Red Hat Enterprise Linux 6

Global File System 2

Red Hat Global File System 2

エディッション 7

Last Updated: 2017-10-12

Red Hat Enterprise Linux 6 Global File System 2

Red Hat Global File System 2
エディッション 7

法律上の通知

概要

本ガイドでは、Red Hat Enterprise Linux 6 対応の Red Hat GFS2 (Red Hat Global File System 2) の設定方法および管理方法について説明しています。

目次

はじめに

本ガイドでは、Resilient Storage Add-On に組み込まれている Red Hat GFS2 (Red Hat Global File System 2) の設定および維持管理の方法について説明します。

1. 対象読者

本ガイドは、主として、以下の作業に精通している Linux システム管理者を対象としています。

- カーネルの設定を含む、Linux システムの管理手順
- Fibre Channel SAN などの共有ストレージネットワークのインストールと設定

2. 関連ドキュメント

Red Hat Enterprise Linux に関する詳細情報は、以下の資料を参照してください。

- 『インストールガイド』 — Red Hat Enterprise Linux 6 のインストールに関する情報を記載しています。
- 『導入ガイド』 — Red Hat Enterprise Linux 6 の導入、設定、管理に関する情報を記載しています。
- 『ストレージ管理ガイド』 — Red Hat Enterprise Linux 6 上のストレージデバイスおよびファイルシステムの効果的な管理方法についての説明を記載しています。

Red Hat Enterprise Linux 6 向け High Availability Add-On および Resilient Storage Add-On の詳細については以下の資料を参照してください。

- 『High Availability Add-On 概要』 — Red Hat High Availability Add-On の全体的な概要を説明しています。
- 『クラスターの管理』 — High Availability Add-On のインストール、設定、および管理に関して説明しています。
- 『論理ボリュームマネージャの管理』 — クラスター化した環境で LVM を稼働させる方法など、論理ボリュームマネージャー (LVM) について詳しく説明しています。
- 『DM Multipath』 — Red Hat Enterprise Linux の Device-Mapper Multipath 機能の使い方について説明しています。
- 『Load Balancer Administration』 — Load Balancer Add-On を使ってパフォーマンス性の高いシステムおよびサービスを構築する方法について説明しています。Load Balancer Add-On は統合ソフトウェアコンポーネントの集合で構成され、複数の実サーバー全体の IP 負荷を分散する Linux Virtual Servers (LVS) を提供します。
- 『リリースノート』 — Red Hat 製品の最新リリースに関する情報を記載しています。

Red Hat Cluster Suite ドキュメントおよびその他の Red Hat ドキュメントは、 Red Hat Enterprise Linux ドキュメント CD またはオンライン (https://access.redhat.com/site/documentation/) でご覧頂けます。HTML、PDF および RPM 形式でご利用いただけます。

3. フィードバック

誤字、脱字を発見された場合、もしくは本ガイドを改善するためのご意見、ご提案がございましたらBugzilla 経由にてご連絡ください (http://bugzilla.redhat.com/) 。製品には **Red Hat Enterprise Linux 6**、コンポーネントには **doc-Global_File_System_2** を選択してください。バグ報告の本文には下記のガイド識別子をご記入いただくようお願いします。

```
rh-gfs2(EN)-6 (2014-10-8T15:15)
```

ご意見、ご提案を頂く場合は、できるだけ具体的にご指摘いただけるようお願いします。誤字、脱字、誤りなどを発見された場合は、セクション番号と該当部分の前後の文章も含めてご報告いただけると迅速な修正を行うことができます。

第1章 GFS2 の概要

Red Hat GFS2 ファイルシステムは、Resilient Storage Add-Onに組み込まれています。Linux カーネルファイルシステムのインターフェイス (VFS 層) と直接連動するネイティブのファイルシステムです。クラスターファイルシステムとして実装する場合、GFS2 は分散メタデータと複数ジャーナルを使用します。Red Hat がサポートしているのは、High Availability Add-Onで実装されている GFS2 ファイルシステムの使用のみになります。

注記

GFS2 ファイルシステムは、スタンドアロンのシステムやクラスター構成の一部として実装することが可能ですが、Red Hat Enterprise Linux 6 リリースでは、単一ノードファイルシステムとしての GFS2 の使用はサポートしていません。Red Hat は、単一ノード向けに最適化された、クラスターファイルシステムよりもオーバーヘッドが概して低い、ハイパフォーマンスな単一ノードファイルシステムを数多くサポートしています。ファイルシステムをマウントする必要があるのが単一のノードのみの場合は、GFS2 ではなくハイパフォーマンスな単一ノードファイルシステムを使用することをお勧めします。

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

注記

Red Hat は、16ノードを超えるクラスターファイルシステムでの導入では GFS2 の使用に対応していません。

GFS2 は、64-bit アーキテクチャーをベースにしており、理論的には 8 EB の ファイルシステムに対応可能です。ただし、現在サポートしている GFS2 ファイルシステムの最大サイズは、64-bit のハードウェアでは 100 TB、32-bit のハードウェアでは 16 TB となっています。ご使用のシステムで、これらのサイズを上回る GFS2 ファイルシステムが必要な場合は、Red Hat のサービス担当者にご連絡ください。

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

クラスター構成の場合は、High Availability Add-Onの設定/管理ツールを使用して、Red Hat GFS2 のノードの設定と管理を行うことができます。Red Hat GFS2 では、クラスター内の GFS2 ノード間におけるデータ共有が可能で、GFS2 ノード全体にわたる、単一で一貫性のあるファイルシステム名前空間ビューを提供します。これにより、同一のノード上のプロセスでローカルファイルシステムのファイルを共有できるのと同様に、異なるノード上のプロセスで GFS2 ファイルを共有でき、顕著な違いは認められません。High Availability Add-Onに関する詳細は 『Red Hat Cluster の設定と管理』 を参照してください。

GFS2 ファイルシステムは LVM 外でも使用することができますが、Red Hat がサポートしているのは、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です。CLVM は、Resilient Storage Add-Onに組み込まれた、クラスター全体の LVM 実装で、クラスター内の LVM 論理ボリュームを管理する、CLVM デーモン **clvmd** により有効化されます。このデーモンにより、LVM2 を使用し

たクラスター全体にわたる論理ボリュームの管理が可能となり、クラスター内の全ノードで論理ボリュームを共有することが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、『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.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。

注記

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

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おり、以下のようなセクションで構成されます。

- 「新機能と変更点」
- 「GFS2 を設定する前に」
- 「GFS と GFS2 の相異点」
- 「GFS2 のインストール」
- 「GFS2 のノードロック機能」

1.1. 新機能と変更点

本セクションでは、Red Hat Enterprise Linux 6 の初版リリースならびにそれ移行のリリースに同梱されている GFS2 ファイルシステムの新機能と変更点、GFS2 関連のドキュメントについて記載しています。

1.1.1. Red Hat Enterprise Linux 6 の新機能と変更点

Red Hat Enterprise Linux 6.0 には以下のようなドキュメント、機能の更新および変更が含まれています。

- Red Hat Enterprise Linux 6 リリースでは GFS2 を単一ノードのファイルシステムとして使用する方法については対応していません。
- Red Hat Enterprise Linux 6 リリースでは GFS から GFS2 ファイルシステムにアップグレードするための **gfs2_convert** コマンドの機能が強化されています。このコマンドについての詳細は 付録B *GFS から GFS2 へのファイルシステム変換* を参照してください。
- Red Hat Enterprise Linux 6 リリースでは **discard**、**nodiscard**、**barrier**、**nobarrier**、**quota_quantum**、**statfs_quantum**、**statfs_percent** のマウントオプションに対応します。GFS2 ファイルシステムのマウント方法については 「ファイルシステムのマウント」 を参照してください。
- 本ドキュメントの Red Hat Enterprise Linux 6 バージョンには、新たなセクション 「GFS2 のノードロック機能」 が追加されました。このセクションでは、GFS2 ファイルシステムの機能の一部について説明しています。

1.1.2. Red Hat Enterprise Linux 6.1 の新機能と変更点

Red Hat Enterprise Linux 6.1 には以下のようなドキュメント、機能の更新および変更が含まれています。

- Red Hat Enterprise Linux 6.1 リリースからは GFS2 で Linux 標準のクォータ機能に対応するようになります。GFS2 クォータ管理については、「GFS2 のクォータ管理」 で説明しています。

  旧バージョンの Red Hat Enterprise Linux では、**gfs2_quota** コマンドを使用してクォータを管理する必要がありました。**gfs2_quota** に関する説明は、付録A *gfs2_quota コマンドを使用した GFS2 のクォータ管理* に記載しています。

- 本ドキュメントに 5章*GFS2 ファイルシステムに伴う問題の診断と修正* が新たに追加されました。

- 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。

1.1.3. Red Hat Enterprise Linux 6.2 の新機能と変更点

Red Hat Enterprise Linux 6.2 には以下のようなドキュメント、機能の更新および変更が含まれています。

- Red Hat Enterprise Linux 6.2 リリースでは、GFS2 で **tunegfs2** コマンドに対応するようになります。**gfs2_tool** コマンドの一部機能がこのコマンドで行えるようになります。詳細は **tunegfs2** の man ページをご覧ください。

  以下のセクションは、**gfs2_tool** コマンドを使用しないで行うことができる管理手順に更新されました。

  - 「**quotasync** コマンドを使用したクォータの同期」 および 「**gfs2_quota** コマンドを使用したクォータの同期」 では、**quota_quantum=** マウントオプションを使用して **quota_quantum** パラメーターをデフォルト値の 60 秒から変更する方法について説明しています。

  - 「ファイルシステム上の動作を一時停止する」 では、**dmsetup suspend** コマンドを使用してファイルシステムへの書き込み動作を一時停止する方法について説明しています。

- 本ガイドに 付録C *GFS2 トレースポイントおよび debugfs glocks ファイル* が新たに追加されました。この付録は、glock **debugfs** インターフェースおよび GFS2 トレースポイントについて記載しています。ファイルシステムの内部について精通している上級ユーザーの方で GFS2 の設計および GFS2 固有の問題のデバッグ方法について詳しく知りたい方を対象としています。

1.1.4. Red Hat Enterprise Linux 6.3 の新機能と変更点

Red Hat Enterprise Linux 6.3 リリースでは、本ガイドに 2章*GFS2 の設定および操作における考慮事項* が新たに追加されました。この章では GFS2 ファイルシステムの作成、使用、管理などを行う上で推奨される値など GFS2 のパフォーマンスを最適化するための推奨事項について説明しています。

更に、ドキュメント全体にわたり、記載内容の明確化を図り、若干の修正を加えました。

1.1.5. Red Hat Enterprise Linux 6.4 の新機能と変更点

Red Hat Enterprise Linux 6.4 リリースで、*2章GFS2 の設定および操作における考慮事項* が更新され、記載を明確にしました。

1.1.6. Red Hat Enterprise Linux 6.6 の新機能と変更点

Red Hat Enterprise Linux 6.6 リリース向けに新しい章 *2章GFS2 の設定および操作における考慮事項* が追加されました。GFS2 ファイルシステムが含まれている Pacemaker クラスターの設定に必要な手順を簡単に説明しています。

更に、ドキュメント全体にわたり、記載内容の明確化を図り、若干の修正を加えました。

1.2. GFS2 を設定する前に

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

GFS2 ノード

クラスター内で GFS2 ファイルシステムをマウントさせるノードを決めます。

ファイルシステムの数

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

ファイルシステム名

各ファイルシステムにはそれぞれ固有の名前を使用します。クラスター上の全 **lock_dlm** ファイルシステムで固有となる名前にしてください。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。例えば、本ガイドで例示している手順では **mydata1** や **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

使用する GFS2 ファイルシステムのジャーナル数を決めます。GFS2 ファイルシステムをマウントするノード毎に 1 つのジャーナルが必要になります。GFS2 では、後日、追加のサーバーがファイルシステムをマウントする時に、ジャーナルを動的に追加することができます。GFS2 ファイルシステムにジャーナルを追加する方法については 「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)。

注記

複数のノードから多数の作成や削除の操作が同じディレクトリー内で同時に実行されると、パフォーマンス関連の問題が見られることがあります。ご使用のシステムでこのような問題が生じた場合は、ノードによるファイルの作成と削除はできる限りそのノード専用のディレクトリーに限定するようにしてください。

GFS2 ファイルシステムの作成、使用、管理についての詳細は *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。

1.3. GFS2 のインストール

Red Hat High Availability Add-On に必要なパッケージの他、GFS2 の **gfs2-utils** パッケージ、CLVM (Clustered Logical Volume Manager) 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 チャンネルの一部となるため、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しておきます。

