウント
作成, GFS2 ファイルシステムの作成
修復, GFS2 ファイルシステムの修復
拡大, GFS2 ファイルシステムの拡張

ファイルシステムにジャーナルを追加, GFS2 ファイルシステムへジャーナルの追加
ファイルシステムのアンマウント, GFS2 ファイルシステムのアンマウント
ファイルシステムのマウント, GFS2 ファイルシステムマウント
ファイルシステムの作成, GFS2 ファイルシステムの作成
ファイルシステムの修復, GFS2 ファイルシステムの修復
ファイルシステムの拡張, GFS2 ファイルシステムの拡張
ファイルシステム上の動作の一時停止, GFS2 ファイルシステム上の動作の一時停止
ファイルシステム拡張用の GFS2 固有のオプション表, 完全な使用法
マウントコマンド, GFS2 ファイルシステムマウント
マウント表, 完全な使用法
新機能と変更点, 新機能と変更点
最大サイズ, GFS2 ファイルシステム, GFS2 サポート制限
概要, GFS2 の概要
新機能と変更点, 新機能と変更点
設定, 事前, GFS2 を設定する前に

設定, 事前, GFS2 を設定する前に
設定における考慮事項, GFS2 の設定および操作における考慮事項
調整, パフォーマンス, GFS2 によるパフォーマンスチューニング

A

acl マウントオプション, GFS2 ファイルシステムマウント
atime、更新の設定, atime 更新の設定
noatime でのマウント , noatime を使用したマウント
relatime でのマウント , relatime を使用したマウント

D

debugfs, GFS2 トレースポイントおよび debugfs glocks ファイル

debugfs ファイル, 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

F

fsck.gfs2 コマンド, GFS2 ファイルシステムの修復

G

GFS2
- atime、更新の設定, atime 更新の設定
  - noatime でのマウント , noatime を使用したマウント
  - relatime でのマウント , relatime を使用したマウント
- クォータの管理, GFS2 のクォータ管理, 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
  - クォータの同期, quotasync コマンドを使用したクォータの同期
- 撤回関数, GFS2 withdraw 機能
- 操作, GFS2 の設定および操作における考慮事項
- 管理, GFS2 の管理
- 設定における考慮事項, GFS2 の設定および操作における考慮事項

GFS2 の管理, GFS2 の管理

GFS2 ファイルシステムの最大サイズ, GFS2 サポート制限

gfs2_grow コマンド, GFS2 ファイルシステムの拡張

gfs2_jadd コマンド, GFS2 ファイルシステムへジャーナルの追加

glock, GFS2 トレースポイントおよび debugfs glocks ファイル

glock のタイプ, 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, glock debugfs インターフェイス

glock フラグ, 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, glock debugfs インターフェイス

glock ホルダーフラグ, 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, glock ホルダー

M

mkfs コマンド, GFS2 ファイルシステムの作成

mkfs.gfs2 コマンドオプションテーブル, 全オプション

P

Posix ロック, POSIX ロックの問題

Q

quotacheck , クォータデータベースファイルの作成

quotacheck コマンド
によるクォータ正確性の確認, クォータの精度維持

quota_quantum 調整可能なパラメーター, quotasync コマンドを使用したクォータの同期

W

withdraw 機能, GFS2, GFS2 withdraw 機能