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 rgmanager lvm2-cluster gfs2-utils
```

Red Hat High Availability Add-On およびクラスター管理に関する全般的な説明については 『Cluster Administration』 ガイドを参照してください。

1.4. GFS と GFS2 の相異点

このセクションでは GFS にはなかった GFS2 の改善点および変更点について説明します。

GFS から GFS2 に移行する場合は、**gfs2_convert** ユーティリティを使用して GFS ファイルシステムを GFS2 に変換する必要があります。**gfs2_convert** ユーティリティについては 付録B *GFS から GFS2 へのファイルシステム変換* を参照してください。

1.4.1. GFS2 のコマンド名

GFS2 の機能は概ね GFS と同じです。ただし、ファイルシステムのコマンド名は GFS ではなく GFS2 になります。表1.1「GFS と GFS2 のコマンド」 に GFS コマンドとそれに対応する GFS2 コマンドを示します。

表1.1 GFS と GFS2 のコマンド

GFS コマンド	GFS2 コマンド	説明
mount	**mount**	ファイルシステムをマウントします。ファイルシステムのタイプが GFS なのか GFS2 なのかについてはシステムで判断されます。GFS2 のマウントオプションについては gfs2_mount(8) の man ページを参照してください。
umount	**umount**	ファイルシステムをアンマウントします。
fsck **gfs_fsck**	**fsck** **fsck.gfs2**	アンマウントしたファイルシステムのチェックと修復を行います。
gfs_grow	**gfs2_grow**	マウントしたファイルシステムを拡張します。
gfs_jadd	**gfs2_jadd**	マウントしたファイルシステムにジャーナルを追加します。

GFS コマンド	GFS2 コマンド	説明
gfs_mkfs **mkfs -t gfs**	**mkfs.gfs2** **mkfs -t gfs2**	ストレージデバイス上にファイルシステムを作成します。
gfs_quota	**gfs2_quota**	マウントしたファイルシステム上のクォータを管理します。Red Hat Enterprise Linux 6.1 リリースからは GFS2 で Linux 標準のクォータ機能に対応するようになります。GFS2 でのクォータ管理に関する詳細は 「GFS2 のクォータ管理」 を参照してください。
gfs_tool	**tunegfs2** mount パラメーター **dmsetup suspend**	ファイルシステムに関する情報の設定、調整、収集を行います。 **tunegfs2** コマンドには Red Hat Enterprise Linux 6.2 リリースより対応しています。また、**gfs2_tool** コマンドもあります。
gfs_edit	**gfs2_edit**	ファイルシステムの内部構造を表示、出力、または編集します。**gfs2_edit** コマンドは GFS ファイルシステムと GFS2 ファイルシステムいずれにも使用できます。
gfs_tool setflag jdata/inherit_jdata	**chattr +j** (preferred)	ファイルまたはディレクトリーに対するジャーナリングを有効にします。
setfacl/getfacl	**setfacl/getfacl**	ファイルまたはディレクトリーのファイルアクセス制御リストを設定または取得します。
setfattr/getfattr	**setfattr/getfattr**	ファイルの拡張属性を設定または取得します。

GFS2 ファイルシステムコマンドに対応するオプションの全一覧は各コマンドの man ページでご覧ください。

1.4.2. GFS と GFS2 のその他の相異点

本セクションでは、 「GFS2 のコマンド名」 に説明されていない GFS と GFS2 の違いについて簡単に説明します。

コンテキスト依存のパス名

GFS2 ファイルシステムでは、ポイント先が変化するファイルやディレクトリーを指すシンボリックリンクを作成できるコンテキスト依存のパス名には対応していません。GFS2 でこのように機能させたい場合は、**mount** コマンドの **bind** オプションを使用します。GFS2 でのコンテキスト依存のパス名やバ

インドのマウントについての詳細は 「複数マウントの結合とコンテキスト依存のパス名」 を参照してください。

gfs2.ko モジュール

GFS ファイルシステムを実装するカーネルモジュールは **gfs.ko** です。 GFS2 ファイルシステムを実装するカーネルモジュールは **gfs2.ko** です。

GFS2 での Quota の施行を有効にする

GFS2 ファイルシステムでは、クォータの施行がデフォルトでは無効になっているため明示的に有効にする必要があります。クォータの施行を有効にするまたは無効にする方法については 「GFS2 のクォータ管理」 を参照してください。

データジャーナリング

GFS2 ファイルシステムでは、ファイルやディレクトリーに **j** フラグをセットしたり外したりする **chattr** コマンドの使用に対応しています。ファイルに **+j** フラグをセットするとそのファイルでのデータジャーナリングが有効になります。**+j** フラグをディレクトリーにセットするということは「jdata を継承させる」ということになります。つまり、このディレクトリー内にその後作成されるファイルやディレクトリーがすべてジャーナリングされることになります。ファイルでデータのジャーナリングを有効または無効にする場合は **chattr** コマンドの使用をお勧めします。

ジャーナルを動的に追加

GFS ファイルシステムでは、ジャーナルはファイルシステムの外に存在する埋め込まれたメタデータなので、ジャーナルを追加する前にファイルシステムを格納している論理ボリュームのサイズを拡張する必要があります。GFS2 ファイルシステムでは、ジャーナルはプレーンファイル (ただし隠しファイル) です。つまり、GFS2 ファイルシステムの場合、追加ジャーナル用の領域がファイルシステムに残っている限り、追加のサーバーがファイルシステムをマウントする際にジャーナルを動的に追加することができます。GFS2 ファイルシステムにジャーナルを追加する方法については 「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。

atime_quantum パラメーターの削除

GFS2 ファイルシステムは調整可能な **atime_quantum** パラメーターには対応していません。GFS ファイルシステムではこのパラメーターを使うと **atime** 更新の発生頻度を指定することができます。GFS2 では代替として **relatime** と **noatime** のマウントオプションに対応しています。GFS で **atime_quantum** パラメーターを使って設定する動作と同じような動作を設定する場合は **relatime** マウントオプションの使用をお勧めします。

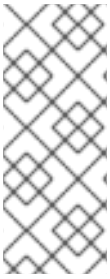
マウントコマンドの data= オプション

GFS2 ファイルシステムをマウントする場合、**mount** コマンドの **data=ordered** または **data=writeback** オプションを指定することができます。**data=ordered** を設定すると、トランザクションによって変更されたユーザーデータがディスクにフラッシュされてから、そのトランザクションがディスクにコミットされます。これにより、クラッシュ後、ファイル内に初期化されていないブロックが出現するのを防ぎます。**data=writeback** を設定すると、ユーザーデータはダーティになった後でも常にディスクに書き込まれます。**ordered** モードと同じ整合性の保証は得られませんが、一部の作業負荷に対して作業速度が若干速くなるはずです。デフォルトは **ordered** モードです。

gfs2_tool コマンド

gfs2_tool コマンドが GFS2 に対して対応しているオプションと **gfs_tool** コマンドが GFS に対して対応しているオプションとは異なります。

- **gfs2_tool** コマンドは、ファイルシステムが格納しているジャーナル数など、現在設定されているジャーナルに関する情報を表示する **journals** パラメーターに対応しています。
- **gfs2_tool** コマンドでは **counters** フラグには対応していません。このフラグは **gfs_tool** コマンドでは GFS の統計を表示する際に使用されます。
- **gfs2_tool** コマンドでは **inherit_jdata** フラグには対応していません。ディレクトリーに「jdata を継承させる」フラグを付ける場合は **jdata** フラグを設定するか、**chattr** コマンドを使って **+j** フラグを設定します。ファイルでデータのジャーナリングを有効または無効にする場合は **chattr** コマンドの使用をお勧めします。



注記

Red Hat Enterprise Linux 6.2 リリースからは GFS2 で **tunegfs2** コマンドに対応するようになります。**gfs2_tool** コマンドの一部機能がこのコマンドで行えるようになります。詳細については **tunegfs2** の man ページを参照してください。**gfs2_tool** コマンドの **settune** および **gettune** 関数は**mount** コマンドのコマンドラインオプションに置き換えられ、必要な時に **fstab** ファイルで設定できるようになりました。

gfs2_edit コマンド

gfs2_edit コマンドが GFS2 に対して対応しているオプションと **gfs_edit** コマンドが GFS に対して対応しているオプションとは異なります。コマンドの各バージョンが対応しているオプションについては **gfs2_edit** および **gfs_edit** の man ページをそれぞれ参照してください。

1.4.3. GFS2 のパフォーマンス向上

ユーザーインターフェースからは GFS ファイルシステムとの違いが見えませんが、GFS2 ファイルシステムにはファイルシステムのパフォーマンスを向上させる数多くの機能が搭載されています。

GFS2 ファイルシステムでは、ファイルシステムのパフォーマンスが以下のように向上しました。

- 単一のディレクトリーにおける頻繁な使用でのパフォーマンス向上
- 同期 I/O 操作の高速化
- キャッシュ読み込みの高速化（オーバーヘッドのロックなし）
- 事前に割り当てされたファイルの使用による、ダイレクト I/O の高速化 (4M など I/O サイズがある程度大きいことが前提)
- I/O 操作の全般的な高速化
- より迅速な **statfs** コールによる **df** コマンド実行の高速化
- **atime** モードの改善により、**atime** によって生成される書き込み I/O 操作回数を GFS よりも低減

GFS2 ファイルシステムはより幅広く、より多くの機能に対応します。

- GFS2 はアップストリームカーネルの一部になります (2.6.19 に統合)。

- GFS2 は以下のような機能に対応しています。
  - 拡張ファイル属性 (**xattr**)
  - 標準の **ioctl**() コールを介した **lsattr**() と **chattr**() の属性設定
  - ナノ秒タイムスタンプ

GFS2 ファイルシステムは、ファイルシステムの内部効率に対し以下のような点を改善しました。

- GFS2 によるカーネルメモリーの使用が低減されています。
- GFS2 ではメタデータ生成番号を必要としません。

  GFS2 メタデータの割り当てに読み込みを必要としません。複数ジャーナル内のメタデータブロックのコピーは、ロックが解放される前にジャーナルからブロックを無効にすることにより管理されます。
- GFS2 には、リンクされていない inode や quota 変更については認識しない、より簡単な ログマネージャが組み込まれています。
- **gfs2_grow** と **gfs2_jadd** のコマンドは、ロックを使用して複数インスタンスが同時に実行しないようにします。
- **creat**() や **mkdir**() のような呼び出しに対する ACL コードが簡素化されています。
- リンクされていない inode、クォータの変更、および **statfs** の変更は、ジャーナルを再マウントしなくても復元されます。

第2章 GFS2 の設定および操作における考慮事項

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

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

重要

Red Hat High Availability Add-On の導入が実際のニーズを満たすのか、また導入環境が対応可能であるのか必ず確認してください。導入する前に、Red Hat 認定の担当者と構成の検証を行ってください。

2.1. フォーマットに関する考慮事項

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

2.1.1. ファイルシステムサイズ: 小さい方が好ましい

GFS2 は、64-bit アーキテクチャーをベースにしており、理論的には 8 EB (エクサバイト) の大きさのファイルシステムの収容が可能です。ただし、64 ビットのハードウェア用の最大サイズとして現在対応しているのは 100 TB になります。また、32 ビットのハードウェアの場合は 16 TB になります。

GFS2 の大型ファイルシステムは可能ですが、それが推奨されるということではない点に注意してください。大ざっぱに言えばサイズは小さいほど好ましいとされています。つまり、10 TB のファイルシステムを 1 つ用意するつもりなら、1 TB のファイルシステムを 10 用意した方がよいということです。

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の所要メモリーが低減されます。

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ればなるほどパフォーマンスも向上します。

GFS2 ファイルシステムのサイズを小さくしすぎると当然、容量不足による不具合が発生する可能性があります。まず使用環境におけるユースケースを考慮してからサイズを決定するようにしてください。

2.1.2. ブロックサイズ: デフォルト (4K) ブロックを推奨

Red Hat Enterprise Linux 6 リリースからはデバイスのトポロジーに応じた最適なブロックサイズの算出が **`mkfs.gfs2`** コマンドにより試行されます。一般的には Linux のデフォルトページのサイズ (メモリー) が 4K のため、4K のブロックが推奨のブロックサイズになります。他のファイルシステムとは異

なり、GFS2 では大半の操作が 4K カーネルバッファーを使用して実行されます。このため、ブロックサイズを 4K にするとカーネル側でのバッファー操作の作業負担が緩和されることになります。

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

2.1.3. ジャーナル数: マウントするノードにつき 1 つ

GFS2 では、ファイルシステムをマウントする必要のあるクラスター内の 1 ノードにつき 1 ジャーナルが必要になります。たとえば、16 ノードで構成されるクラスターがあるとします。ファイルシステムをマウントさせるのは 16 ノードのうち 2 ノードのみの場合、必要となるジャーナル数も 2 つのみです。3 つ目のノードでマウントさせる必要が生じた場合は **gfs2_jadd** コマンドを使用すればいつでもジャーナルを追加することができます。GFS2 ではジャーナルを臨機応変に追加することができます。

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

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

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

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

2.1.5. リソースグループのサイズと数

GFS2 ファイルシステムを **mkfs.gfs2** コマンドで作成すると、ストレージが均一のサイズに分割されます。この分割された部分それぞれがリソースグループと呼ばれます。このリソースグループに最適なサイズの算出はコマンドによって試行されます (32MB から 2GB の範囲)。**mkfs.gfs2** コマンドの**-r** オプションを使用するとデフォルトを無効にすることができます。

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

最適なパフォーマンスとなるサイズを判断するためにはいろいろなサイズで試してみる必要があります。完全な実稼働環境に導入を行う前に、こうしたテストをテスト用クラスターで行って最適なサイズを確定するのが最良の方法となります。

使用するファイルシステムのリソースグループ数が多すぎると (リソースグループのサイズが小さすぎる)、空きブロックの検索を大量のリソースグループに対して行わねばならずブロック割り当てに無駄な時間がかかりすぎてしまう場合があります。ファイルシステムの使用率が高いほど、検索対象となるリソースグループも多くなり、そのリソースグループそれぞれにクラスター全体のロックが必要となるためパフォーマンス低下の原因となります。

しかし、ファイルシステムのリソースグループ数が少なすぎると (リソースグループのサイズが大きすぎる)、同じリソースグループをロックしようとブロック割り当ての競合が頻繁に発生する可能性があり、パフォーマンスにも影響を及ぼします。例えば、10GB の大きさのファイルシステムを 5 個のリソースグループ (2GB サイズ X 5) に分割した場合と 320 個のリソースグループ (32MB サイズ x 320) に分割した場合とで比較してみます。クラスター内のノードの競合がより激しくなるのは 5 個のリソースグループの方です。また、ブロック割り当ての度、数が少ないリソースグループ内で空きブロックがあるリソースグループを探そうとするため、ファイルシステムがほぼ満杯の状態になるとさらに問題は悪化します。GFS2 ではこのような問題を以下の 2 種類の方法で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、GFS2 はその情報を記憶してそれ以降からの割り当てではそのリソースグループを検索しないようにします (ブロックが解放されるまで)。ファイルの削除をまったく行わなければ競合度は緩和されます。ただし、ブロックの削除と新規ブロックの割り当てを絶えず行っているアプリケーションがあり、この動作がほぼ満杯状態のファイルシステムで行われると競合度が極めて高くなり、パフォーマンスに深刻な悪影響を及ぼすことになります。

- 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 (ファイルへの書き込み追加など)、GFS2 では新しいブロックをそのファイルの一部として同じリソースグループ内にまとめようとします。こうすることでパフォーマンスの向上を図ります。つまり回転しているディスクではブロック同士が物理的に近い距離にあった方が検索にかかる時間も少なくなるということです。

最悪の事例は、全ノードがファイルの作成を行う中心的なディレクトリーがひとつある場合です。この場合、すべてのノードが同じリソースグループを常にロックしようと競合してしまいます。

2.2. ファイルシステムの断片化

Red Hat Enterprise Linux 6.4 から、GFS2 でファイルの断片化管理を向上できるようになっています。Red Hat Enterprise Linux 6.4 では、同時書き込みによるファイルの断片化を低減しパフォーマンスの向上を図っています。

Red Hat Enterprise Linux では GFS2 向けのデフラグツールがありませんが、filefrag ツールでファイルを識別して一時ファイルにコピーし、この一時ファイルの名前を変更することでオリジナルファイルを置き換えるという手順を踏んで、個別ファイルをデフラグすることができます (この手順は、書き込みが順次行われている限り Red Hat Enterprise Linux 6.4 以前のバージョンでも可能です)。

2.3. ブロック割り当てにおける問題

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

2.3.1. ファイルシステムに空き領域を確保する

GFS2 ファイルシステムがほぼ満杯となると、ブロックアロケーターが割り当てる新規ブロック用の領域を見つけるのが難しくなります。このため、アロケーターによって割り当てられたブロックはリソースグループの末尾またはごく小さなスライスにに詰め込まれるため、ファイルが断片化する可能性がはるかに高くなります。このようなファイルの断片化はパフォーマンス上の問題を引き起こす場合があります。また、GFS2 がほぼ満杯となると、GFS2 のブロックアロケーターが複数のリソースグループを検索するため時間がかかり、十分な空き領域のあるファイルシステムでは必ずしも発生しないロック競合が生じます。これはパフォーマンス上の問題をもたらす可能性があります。

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

2.3.2. 可能な場合には各ノードに独自のファイルを割り当てる

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

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

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

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

2.3.3. 可能な場合には事前に割り当てる

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

2.4. クラスターに関する考慮事項

システムを構成するノードの数を決定する際には、高可用性とパフォーマンスの間でトレードオフがある点に注意してください。ノード数が増えるにしたがって、ワークロードのスケーリングが困難となります。このため、Red Hat は 16 ノードを越えるクラスターファイルシステムの導入に対しては 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のデプロイは、単一ノードデプロイメントと単純に置き換えることはできません。新規インストールには、8-12 週間程度のテスト期間を設けて、システムをテストし、必要とされるパフォーマンスレベルで動作することを確認することをお勧めします。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。

クラスター導入を検討中のお客様には、Red Hat サポートによる構成のレビューを導入前に受けていただくことをお勧めします。これにより、後日にサポート上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

GFS2 ファイルシステムをマウントする際は一般的には **noatime** および **nodiratime** の引数を使用することを推奨しています。これにより、アクセスの度に生じる GFS2 によるディスク inode の更新時間が低減されます。

2.5.2. DLM チューニングオプション: DLM テーブルサイズを拡大

DLM はクラスター内のノード間でのロック情報の管理、調整、受け渡しなどに複数のテーブルを使用します。この DLM テーブルのサイズを大きくすることでパフォーマンスが向上する場合があります。Red Hat Enterprise Linux 6.1 以降では、これらのテーブルのデフォルトサイズが拡大されていますが、以下のコマンドを使用して手動で拡大することも可能です。

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

これらのコマンドは、永続的ではなく、再起動後には無効となるため、起動スクリプトに追加して、GFS2 ファイルシステムをマウントする前に実行する必要があります。さもなければ、変更は警告なしに無視されてしまいます。

GFS2 ノードのロックに関する詳細は 「GFS2 のノードロック機能」 を参照してください。

2.5.3. VFS チューニングオプション: リサーチと実験

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

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

以下のコマンドで値を調整します。

```
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ためには、各種 VFS オプションを十分に調査しまずテスト用クラスター上で実験を行ってから完全な実稼働環境に導入してください。

2.5.4. SELinux: GFS2 では SELinux の使用を避ける

大半の状況で Security Enhanced Linux (SELinux) の使用が安全確保のため強く推奨されていますが、GFS2 での使用には対応していません。SELinux はすべてのファイルシステムのオブジェクトに関する拡張属性を使用して情報を格納します。この拡張属性の読み取り、書き込み、メンテナンスは可能ですが、GFS2 の動作速度がかなり低下してしまいます。このため GFS2 ファイルシステムでは SELinux を無効にするようにしてください。

2.5.5. GFS2 で NFS をセットアップする

GFS2 のロックサブシステムの複雑性とクラスター化の特性のため、GFS2 での NFS のセットアップには充分な事前準備と細心の注意を払う必要があります。本セクションでは、GFS2 ファイルシステムで NFS サービスを設定するにあたって考慮すべき注意事項について説明します。

警告

GFS2 ファイルシステムが NFS のエクスポートで、かつ NFS クライアントアプリケーションが POSIX ロックを使用する場合、**`localflocks`** オプションを使用してファイルシステムをマウントする必要があります。これが意図する効果は、各サーバーからの POSIX ロックがローカル (つまり、クラスター化されず、相互に独立した状態) となるように強制することです (GFS2 が NFS から クラスターのノード全体に POSIX ロックの実装を試みると数多くの問題があります)。NFS クライアント上で稼働す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、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できるということになります。すべてのクライアントが単一のサーバーから NFS をマウントしている場合、複数のサーバーが同じロックを別々に許可するという問題はなくなります。**`localflocks`** オプションでファイルシステムをマウントするべきかどうかがわからない場合には、このオプションは使用しない方がよいでしょう。ロックはクラスターベースで機能する方が安全です。

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

- Red Hat で対応しているのは、以下のような特性を持つアクティブ/パッシブ構成のロックを備えた NFSv3 を使用する Red Hat High Availability Add-On 設定のみになります。
  - バックエンドファイルシステムは、2 〜 16 ノードのクラスター上で稼働する GFS2 ファイルシステムです。
  - NFSv3 サーバーは、単一クラスターノードから GFS2 ファイルシステム全体を一度にエクスポートするサービスとして定義されます。
  - NFS サーバーは、一つのクラスターノードから他のクラスターノードへのフェイルオーバーが可能です (アクティブ/パッシブ構成)。
  - NFS サーバー経由 **以外の** GFS2 ファイルシステムへのアクセスは許可されません。これには、ローカルの GFS2 ファイルシステムアクセスと、Samba または クラスター化された Samba を介したアクセスの両方が含まれます。
  - システム上で NFS クォータはサポートされていません。

  この構成では、ノードに障害が発生しても、NFS サーバーがノードからノードへフェイルオーバーすることにより、**`fsck`** コマンドを実行する必要は生じないため、ファイルシステムに高可用性を提供し、システムダウンタイムを削減します。
- **`fsid=`** NFS オプションは、GFS2 の NFS エクスポートには必須です。
- クラスターに問題が生じた場合 (例: クラスタが定足数に達せず、フェンシングが失敗した場合)

には、クラスター化された論理ボリュームと GFS2 ファイルシステムはフリーズされ、クラスターが定足数を満たすようになるまではアクセスできません。この手順で説明したような、単純なフェイルオーバー解決方法がご使用のシステムに最も適切かどうかを判断する際には、この可能性を考慮する必要があります。

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

Red Hat Enterprise Linux 6.2 リリースでは、アクティブ/アクティブ構成が可能な CTDB を使用する GFS2 ファイルシステムから Samba (SMB または Windows) ファイルサービスを使用することができます。クラスター化された Samba 設定についての情報は 『クラスター管理』 のドキュメントを参照してください。

Samba 外から Samba 共有内のデータへの同時アクセスには対応していません。GFS2 クラスターのリースは Samba のファイル処理速度を低下させるため現在、対応していません。

2.6. ファイルシステムのバックアップ

ファイルシステムのサイズに関わらず問題が発生したときのために GFS2 ファイルシステムを定期的にバックアップするよう推奨しています。RAID、マルチパス、ミラーリング、スナップショット、その他の冗長形式などを備えているから安全だと思っている管理者も多くいますが絶対に安全ということはありません。

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

バックアップが完了した後にキャッシュを削除すると、他のノードがクラスターのロック/キャッシュのオーナーシップを再取得する時間が短縮されます。ただし、バックアッププロセスが開始する前に他のノードがキャッシュしていたデータのキャッシュが停止されることになるため、これでも理想的ではありません。以下のコマンドを実行すると、バックアップの完了後にキャッシュを削除することができます。

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

クラスター内の各ノードがそれぞれのファイルをバックアップした方がノード間でバックアップ作業が分割されるため時間が短縮されます。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を使うとこれを行うことができます。

GFS2 バックアップを作成する場合、SAN 上でハードウェアの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渡してそのシステムでバックアップを作成するのが最適な方法になります。バックアップを作成するシステムはクラスター内のノードではないため、**-o lockproto=lock_nolock** を使ってそのシステムにスナップショットをマウントさせます。

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導入する際には、以下のようなハードウェア考慮事項を検討する必要があります。

- より高品質なストレージオプションを使用する

  GFS2 は、iSCSI や Fibre Channel over Ethernet (FCoE) など低コストの共有ストレージオプションで稼働可能ですが、キャッシュ容量が大きな高品質のストレージを購入するとパフォー

マンスが向上します。Red Hat は、ファイバーチャンネル相互接続の SAN ストレージ上で大半の品質テスト、健全性テスト、パフォーマンステストを行っています。何かを導入する場合はまずテストを実施してから導入に移るというのが常に原則です。

- 導入前にネットワーク機器をテストする

  高品質の高速ネットワーク機器を使用することにより、クラスターの通信および GFS2 が高速化され、信頼性が向上しますが、一番高いハードウェアを選択して購入する必要はありません。高額なネットワークスイッチのなかには **`fcntl`** ロック (flock) の受け渡しに使用されるマルチキャストパケットの受け渡しで問題があるものがあります。一方、安価で普通のネットワークスイッチであっても高速で信頼できるものもあります。完全な実稼働環境に導入する前に機器をテストしておくのが一般的には原則です。

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 と Red Hat Global File System 2 (GFS2) を使った Red Hat Enterprise Linux クラスターの導入およびアップグレードに関する実践例の詳細を「Red Hat Enterprise Linux Cluster, High Availability, and GFS Deployment Best Practices (https://access.redhat.com/site/articles/40051)」で確認してください。

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非常に重要となります。単一ノードファイルシステムはキャッシュと共に実装され、頻繁に要求されるデータを使用する場合にディスクへのアクセスの待ち時間をなくすことを目的としています。Linux では、ページキャッシュ (および以前はバッファーキャッシュ) によりこのキャシング機能が提供されます。

GFS2 では各ノードに独自のページキャッシュがあり、オンディスクデータの一部が含まれていることがあります。 GFS2 は *glocks* (ジーロックスと発音) と呼ばれるロック機能のメカニズムを使用してノード間のキャッシュの整合性を維持します。glock サブシステムは、*分散型ロックマネージャー* (DLM) を下位の通信層として使用して実装される、キャッシュ管理機能を提供します。

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

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

別のノードが即時に許可できない glock を要求した場合、DLM はその glock を保持しているため新しい要求の妨げとなっているノードにメッセージを送り、その glock をドロップするよう要求します。glock のドロップは時間がかかるプロセスとなる場合があります (大半のファイルシステム操作の基準で判断した場合)。共有の glock をドロップする場合はキャッシュの無効化だけなので比較的短時間で完了し、キャッシュされたデータ量に比例します。

glock を限定してドロップするには、ログをフラッシュして変更されたデータをディスクに書き戻した後、共有 glock ごとに無効化を行う必要があります。

単一ノードファイルシステムと GFS2 で異なるのは、単一ノードファイルシステムのキャッシュは一つであるのに対して、GFS2 にはノードごとに別個のキャッシュがある点です。いずれの場合も、キャッシュ済みデータへのアクセスの待ち時間は同じ程度ですが、キャッシュされていないデータへのアクセ

スに関しては、別のノードが以前に同じデータをキャッシュしていた場合、GFS2 の方がはるかに長くなります。

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

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

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

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

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

2.9.1. POSIX ロックの問題

POSIX ロックを使用する場合、以下の点を考慮する必要があります。

- 処理速度は Flock を使用した方が POSIX ロックより早くなります。
- クラスター環境では指定のプロセス ID が別のノード用である場合もあるため、GFS2 で POSIX ロックを使用するプログラムは、**GETLK** 関数の使用を避けるべきです。

2.9.2. GFS2 によるパフォーマンス調整

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

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

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

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

バックアップは、難しいことが多いもう一つの領域です。可能であれば、特定の inode セットをキャッシュしているノードから直接、各ノードの作業セットをバックアップしておくことを強く推奨します。定時に実行するバックアップスクリプトを使用していて、それが GFS2 で実行しているアプリケーションの応答時間の急増と同時に発生している場合には、クラスターがページキャッシュを効率的に使用していない可能性が高くなります。

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

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

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

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

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

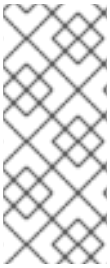
本セクションには、GFS2 ロックダンプの概要を記載しています。GFS2 ロックダンプについての詳しい説明は、付録C *GFS2 トレースポイントおよび debugfs glocks ファイル* を参照してください。

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

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

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

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



注記

debugfs ファイルのコピーは 2 つ取っておくと役立つ場合があります。その際、2 つ目のコピーは、最初のコピーから数秒または数分遅らせます。同じ glock 番号に関連する 2 つのトレースでホルダー情報を比較すると、作業負荷が進行しているか (つまり、単なる速度低下)、スタックしているか (その場合は必ずバグなので、Red Hat サポートに直ちに報告する必要があり) を見極めることができます。

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。ホルダー行のフラグフィールド f: には、待機中の要求を示す「W」フラグ、または承認済み要求を

示す「H」フラグが表示されます。待機中の要求が多数ある glocks は特定の競合が発生している可能性が高くなります。

表2.1「glock フラグ」 では各 glock フラグが示す意味を説明しています。表2.2「Glock ホルダーフラグ」 では各 glock ホルダーフラグが示す意味を glock ダンプに出現する順序で説明しています。

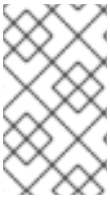
表2.1 glock フラグ

フラグ	名前	意味
b	Blocking	ロックのフラグがセットされていると有効、DLM から要求された動作はブロックされる可能性があるという意味、(このフラグは降格の動作や「try」ロックでクリアになります。他のノードによるロック降格に要される時間とは別に DLM 応答時間の統計データを収集することがフラグの目的です。)
d	Pending demote	遅延 (リモートの) 降格の要求
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のコミットが必要
F	Frozen	リモートノードからの返信を無視 - 復元の進行中、(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ありません。復元中にしか使用されません。)
i	Invalidate in progress	この glock のページの無効化が進行中
I	Initial	DLM がこの glock に関連付けられるとセットされる
l	Locked	glock は状態を変更中
L	LRU	LRU 一覧に glock が記載されるとセットされる
o	Object	glock がオブジェクトに関連付けられるとセットされる (type 2 glock の場合 inode、type 3 glock の場合リソースグループ)
p	Demote in progress	glock は降格要求に応答中
q	Queued	glock に対してホルダーがキュー待ちになるとセットされる、glock が行われキュー待ちのホルダーがなくなると消去される、(glock 維持の最小時間を計算するアルゴリズムの一部として使用されます。)
r	Reply pending	リモートノードから受信した返信の処理の待機中
y	Dirty	この glock を解放する前にディスクへのデータのフラッシュが必要

表2.2 Glock ホルダーフラグ

フラグ	名前	意味
a	Async	glock の結果を待たない (結果を後でポールします)
A	Any	互換性のあるロックモードはすべて受け入れ可能
c	No cache	ロック解除時に DLM ロックを即時降格
e	No expire	後続のロック取り消し要求を無視
E	exact	完全一致するロックモード
F	First	このロックにホルダーを最初に付与する必要がある場合にセットされる
H	Holder	要求したロックが付与されたことを示す
p	Priority	キューの先頭にある待機ホルダー
t	Try	「try」ロック
T	Try 1CB	コールバックを送信する「try」ロック
W	Wait	要求完了の待機中にセットされる

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



注記

ロック競合が発生している時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 を検索する場合は、まず先にアプリケーションを停止してから **`find`** を実行してください。

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

表2.3 glock のタイプ

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

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

特定された glock が異なるタイプだった場合に、最も可能性が高いのはタイプ 3: (リソースグループ) です。通常の負荷時に別のタイプの glock を待機しているプロセスを多数確認した場合は、Red Hat サポートにご報告ください。

リソースグループロックで数多くの待機要求が待ち行列に入っている場合は、複数の理由が考えられます。 その一つとして、ノード数がファイルシステム内のリソースグループ数に比べて多い場合があります。 また、ファイルシステムがほぼ完全に満杯の場合も考えられます (空きブロックの検索に大抵時間がかかる)。いずれの場合も、ストレージを追加した上で **gfs2_grow** コマンドを使っ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 使用開始

この章では、GFS2 の初期セットアップの手順を説明します。以下のセクションで構成されます。

- 「事前に必要な作業」
- 「初期セットアップの作業」

3.1. 事前に必要な作業

Red Hat GFS2 を設定する前に以下の作業を完了しておいてください。

- GFS2 ノードに持たせる主要な特徴をメモしておきます (「GFS2 を設定する前に」 参照)。
- 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 Network Time Protocol (NTP) ソフトウェアを使用することを推奨します。

注記

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

- GFS2 をクラスター環境で使用するには、LVM 論理ボリュームマネージャーのクラスタリング拡張機能セットである Clustered Logical Volume Manager (CLVM) を使用するようにシステムを設定する必要があります。CLVM を使用するには、**`clvmd`** デーモンを含む Red Hat Cluster Suite のソフトウェアを実行している必要があります。CLVM の使用に関する詳細は、『論理ボリュームマネージャの管理』 を参照してください。Red Hat Cluster Suite のインストールおよび管理に関する詳細は 『Cluster Administration』 を参照してください。

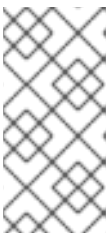
3.2. 初期セットアップの作業

GFS2 の初期セットアップは、以下のような作業で構成されます。

1. 論理ボリュームのセットアップ
2. GFS2 ファイルシステムの作成
3. ファイルシステムのマウント

以下の手順に従って GFS2 の初期セットアップを行います。

1. LVM を使用して、各 Red Hat GFS2ファイルシステムに論理ボリュームを作成します。



注記

Red Hat Cluster Suite に同梱されている **`init.d`** スクリプトを使用すると論理ボリュームを有効にしたり無効にしたりできます。**`init.d`** スクリプトの詳細は **Configuring and Managing a Red Hat Cluster (Red Hat Cluster の設定と管理)** を参照してください。

2. 手順 1 で作成した論理ボリューム上に GFS2 ファイルシステムを作成します。ファイルシステム名はそれぞれ固有の名前にしてください。GFS2 ファイルシステムの作成方法については「ファイルシステムの作成」 を参照してください。

   次のいずれかを使ってクラスター化した GFS2 ファイルシステムを作成します。

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

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

   GFS2 ファイルシステムの作成方法については 「ファイルシステムの作成」 を参照してください。

3. 各ノードで GFS2 のファイルシステムをマウントします。 GFS2 ファイルシステムをマウントする方法については 「ファイルシステムのマウント」 を参照してください。

   コマンドの使用法:

   mount *BlockDevice MountPoint*

   mount -o acl *BlockDevice MountPoint*

   -o acl のマウントオプションを使用すると、ファイルの ACL を操作することができます。**-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。

   注記

   Red Hat High Availability Add-On に同梱の **init.d** スクリプトを使用すると GFS2 ファイルシステムのマウントおよびアンマウントを自動化することができます。

第4章 GFS2 の管理

本章は GFS2 を管理するための作業とコマンドを説明するため以下のセクションで構成されています。

- 「ファイルシステムの作成」
- 「ファイルシステムのマウント」
- 「ファイルシステムのアンマウント」
- 「GFS2 のクォータ管理」
- 「ファイルシステムの拡張」
- 「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」
- 「ファイルシステム上の動作を一時停止する」
- 「ファイルシステムの修復」
- 「複数マウントの結合とコンテキスト依存のパス名」
- 「複数マウントの結合とファイルシステムのマウント順序」
- 「GFS2 の withdraw 関数」

4.1. ファイルシステムの作成

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

- プロトコル/モジュールのロック名（クラスターのロックプロトコルは **lock_dlm**）
- クラスター名（クラスター設定の一部として実行している場合）
- ジャーナルの数（ファイルシステムをマウントするノード 1 つにつき、ジャーナルが 1 つ必要）

GFS2 ファイルシステムを作成する場合、直接 **mkfs.gfs2** コマンドを使用して作成するか、**mkfs** コマンドに ***-t*** パラメーターを付けファイルシステムのタイプを **gfs2** に指定して作成することもできます。

注記

mkfs.gfs2 コマンドで一旦 GFS2 ファイルシステムを作成するとそのファイルシステムのサイズを縮小することはできなくなります。 ただし、 「ファイルシステムの拡張」に説明されているように **gfs2_grow** コマンドを使用すると既存ファイルシステムのサイズを拡大することはできます。

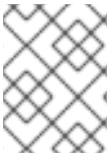
使用法

次のいずれかを使ってクラスター化された GFS2 ファイルシステムを作成します。

```
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は次のずれかを使用します。



注記

Red Hat Enterprise Linux 6 リリースの場合、Red Hat は GFS2 を単一ノードのファイルシステムとして使用する方法には対応していません。

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

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

警告

上記のコマンドの使用は ***LockProtoName*** および ***LockTableName*** のパラメーターの使い方を十分に理解している場合に限ってください。***LockProtoName*** と ***LockTableName*** のパラメーターを不適切に使用をすると、ファイルシステムまたはロックスペースが破損する可能性があります。

LockProtoName

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

LockTableName

クラスター設定の GFS2 ファイルシステム用に指定するパラメーターです。***ClusterName:FSName*** などのように、2 種類の値をコロンで区切って指定します (空白なし)。

- ***ClusterName***: クラスター名、GFS2 ファイルシステムを作成するクラスターを指定します。
- ***FSName***: ファイルシステムの名前、1 文字から 16 文字までの長さで指定します。クラスター全体の **lock_dlm** ファイルシステムおよびローカルの各ノードの全ファイルシステム (**lock_dlm** と **lock_nolock**) で固有となる名前にしてください。

Number

mkfs.gfs2 コマンドで作成するジャーナルの数を指定します。ファイルシステムをマウントするノード毎に 1 つのジャーナルが必要です。「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しなくても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、クラスター化ファイルシステムなので、**lock_dlm** はファイルシステムが使用するロックプロトコルを指定しています。クラスター名は **alpha**、ファイルシステム名は **mydata1** にしています。ジャーナルの数は 8 つ、作成先に **/dev/vg01/lvol0** を指定しています。

```
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
```

次に 2 番目の **lock_dlm** ファイルシステムを作成します。クラスター **alpha** 内で使用するよう指定しています。**mydata2** というファイルシステム名を付け、ジャーナルの数は 8 つ、作成先に **/dev/vg01/lvol1** を指定しています。

```
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4.1「コマンドオプション: **mkfs.gfs2**」 で **mkfs.gfs2** コマンドのオプションを説明します (フラグとパラメーター)。

表4.1 コマンドオプション: mkfs.gfs2

フラグ	パラメーター	説明
-c	***Megabytes***	各ジャーナルのクォータ変更ファイルの初期サイズを ***Megabytes*** に設定します。
-D		デバッグの出力を有効にします。
-h		ヘルプ。使用可能なオプションを表示します。
-J	***MegaBytes***	ジャーナルのサイズをメガバイトで指定します。デフォルトのジャーナルサイズは 128 メガバイトです。最低サイズは 8 メガバイトです。ジャーナルが大きいほどパフォーマンスは向上しますが、メモリーの消費も小さいジャーナルに比べ増大します。

フラグ	パラメーター	説明
-j	***Number***	**mkfs.gfs2** コマンドで作成されるジャーナルの数を指定します。ファイルシステムをマウントするノード毎に 1 つのジャーナルが必要になります。このオプションを指定しないと作成されるジャーナルは 1 になります。GFS2 ファイルシステムの場合はファイルシステムを拡張しなくても後でジャーナルを追加することができます。
-O		**mkfs.gfs2** コマンドでファイルシステムへの書き込み前に確認プロンプトを表示しないようにします。
-p	***LockProtoName***	使用するロックプロトコルの名前を指定します。認識できるロックプロトコルは以下のプロトコルです。 **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で指定します。リソースグループの最小サイズは 32 MB です。リソースグループの最大サイズは 2048 MB です。超大型のファイルシステムでは、リソースグループのサイズを大きくするとパフォーマンスを向上させることができます。リソースグループのサイズを指定しなかった場合には、mkfs.gfs2 によりファイルシステムのサイズに基づいてリソースグループのサイズが選択されます。平均的なサイズのファイルシステムでは 256 MB のリソースグループとなります。より大きなファイルシステムの場合は、パフォーマンスを向上させるために、リソースグループサイズがより大きくなります。

フラグ	パラメーター	説明
-t	***LockTableName***	**lock_dlm** プロトコルを使用する場合のロックテーブルフィールドの指定に使用する固有の識別子です。**lock_nolock** プロトコルの場合は使用しません。 ***ClusterName:FSName*** のように 2 種類の値をコロンで区切って指定します (空白なし)。 ***ClusterName***、作成する GFS2 ファイルシステムを使用させるクラスター名です。このクラスターのメンバーにのみこのファイルシステムの使用が許可されます。クラスター名は **Cluster Configuration Tool (クラスター設定ツール)** で **/etc/cluster/cluster.conf** ファイル内に設定され、Red Hat Cluster Suite のクラスター管理 GUI 内の **Cluster Status Tool (クラスターステータスツール)** に表示されます。 ***FSName***、ファイルシステム名です。1 文字から 16 文字までの長さにします。クラスター内の全ファイルシステムで固有となる名前にします。
-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

4.2. ファイルシステムのマウント

GFS2 ファイルシステムをマウントする前に、そのファイルシステムが存在していること (「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になっていること、クラスタのシステムとロックのシステムをサポートする機能が起動していること (**Configuring and Managing a Red Hat Cluster (Red Hat Cluster の設定と管理)** 参照) を確認します。これらの要件が揃っていることを確認した上で、Linux ファイルシステムと同様に GFS2 ファイルシステムをマウントします。

注記

Cluster Manager (**cman**) が起動していない時に GFS2 ファイルシステムのマウントを試みると、以下のようなエラーメッセージが表示されます。

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

ファイル ACL を操作するには **-o acl** マウントオプションを付け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付けずにファイルシステムをマウントすると、ユーザーは ACL の表示 (**getfacl**)はできますが ACL の設定 (**setfacl**) ができなくなります。

使用法

ACL 操作なしのマウント

```
mount BlockDevice MountPoint
```

ACL 操作が可能なマウント

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

-o acl
ACL のファイル操作を許可する GFS2 固有のオプション

BlockDevice
GFS2 ファイルシステムが存在しているブロックデバイス

MountPoint
GFS2 ファイルシステムのマウント先となるディレクトリー

例

この例では、**/dev/vg01/lvol0** にある GFS2 ファイルシステムを **/mygfs2** ディレクトリーにマウントしています。

```
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4.2「GFS2 固有のマウントオプション」 を参照)、使用可能な標準の Linux **mount -o** オプションのいずれかまたは両方の組み合わせになります。複数の ***option*** パラメーターを使用する場合はコンマで区切り空白は入れません。

注記

mount コマンドは Linux システムのコマンドです。このセクションで説明している GFS2 固有のオプションの他にも、標準の **mount** コマンドのオプション （**-r**) も使用できます。Linux **mount** コマンドのオプションについては Linux の **mount** man ページでご覧ください。

表4.2「GFS2 固有のマウントオプション」 ではマウント時に GFS2 へ渡すことのできる GFS2 固有の **-o option** 値を説明します。

注記

この表にはローカルのファイルシステムで使用するオプションしか記載していません。ただし、Red Hat Enterprise Linux 6 リリースの場合、Red Hat では単一ノードのファイルシステムとしての GFS2 の使用については対応していないので注意してください。クラスターファイルシステムのスナップショットのマウントを目的とした単一ノード GFS2 ファイルシステムについては引き続きサポートしています (バックアップなど)。

表4.2 GFS2 固有のマウントオプション

オプション	説明
acl	ファイル ACL の操作を可能にします。 **acl** マウントオプションを指定せずにファイルシステムをマウントした場合、ユーザーは ACL の表示 (**getfacl**) はできますが、設定 (**setfacl**) はできません。
data=[ordered|writeback]	**data=ordered** を設定すると、トランザクションによって変更されたユーザーデータは、トランザクションがディスクにコミットされる前にディスクにフラッシュされます。これは、クラッシュの後にファイル内で初期化されていないブロックがユーザーに表示されるのを防ぎます。**data=writeback** モードが設定されている場合は、ユーザーデータはダーティとなった後でも、随時ディスクに書き込まれます。これは、**ordered** モードと同じ一貫性保証は提供しませんが、一部のワークロードでは若干速くなるはずです。デフォルト値は **ordered** モードです。
ignore_local_fs **注意：** GFS2 ファイルシステムを共有する場合はこのオプションは **使用しないでください**。	GFS2 がファイルシステムをマルチホストファイルシステムとして扱うように強制します。デフォルトでは、**lock_nolock** を使用すると **localflocks** のフラグが自動的に有効になります。
localflocks **注意：** GFS2 ファイルシステムを共有する場合は個のオプションは使用しないでください。	VFS (virtual file system) レイヤーですべての flock と fcntl を実行するよう、GFS2 に指示します。**localflocks** フラグは **lock_nolock** によって自動的に有効になります。

オプション	説明
lockproto=*LockModuleName*	ユーザーが、ファイルシステムで使用するロックプロトコルを指定できるようにします。***LockModuleName*** が指定されていない場合は、ファイルシステムのスーパーブロックからロックプロトコル名が読み込まれます。
locktable=*LockTableName*	ユーザーがファイルシステムで使用するロックテーブルを指定できるようにします。
quota=[off/account/on]	ファイルシステムのクォータのオン/オフを切り替えます。**account** の状態となるようにクォータを設定すると、UID/GID 毎の使用状況の統計はファイルシステムによって正しく維持され、上限と警告の値は無視されます。デフォルト値は **off** です。
errors=panic|withdraw	**errors=panic** を指定するとファイルシステムのエラー発生時にはカーネルパニックが起こります。デフォルトの動作は **errors=withdraw** です。この場合、withdraw が作動してファイルシステムが使用不可になり次回の再起動までアクセスできないようになります。場合によっては、システムが稼働したままの可能性もあります。GFS2 の withdraw 関数については 「GFS2 の withdraw 関数」 を参照してください。
discard/nodiscard	解放されたブロックに対して I/O 要求の「破棄」を生成します。対応するハードウェアで使用するとシンプロビジョニングや同様のスキームを実装することができます。
barrier/nobarrier	ジャーナルをフラッシュする際に I/O バリアを送信します。デフォルト値は **on** です。ベースのデバイスが I/O バリアに対応していない場合は自動的に **off** になります。書き込みキャッシュの内容の消失しないよう設計されているブロックデバイスでない限り (UPS 上にある場合や書き込みキャッシュがない場合)、GFS2 では I/O バリアを常に使用することを強く推奨します。
quota_quantum=*secs*	クォータ情報に関する変更がクォータファイルに書き込まれるまでに一つのノードに留めることができる秒数を設定します。パラメーターの設定には、この方法が推奨されます。値はゼロより大きい整数になります。デフォルトは 60 秒です。これより短く設定すると遅延クォータ情報の更新速度が速くなるため、そのクォータを越えてしまう可能性が少なくなります。長く設定すると、クォータに伴うファイルシステムの動作速度が高速化され、効率性が向上します。

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** の変更が **statfs** のマスターファイルに同期されるまでの最大時間を設定します。速度を高くして **statfs** の正確性を低くしたり、速度を低くして正確性を高めたりするなどの調整が可能です。このオプションを 0 に設定すると **statfs** は常に true の値を報告するようになります。
statfs_percent=*value*	有効期間が切れていない場合でも、 **statfs** のマスターファイルに戻って同期するまでにローカルベースでの **statfs** 情報の変更率の上限を指定します。**statfs_quantum** の設定が 0 の場合はこの設定は無視されます。

4.3. ファイルシステムのアンマウント

GFS2 ファイルシステムは他の Linux ファイルシステムと同様、**umount** コマンドを使ってアンマウントすることができます。

注記

umount コマンドは Linux システムのコマンドです。 このコマンドについては Linux **umount** コマンドの man ページをご覧ください。

使用法

```
umount MountPoint
```

MountPoint

GFS2 ファイルシステムが現在マウントされているディレクトリーです。

4.4. GFS2 ファイルシステムをマウントする際の注意事項

GFS2 ファイルシステムを **fstab** ファイル内のエントリによる自動マウントではなく手動でマウントしていた場合、システムのシャットダウンでファイルシステムのアンマウントが行われる際にシステムで認識されません。このため、GFS2 スクリプトによる GFS2 ファイルシステムのアンマウントが行われないことになります。GFS2 シャットダウンスクリプトを実行すると、標準のシャットダウンプロセスでクラスターのインフラストラクチャーなど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が、クラスターのインフラストラクチャーがないためアンマウントは失敗し、システムがハングすることになります。

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

- GFS2 ファイルシステムをマウントする場合は常に **/etc/fstab** ファイル内のエントリを使用します。

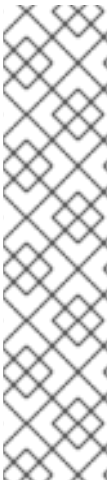
- GFS2 ファイルシステムを **mount** コマンドを使って手作業でマウントした場合はシャットダウンまたは再起動を行う前に必ず **umount** コマンドを使って手作業でファイルシステムをアンマウントします。

上記を行っている上で、システムのシャットダウン時にアンマウントが行われるとシステムがハングしてしまう場合はハードウェアの再起動を行ってください。 ファイルシステムはシャットダウンプロセスの前半で同期されるためデータが消失する可能性はないでしょう。

4.5. GFS2 のクォータ管理

ファイルシステムのクォータは、ユーザーやグループが使用できるファイルシステムのサイズを制限するのに使われます。これが設定されるまでは、ユーザーやグループのクォータには上限がありません。**quota=on** または **quota=account** のオプションで GFS2 ファイルシステムがマウントされると、上限が設定されていない場合でも、GFS2 は 各ユーザーとグループが使用する領域を追跡します。GFS2 は、システムがクラッシュしてもクォータの使用状況を再構築する必要がないように、トランザクションの形でクォータ情報を更新します。

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



注記

Red Hat Enterprise Linux 6.1 リリースからは GFS2 で Linux 標準のクォータ機能に対応するようになります。このクォータ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。GFS2 でクォータを管理する場合は推奨の管理方法になります。また、GFS2 の新規導入でクォータを使用する場合はこのクォータ機能を使用してください。このセクションでは、この機能を使用した GFS2 クォータ管理について説明します。

Red Hat Enterprise Linux の旧バージョンでは、GFS2 でクォータを管理するには **gfs2_quota** コマンドを使用する必要がありました。**gfs2_quota** コマンドの使い方については 付録A *gfs2_quota コマンドを使用した GFS2 のクォータ管理* を参照してください。

4.5.1. ディスククォータの設定

以下の手順にしたがってディスククォータを実装します。

1. 施行モードまたはアカウントモードでクォータを設定します。
2. 現在のブロック使用率の情報が含まれる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 (アカウントモードの場合はポリシーは適用されません)。

各ステップについて詳しく見ていきます。

4.5.1.1. 施行モードまたはアカウントモードでクォータを設定する

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

上限や警告の値は施行せずディスクの使用状況を追跡し全ユーザーおよびグループのクォータアカウント情報を維持することもできます。**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。

使用法

クォータ施行モードでファイルシステムをマウントするには、**quota=on** オプションを指定してファイルシステムをマウントします。

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

クォータの上限は施行せずにクォータアカウント情報の維持モードでファイルシステムをマウントするには、**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。

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

クォータ無効モードでファイルシステムをマウントするには、**quota=off** 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。これがデフォルトの設定です。

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

quota={on|off|account}

on - ファイルシステムをマウントしたときクォータが有効になります。

off - ファイルシステムをマウントしたときクォータが無効になります。

account - ファイルシステムでユーザーとグループのディスク使用状況の統計を維持管理します。クォータの上限は施行されません。

BlockDevice

GFS2 ファイルシステムが存在しているブロックデバイス

MountPoint

GFS2 ファイルシステムのマウント先となるディレクトリー

例

以下の例では、**/dev/vg01/lvol0** にある GFS2 ファイルシステムが **/mygfs2** ディレクトリーにマウントされクォータが施行されます。

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

以下の例では、**/dev/vg01/lvol0** にある GFS2 ファイルシステムが **/mygfs2** ディレクトリーにマウントされクォータアカウント情報が維持管理されますが上限は施行されません。

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

4.5.1.2. クォータデータベースファイルを作成する

各ファイルシステムをクォータ有効モードでマウントすると、システムはディスククォータ付きで動作することができるようになります。ただし、ファイルシステム自体はまだクォータに対応する準備が整っていない状態です。次のステップは **quotacheck** コマンドの実行です。

quotacheck コマンドは、クォータが有効になるファイルシステムを検証し、ファイルシステムごとの現在のディスク使用状況一覧を構築します。この表を使ってディスク使用状況に関するオペレーティングシステムのコピーが更新されます。また、ファイルシステムのディスククォータファイルが更新されます。

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** コマンドの **-u** と **-g** のオプションを使用します。ユーザーおよびグループ両方のクォータを初期化するため両オプションを指定する必要があります。たとえば、**/home** ファイルシステムにクォータを実施している場合には、**/home** ディレクトリーにファイルを作成します。

```
quotacheck -ug /home
```

4.5.1.3. ユーザーごとのクォータ割り当て

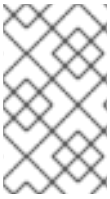
最後のステップは、**edquota** コマンドを使ってディスククォータを割り当てる作業です。ファイルシステムをアカウントモードでマウントしている場合には (**quota=account** オプションを指定) クォータは施行されないので注意してください。

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

```
edquota username
```

クォータを必要とするユーザーごとにこのステップを実行します。たとえば、クォータを **/etc/fstab** 内で **/home** パーティションに対して有効にしている場合 (以下の例では **/dev/VolGroup00/LogVol02**)、コマンド **edquota testuser** を実行すると、エディターで以下のようなシステムのデフォルト設定が表示されます。

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



注記

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

第 1 列はクォータを有効化にしているファイルシステムの名前です。第 2 列はユーザーが現在使用中のブロック数です。その後の 2 列はファイルシステム上のユーザーにソフトおよびハードのブロックリミット (上限) を設定する場合に使用されます。

ソフトブロックリミットは、使用可能な最大ディスク容量を定義します。

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

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

いずれかの値が 0 にセットされている場合、そのリミットは設定されていないことになります。テキストエディターで希望の上限に変更します。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

4.5.1.4. グループごとのクォータ割り当て

クォータはグループごとに割り当てることも可能です。ファイルシステムをアカウントモードでマウントしている場合は (**quota=account** オプションを指定) クォータは施行されないので注意してください。

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

```
edquota -g devel
```

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

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

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

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

```
quota -g devel
```

4.5.2. ディスククォータの管理

クォータを実装する場合はメンテナンスが必要になります。大半はクォータが超過していないか監視したり、クォータの精度を確認するといった形になります。

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
```

クォータを有効にしているすべてのファイルシステム (オプション **-a**) のディスク使用状況レポートを表示するには、以下のコマンドを使用します。

```
repquota -a
```

レポートはわかりやすい表示になっていますがいくつか説明しておくべき点があります。各ユーザーの後ろに表示される **--** でブロックリミットを超過しているかどうかが即時に判断できます。ブロックソフトリミットを超過している場合には最初の **-** の場所に**+** が表示されます。2 番目の **-** は inode リミットを示しますが、GFS2 ファイルシステムでは inode リミットに対応していないため、この箇所は **-** のままとなります。GFS2 ファイルシステムでは猶予期間に対応してないため、**grace** の列は空欄のままとなります。

NFS を経由させている場合は、ベースとなる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ないので注意してください。

4.5.3. クォータの精度維持

クォータが無効な状態で一定期間ファイルシステムを運用した後に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し、確認と修復を行ってください。また、クォータファイルが正確でないと思われる場合にも **quotacheck** を実行して確認することができます。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と精度が失われる問題が発生することがあります。

quotacheck コマンドの詳細は **quotacheck** の man ページを参照してください。

注記

クォータの計算値はディスクのアクティビティ状態に影響され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になっている時に実行してください。

4.5.4. quotasync コマンドを使用したクォータの同期

GFS2 ではクォータ情報はすべてディスク上にある独自の内部ファイルに格納されます。GFS2 ノードはファイルシステムの書き込みのたびにこのクォータファイルを更新するのではなく、デフォルトでは 60 秒ごとにクォータファイルの更新を行うようになっています。これはノード間でクォータファイルへの書き込み競合が発生しパフォーマンスの低下を招かないようにするため不可欠になります。

ユーザーまたはグループのクォータが上限に近づくと、GFS2 は、クォータファイルの更新間隔を動的に短縮して上限の超過を防止します。クォータ同期の通常の間隔は、調整可能なパラメーター **quota_quantum** です。表4.2「GFS2 固有のマウントオプション」 で説明しているように、**quota_quantum=** マウントオプションを使用するとデフォルト値の 60 秒を変更することができます。**quota_quantum** パラメーターはファイルシステムがマウントされるたび各ノードに設定しなければなりません。**quota_quantum** に行った変更はアンマウント後は維持されません。**quota_quantum** 値は **mount -o remount** を使用して更新することができます。

GFS2 で実行される自動的な更新の合間にノードのクォータ情報をディスク上のクォータファイルに同期する場合は **gfs2_quota sync** コマンドを使用します。

使用法

クォータ情報の同期

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

u

ユーザーのクォータファイルを同期します。

g

グループのクォータファイルを同期します。

a

現在クォータが有効になっていて同期に対応している全ファイルシステムを同期します。-a を使用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してください。

mntpnt

動作を適用する 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

動作を適用する GFS2 ファイルシステムを指定します。

secs

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

例

以下の例では、コマンドが実行されるノードのキャッシュ済みダーティクォータをファイルファイルシステム **/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) に変更しています。

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

4.5.5. 参考情報

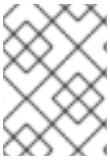
ディスククォータに関する詳細は以下のコマンドの **man** ページを参照してください。

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

4.6. 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常駐するデバイスを拡大した後に、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、新たに初期化された GFS2 ファイルシステムの拡張により、ファイルシステムの現在の終端とデバイスの終端との間の予備領域がすべて埋められます。この充填操作が終了すると、ファイルシステムのリソースインデックスが更新されます。この後に、クラスター内のすべてのノードが追加ストレージ領域を使用できるようになります。

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



注記

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

使用法

```
gfs2_grow MountPoint
```

MountPoint

動作を適用する 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次の作業を行ってください。

- 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、拡張対象の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 でベースとなるクラスターボリュームの拡張を行います。LVM ボリュームの管理については 『Logical Volume Manager Administration (論理ボリュームマネージャの管理)』 を参照してください。

gfs2_grow コマンドを実行した後、ファイルシステム内で新たに拡張した領域が使用できるようになっているか **df** コマンドを使って確認します。

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が拡張されています。

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

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です。

Device

ファイルシステムのデバイスノードです。

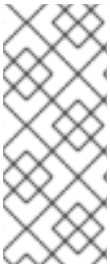
表4.3「ファイルシステムを拡張する際に使用できる GFS2 固有のオプション」 では、GFS2 ファイルシステムを拡張する際に使用できる GFS2 固有のオプションについて説明しています。

表4.3 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

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
-q	非表示、出力表示を少なくする
-r MegaBytes	新規のリソースグループサイズを指定、デフォルトサイズは 256 MB
-T	テスト、計算だけを行いディスクへのデータ書き込みおよびファイルシステムの拡張などは行いません
-V	コマンドのバージョン情報を表示

4.7. ファイルシステムへジャーナルの追加

GFS2 ファイルシステムにジャーナルを追加する場合に **gfs2_jadd** コマンドを使用します。ベースとなる論理ボリュームを拡張せずにいつでも動的に GFS2 ファイルシステムにジャーナルを追加することができます。**gfs2_jadd** コマンドはマウントしているファイルシステムで実行しなければなりませんが、この作業を行う必要があるのはクラスター内の 1 ノードのみです。他のノードはすべて、ファイルシステムが拡張されたことを認識します。



注記

GFS2 ファイルシステムを格納する論理ボリュームが拡張されそのファイルシステムより大きいサイズにしてあったとしても、GFS2 ファイルシステムが満杯になっている場合は **gfs2_jadd** コマンドを使用しても失敗します。GFS2 ファイルシステムではジャーナルは埋め込まれたメタデータではなくプレーンなファイルとなるため、単にベースの論理ボリュームを拡張してもジャーナル用の領域は確保できないためです。

ジャーナルを GFS ファイルシステムに追加する前に、**gfs2_tool** コマンドの **journals** オプションを使用すると GFS2 ファイルシステムに現在格納されているジャーナル数を確認することができます。以下の例では、**/mnt/gfs2** にマウントしているファイルシステムのジャーナル数とそのサイズを表示しています。

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

使用法

```
gfs2_jadd -j Number MountPoint
```

Number

新規に追加するジャーナル数です。

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です。

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 -j1 /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 -j2 /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です。

Device

ファイルシステムのデバイスノードです。

表4.4「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4.4 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、デフォルトのジャーナルサイズは 128 メガバイトで最小サイズは 32 メガバイト (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場合は各サイズのジャーナル毎に **gfs2_jadd** コマンドを実行する、ファイルシステム作成時に指定したジャーナルセグメントのサイズの倍数になるようサイズの端数を切り捨てる)
-j	***Number***	**gfs2_jadd** コマンドで追加する新規ジャーナルの数、デフォルト値は 1
-q		非表示、出力表示を少なくする
-V		コマンドのバージョン情報を表示

4.8. データジャーナリング

通常、GFS2 はそのジャーナルにメタデータのみを書き込みます。この後、ファイルの内容がファイルシステムバッファをフラッシュするカーネルの定期的な同期によってディスクに書き込まれます。任意のファイルで **fsync()** を呼び出すと、そのファイルのデータがディスクに即時に書き込まれます。ディスクにより全データが安全に書き込まれたことが報告されると、この呼び出しが返されます。

データのジャーナリングを行うと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、特に小さなファイルの場合は **fsync()** にかかる時間を短縮することができます。ファイルサイズが大きくなるとこの利点は急激に少なくなります。データジャーナリング機能をオンにすると、中型および大型ファイルへの書き込み速度がかなり遅くなります。

ファイルデータの同期を **fsync()** に依存してするアプリケーションの場合、データジャーナリングを使用するとパフォーマンスが向上する場合があります。フラグを付けたディレクトリー (および、その全サブディレクトリー) 内に作成される GFS2 ファイルはすべてデータジャーナリングが自動的に有効になります。また、長さゼロの既存ファイルについてもデータジャーナリングをオンまたはオフにすることができます。

任意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に「inherit jdata」がセットされます。このディレクトリー内に以後作成されるファイルやディレクトリーがすべてジャーナリングされるという意味です。**chattr** コマンドを使うと任意のファイルのデータジャーナリングを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** ファイルでデータジャーナリングを有効にし、フラグが正しくセットされたか確認しています。

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

以下のコマンドは、**/mnt/gfs2/gfs2_dir/newfile** ファイルでデータジャーナリングを無効にし、フラグが正しくセットされたか確認しています。

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

また、**chattr** コマンドを使用してディレクトリーに **j** フラグを設定することもできます。このフラグをディレクトリーに設定すると、そのディレクトリーで以降に作成されるファイルやディレクトリーはすべてジャーナリングされます。以下の一連のコマンドでは **gfs2_dir** ディレクトリーに **j** フラグを設定し、そのフラグが正しくセットされたか確認しています。次に、**/mnt/gfs2/gfs2_dir** ディレクトリー内に **newfile** と言う新しいファイルを作成、そのファイルに **j** フラグが正しく設定されたか確認しています。**j** フラグをディレクトリーに設定したので、**newfile** にもジャーナリングが有効になっているはずです。

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

4.9. ATIME 更新の設定

ファイル inode とディレクトリー inode にはそれぞれに関連付けられた 3 種類のタイムスタンプがあります。

- **ctime** — inode のステータスが最後に変更された時刻
- **mtime** — ファイル (またはディレクトリー) の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) の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 ファイルシステムでデフォルトになっているように **atime** 更新が有効の場合は、ファイルが読み込まれる度に inode の更新が必要になります。

atime で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ありません。この更新により書き込みおよびファイルのロックに大量の不要なトラフィックが伴う場合があり、これが原因でパフォーマンスが低下する可能性があります。したがって、**atime** 更新はオフにするかその頻度を減らした方が良いでしょう。

atime 更新の頻度を減らす方法が 2 通りあります。

- **relatime** (relative atime) でマウントする方法、前回の **atime** が **mtime** または **ctime** より古い場合に **atime** を更新します。
- **noatime** でマウントする方法、そのファイルシステムでは **atime** 更新を無効にします。

4.9.1. relatime でマウントする方法

Linux のマウントオプション **relatime** (relative atime) は、ファイルシステムをマウントするときに指定できます。指定すると、前回の **atime** が **mtime** または **ctime** よりも古い場合に **atime** が更新されます。

使用法

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

BlockDevice

GFS2 ファイルシステムが存在しているブロックデバイス

MountPoint

GFS2 ファイルシステムのマウント先となるディレクトリー

例

以下の例では、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** の更新は前回の **atime** が **mtime** または **ctime** より古い場合にのみ行われます。

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

4.9.2. noatime でマウントする

Linux のマウントオプション **noatime** は、ファイルシステムをマウントするときに指定できます。そのファイルシステムでの **atime** 更新を無効にします。

使用法

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

BlockDevice

GFS2 ファイルシステムが存在しているブロックデバイス

MountPoint

GFS2 ファイルシステムのマウント先となるディレクトリー

例

以下の例では、GFS2 ファイルシステムは **/dev/vg01/lvol0** に存在し、**/mygfs2** ディレクトリーにマウントされます。**atime** 更新は無効になります。

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

4.10. ファイルシステム上の動作を一時停止する

dmsetup suspend コマンドを使用すると、ファイルシステムへの書き込み動作を一時停止することが

できます。書き込み動作を一時停止することにより、ハードウェアベース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ファイルシステムをキャプチャーすることができるようになります。一時停止を終了するには、**dmsetup resume** コマンドを実行します。

使用法

一時停止の開始

```
dmsetup suspend MountPoint
```

一時停止の終了

```
dmsetup resume MountPoint
```

MountPoint

ファイルシステムを指定します。

例

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

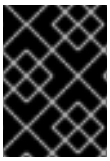
```
# dmsetup suspend /mygfs2
```

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

```
# dmsetup resume /mygfs2
```

4.11. ファイルシステムの修復

ファイルシステムをマウントした状態でノードに障害が発生した場合はファイルシステムのジャーナリングで迅速な復旧が可能です。しかし、ストレージデバイスへの電力供給が断たれたり物理的に電源が切断されるとファイルシステムの破損が発生する可能性があります (ストレージサブシステムの障害からの復旧にジャーナリングは使用できません)。この種の破損が発生した場合は、**fsck.gfs2** コマンドを使用して GFS2 ファイルシステムの復旧を行います。



重要

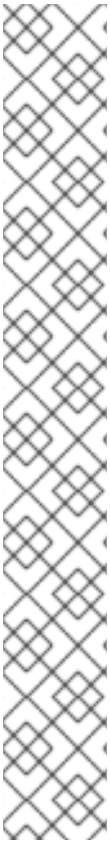
fsck.gfs2 コマンドの実行は、必ずすべてのノードからファイルシステムをアンマウントした上で行ってください。

重要

fsck.gfs2 コマンドは、起動時の GFS2 ファイルシステムのチェックには使用しないでください。**fsck.gfs2** コマンドでは、起動時、ファイルシステムがクラスター内の別のノードでマウントされているかどうかを判断することができません 。**fsck.gfs2** コマンドはシステムの起動後、手動で行うようにしてください。

起動時、GFS2 ファイルシステムで **fsck.gfs2** コマンドが実行されないようにするため、**/etc/fstab** ファイルを変更します。以下の例のように、GFS2 ファイルシステムのマウントポイントの最後の 2 つのコラムが "1 1" (またはそれ以外の数字) ではなく "0 0" となるよう編集します。

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



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には、**fsck.gfs2** コマンドが以下の点で **gfs_fsck** の旧リリースとは異なる点に注意してください。

- **fsck.gfs2** の実行中に **Ctrl+C** を押すと処理が中断され、コマンドを中止して現在残っているパスを省略するのか、処理を続行するのかを尋ねるプロンプトが表示されます。
- **-v** フラグを使用すると詳細を表示させることができます。**-v** フラグを 2 つ付けると更に表示する詳細度が高くなります。
- **-q** フラグを使用すると詳細を表示しなくなります。**-q** フラグを 2 つ付けると更に表示する詳細度が低くなります。
- **-n** オプションを使用すると、ファイルシステムを読み取り専用で開きすべてのクエリーに対して自動的に **no** と返します。このオプションは **fsck.gfs2** の機能を実際には実施せず、エラーを発見するだけの目的でコマンドを実行させたい場合に使用します。

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を確認してそれに 5/8 をかけます。

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** コマンドの実行に必要なメモリー数を概算する場合は、まず 16TB を 4K で割り算してファイルシステムに含まれているメモリーのブロック数を確認します。

```
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 をかけ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ということになります。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、つまり 11 GB の空きメモリーが必要になります。

使用法

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

-y

-y フラグを設定すると全質問に対して **yes** と返します。**-y** フラグを指定した場合は変更に対する答えを入力するプロンプトが表示されません。

BlockDevice

GFS2 ファイルシステムが存在しているブロックデバイス

例

この例では、ブロックデバイス **/dev/testvol/testlv** に存在している GFS2 ファイルシステムが修復されます。修復に関する質問はすべて自動的に **yes** で回答されます。

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

4.12. 複数マウントの結合とコンテキスト依存のパス名

GFS2 ファイルシステムではコンテキスト依存のパス名 (CDPN) には対応していません (コンテキスト依存のパス名を使用すると可変のファイルやディレクトリーをポイントするシンボリックリンクを作成することが可能)。GFS2 では、**mount** コマンドの **bind** オプションを使用すると同様の動作を行わせ

ることができます。

mount コマンドの **bind** オプションを使用すると、ファイル階層の一部をオリジナルの場所で使用可能な状態のまま別の場所でも再マウントできるようになります。コマンドの形式を以下に示します。

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

このコマンドを実行すると、***olddir*** ディレクトリーのコンテンツは ***olddir*** と ***newdir*** の 2 ヶ所で使用できるようになります。またこのオプションを使用するとひとつのファイルが 2 ヶ所でアクセスできるようになります。

例えば、以下のコマンドを実行すると、**/root/tmp** のコンテンツが以前にマウントした **/var/log** ディレクトリーの内容と全く同じになります。

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

別の方法として、**/etc/fstab** ファイル内のエントリーを使用してもマウント時に同じことが行えます。以下の **/etc/fstab** エントリーの場合、**/root/tmp** の内容は **/var/log** ディレクトリーの内容と全く同じになります。

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

ファイルシステムをマウントしたら、以下の例のように **mount** コマンドを使用してファイルシステムがマウントされているか確認することができます。

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

コンテキスト依存のパス名に対応しているファイルシステムでは **/bin** ディレクトリーにコンテキスト依存のパス名を定義している場合があります。この場合、システムアーキテクチャーに応じて以下のパスのいずれか 1 つに解決されます。

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

同じような動作をさせるため空の **/bin** ディレクトリーを作成します。次にスクリプトまたは **/etc/fstab** ファイル内の エントリーを使用して **mount -bind** コマンドで各アーキテクチャーのディレクトリーを **/bin** ディレクトリーにマウントします。例えば、次のコマンドをスクリプト内の 1 行に使用します。

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

または、**/etc/fstab** ファイル内で以下のエントリーを使用します。

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

定義する基準 (ファイルシステムの **%fill** の値など) に応じて異なる複数のディレクトリーをマウントすることができるため、マウントの結合によりコンテキスト依存のパス名より高い柔軟性を得ることが

できます。コンテキスト依存のパス名の場合は対象範囲がより限定されています。ただし、**%fill** の値などの基準に応じて、独自のマウント用スクリプトを記述する必要があるので注意してください。

警告

bind オプションでファイルシステムをマウントしたときにオリジナルのファイルシステムが **rw** でマウントされている場合、**ro** フラグを使用してもこのファイルシステムは **rw** でマウントされることになります。 つまり、**ro** フラグは警告なしに無視されます。このような場合には、**/proc/mounts** ディレクトリー内でそのファイルシステムに **ro** の印が付いている可能性があります。このため意図しない結果となっている場合があります。

4.13. 複数マウントの結合とファイルシステムのマウント順序

mount コマンドの **bind** オプションを使用する場合、ファイルシステムが必ず正しい順序でマウントされるよう確認してください。次の例の場合、まず **/var/log** ディレクトリーを先にマウントしてから **/tmp** ディレクトリーで bind を使ってマウントを実行する必要があります。

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

ファイルシステムのマウント順序は次のように決定されます。

- 一般的には、ファイルシステムのマウント順序は **fstab** ファイルに出現する順序で決定されます。**_netdev** フラグを付けてマウントされるファイルシステム、独自の **init** スクリプトを持っているファイルシステムについては例外となります。
- 独自の **init** スクリプトを持つファイルシステムは初期化プロセスの後半、**fstab** ファイルに記載されているファイルシステムがマウントされた後にマウントが行われます。
- **_netdev** フラグを付けてマウントされるファイルシステムはシステムでネットワークが有効化された際にマウントされます。

bind を使って GFS2 ファイルシステムをマウントする必要があるような構成の場合には、以下のように **fstab** ファイルで順序付けすることができます。

1. bind を使ったマウントに必要なローカルのファイルシステムをマウントします。
2. GFS2 ファイルシステムをマウントするディレクトリーを bind を使ってマウントします。
3. GFS2 ファイルシステムをマウントします。

ローカルのディレクトリーやファイルシステムを GFS2 ファイルシステムに bind を使ってマウントしなければならない構成の場合、**fstab** ファイルに正しいマウント順でファイルシステムを記載しても、GFS2 **init** スクリプトが実行されるまで GFS2 ファイルシステムがマウントされないため、記載したファイルシステムは正しくマウントされません。この場合、GFS2 ファイルシステムがマウントされるまで待機してから bind を使ったマウントを実行するような **init** スクリプトを記述する必要があります。

以下にカスタムの **init** スクリプトの例を示します。このスクリプトにより 2 つの ディレクトリーが

GFS2 ファイルシステムの 2 つのディレクトリーに bind を使ってマウントされます。**/mnt/gfs2a** に既存の GFS2 マウントポイントがあり、クラスターの起動後、GFS2 **init** スクリプトが実行されるとマウントされます。

スクリプト内の **chkconfig** の値の意味は次の通りです。

- 345 - スクリプトが起動するランレベル
- 29 - 起動の優先度、起動時に起動優先度が 26 の GFS2 **init** スクリプトが実行された後、起動優先度 29 のこのスクリプトが実行されます
- 73 - 停止の優先度、シャットダウン時に停止優先度が 73 のこのスクリプトが先に停止されてから、停止優先度が 74 の GFS2 スクリプトが停止されます

起動と停止の値は **service start** および **service stop** のコマンドを実行すると、指定した操作を手動で実行可能であることを示しています。たとえば、スクリプト名が **fredwilma** の場合には、**service fredwilma start** を実行することができます。

このスクリプトは **/etc/init.d** ディレクトリー内に他のスクリプトと同じパーミッションを与えて配置してください。次に **chkconfig on** コマンドを実行してスクリプトを指定のランレベルにリンクします。たとえば、スクリプト名が **fredwilma** なら **chkconfig fredwilma on** を実行します。

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

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

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

  status)
        ;;

  restart)
        $0 stop
```

```
        $0 start
        ;;

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

exit 0
```

4.14. GFS2 の WITHDRAW 関数

GF2S の *withdraw* 関数は、クラスター内における GFS2 ファイルシステムのデータ整合性機能です。GFS2 カーネルモジュールが、I/O 操作の後に GFS2 ファイルシステムで不整合を検出すると、クラスターはそのファイルシステムを使用できなくなります。その I/O 操作は停止して、システムはさらなる I/O 操作がエラーで停止するのを待ち、それ以上の被害を防ぎます。このような事態が発生した場合は、その他のサービスやアプリケーションをすべて手動で停止した後に、GFS2 ファイルシステムの再起動と再マウントを行い、ジャーナルを再生することができます。問題が解決しない場合は、クラスター内のすべてのノードからファイルシステムをアンマウントして、**fsck.gfs2** コマンドでファイルシステムのリカバリを行います。GFS の withdraw 関数は、カーネルパニックほど深刻ではありません。カーネルパニックが発生した場合には、問題のあるノードは別のノードによってフェンシングされます。

gfs2 起動スクリプトを有効にして GFS2 ファイルシステムを **/etc/fstab** ファイルに含ませているようなシステム構成の場合、再起動を行うと GFS2 ファイルシステムも再マウントされます。GFS2 ファイルシステムに破損が検出され withdraw が作動してそのファイルシステムが使用できなくなった場合には、ファイルシステムの再マウントを行う前に **fsck.gfs2** コマンドを実行することを推奨します。この場合、以下の手順で起動時にファイルシステムが再マウントされないようにします。

1. 次のコマンドを使用して影響を受けたノードで起動スクリプトを一時的に無効にします。

   ```
   # chkconfig gfs2 off
   ```

2. 影響を受けたノードを再起動してクラスターソフトウェアを開始します。GFS2 ファイルシステムはマウントされません。

3. クラスター内のすべてのノードからファイルシステムをアンマウントします。

4. 壊れたファイルシステムがないことを確認するため、ひとつのノードのファイルシステムのみから **fsck.gfs2** を実行します。

5. 次のコマンドを実行して、影響を受けたノードで起動スクリプトを再度有効にします。

   ```
   # chkconfig gfs2 on
   ```

6. クラスター内の全ノードから GFS2 ファイルシステムを再度マウントします。

withdraw の作動で GFS2 が使用できなくなるような不整合の例のひとつとしてブロック数の不一致があります。GFS カーネルがファイルシステムからファイルを削除する場合、そのファイルに関連する全データおよびメタデータのブロックが体系的に削除されます。この動作が終了すると、ブロック数の

チェックが行われます。カウントされたブロック数が 1 (残されたのはディスクの inode 自体のみ) にならない場合、ブロック数が検出されたブロック一覧と一致しないためファイルシステムが不整合であるということになります。

-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と、GFS2 の withdraw 関数を上書きすることができます。このオプションを指定すると、通常 withdraw が作動するエラーでパニックが発生することになります。パニックでノードのクラスター通信が停止するためそのノードの排他処理が行われます。

内部では、カーネルが **gfs_controld** デーモンにメッセージを送信して withdraw を要求することにより GFS2 の withdraw 関数が作動します。**gfs_controld** デーモンは **dmsetup** プログラムを実行してデバイスマッパーエラーのターゲットをファイルシステム下に配置し、これ以上ブロックデバイスがアクセスされないようにします。次にこの動作が終了したことをカーネルに伝えます。CLVM デバイスを常に GFS2 下で使用することが GFS2 のサポート要件になっているのはこのためです。GFS2 下で CLVM デバイスを使用していない場合デバイスマッパーターゲットを挿入できなくなります。

デバイスマッパーエラーターゲットの目的はそれ以降の I/O 操作がすべて I/O エラーになることでファイルシステムを順番通りにアンマウントさせることです。このため、withdraw が作動するとデバイスマッパーデバイスからの多数の I/O エラーがシステムログにレポートされるのが見られますがこれが通常です。

dmsetup プログラムが要求通りにエラーターゲットを挿入できない場合、withdraw の作動に失敗する場合があります。withdraw 作動の時点でメモリー不足が発生している場合や、最初の段階で withdraw を作動させた問題が原因でメモリーを再利用できないなどの場合に withdraw の作動に失敗する可能性があります。

withdraw が作動したからといって必ずしも GFS2 にエラーがあるということではありません。ベースとなっているブロックデバイス関連のデバイス I/O エラーにより withdraw 関数が作動することもあります。withdraw が作動した場合には、GFS2 のエラーなのかそれ以外のエラーなのかを判断するためログを確認することを強く推奨します。

第5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解説しています。

5.1. 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。

5.2. GFS2 ファイルシステムがハングし単一ノードの再起動を必要とする

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は戻ってこないが、特定のノードを再起動するとシステムが正常な状態に戻る場合には、ロックの問題もしくはバグの兆候である可能性があります。このような事態が発生した場合には、以下のデータを収集してください。

- 各ノード上のファイルシステム用の gfs2 ロックダンプ

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

- 各ノード上のファイルシステム用の DLM ロックダンプ: この情報は **dlm_tool** を使用すると取得できます。

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

  上記コマンドの *lsname* にはハングしているファイルシステムの DLM で使用されているロックスペース名を入力します。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入力してください。

5.3. GFS2 ファイルシステムがハングし全ノードの再起動を必要とする

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が戻ってこないため、使用できる状態にするにはクラスター内の全ノードを再起動する必要がある場合には以下の問題を確認してください。

- 排他処理機能に障害が発生している可能性があります。排他処理機能に障害が発生した場合、GFS2 ファイルシステムはデータの整合性を確保するためフリーズします。メッセージログを確認し、ハング時に排他処理機能に障害が発生していなかったか確認してください。排他処理機能のフェンスが正しく設定されているか確認してください。

- withdraw が作動し GFS2 ファイルシステムが使用できなくなっている可能性があります。メッセージログ中に **withdraw** という単語がないか、withdraw が作動してファイルシステムが使用できなくなったことを示す GFS2 のメッセージやコールトレースがないか調べます。withdraw の作動はファイルシステムの破損、ストレージの障害、またはバグなどの兆候になります。ファイルシステムをアンマウントして **gfs2-utils** パッケージを更新し、**fsck** コマンドをファイルシステム上で実行してファイルシステムが再び正しく動作するようにします。Red Hat サポートのチケットを作成し GFS2 の withdraw が作動した旨の報告をお願いします。また sosreport にログを付けて送信してください。

GFS2 の withdraw 関数の詳細は 「GFS2 の withdraw 関数」 を参照してください。

- ロック関連の問題が発生したかバグの可能性があります。発生中にデータを収集し、「GFS2 ファイルシステムがハングし単一ノードの再起動を必要とする」 の記載にしたがって Red Hat サポートのチケットを作成してください。

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

クラスターに新規ノードを追加したあとそのノードで GFS2 ファイルシステムをマウントできないことがわかった場合には、GFS2 ファイルシステムにアクセスしようとしているノード数より GFS2 ファイルシステムのジャーナル数の方が少ない可能性があります。ファイルシステムをマウントする GFS2 ホストごとに 1 つのジャーナルが必要です (**spectator** マウントオプションを付けてマウントした GFS2 ファイルシステムにはジャーナルが必要ないため例外となります)。GFS2 ファイルシステムにジャーナルを追加するには、「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_jadd** コマンドを使用します。

5.5. 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 ファイルシステムで **df** コマンドを使用すると占有されている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によってディスクの領域 (ジャーナル数 x ジャーナルサイズ) が消費されているためです。GFS2 ファイルシステムを作成したとき多数のジャーナルを持たせた、またはジャーナルのサイズを大きなサイズにした場合、**df** を実行すると (ジャーナル数 x ジャーナルサイズ) の領域が既に使用中として表示されます。特にたくさんのジャーナル数や大きなジャーナルサイズを指定したわけではなくても、GFS2 ファイルシステム自体が小さいと (1 GB 以下) デフォルトの GFS2 ジャーナルのサイズであってもかなりの割合の領域が使用中として示されます。

第6章 PACEMAKER クラスターでの GFS2 ファイルシステムの設定

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を簡単に説明します。

各ノードにクラスター化した LVM パッケージ、クラスターソフトウェア、GFS2 をインストールしたら **cman**、**clvmd**、**pacemaker** のサービスを各ノードで起動して Pacemaker 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してください。Pacemaker クラスターの設定方法については『Configuring the Red Hat High Availability Add-On with Pacemaker (Pacemaker を使った Red Hat High Availability Add-On の設定)』 を参照してください。

1. Pacemaker のグローバルパラメーター **no_quorum_policy** を **freeze** にセットします。

注記

no-quorum-policy はデフォルトでは **stop** にセットされます。つまり、定足数を失うと残りのパーティション上の全リソースが直ちに停止されることになります。一般的にはこのデフォルトが最も安全で最適なオプションとなりますが、一般的なリソースとは異なり、GFS2 が正しく機能するには定足数が必要です。定足数を失うと、GFS2 マウントを使用しているアプリケーションおよび GFS2 マウント自体の両方が正しく停止できなくなります。定足数がないままこうしたリソースを停止しようとすると停止に失敗します。最終的には定足数を失う度にクラスター全体が排他処理されることになります。

このような状況に対処するため、GFS2 を使用している場合は **no-quorum-policy=freeze** をセットします。これにより、定足数が失われた場合、定足数を取り戻すまで残りのパーティションは何の動作も行わなくなります。

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

2. クラスター化ロックに対応するようロックタイプが **/etc/lvm/lvm.conf** ファイル内で 3 にセットされているか確認した後、クラスター化論理ボリュームを作成して、そのボリュームを GFS2 ファイルシステムで初期化します。クラスター内の各ノードには必ず十分なジャーナルを作成してください。

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

3. **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。マウントポイントの指定は **options=*options*** を使ってリソース設定の一部として行います。設定オプションの全一覧を表示させる場合は **pcs resource describe Filesystem** コマンドを実行します。

   以下のクラスターリソース作成コマンドではマウントオプションに **noatime** を指定しています。

```
# pcs resource create clusterfs Filesystem
```

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

4. 期待通り、GFS2 がマウントされているか確認します。

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

5. (オプション) 全クラスターノードを再起動して GFS2 が永続的であり復元機能が動作することを確認します。

付録A GFS2_QUOTA コマンドを使用した GFS2 のクォータ管理

Red Hat Enterprise Linux 6.1 リリースからは GFS2 で Linux 標準のクォータ機能に対応するようになります。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。GFS2 でクォータを管理する場合は推奨の管理方法になります。また、GFS2 の新規導入でクォータを使用する場合はこのクォータ機能を使用してください。Linux 標準のクォータ機能の使い方については 「GFS2 のクォータ管理」 を参照してください。

Red Hat Enterprise Linux の旧リリースでは、**gfs2_quota** コマンドを使用して GFS2 のクォータを管理する必要がありました。この付録では、**gfs2_quota** コマンドを使用した GFS2 ファイルシステムのクォータ管理について説明します。

A.1. GFS2_QUOTA コマンドを使用したクォータの設定

ユーザー ID (UID) またはグループ ID (GID) ごとに、*ハードリミット* と *ソフトリミット* の 2 つのクォータ設定が使用できます。

ハードリミットは、使用可能な容量です。ファイルシステムは、ユーザーやグループに対して、このディスク領域サイズを越える容量の使用は許可しません。ハードリミット値が **ゼロ** の場合は、上限が適用されていないことになります。

通常、ソフトリミットはハードリミットの値を下回ります。ファイルシステムは、ソフトリミットに達すると、ユーザーまたはグループに通知して、使用量を警告します。ソフトリミットの値が **ゼロ** の場合は、上限が適用されていないことになります。

gfs2_quota コマンドを使用して上限を設定することができます。このコマンドを実行する必要があるのは、GFS2 がマウントされている単一のノード上のみです。

デフォルトでは GFS2 ファイルシステムにはクォータの施行は設定されていません。クォータアカウントを有効にするには、「クォータ施行を有効/無効にする」 に記載したように GFS2 ファイルシステムをマウントする時に **mount** コマンドの **quota=** を使用します。

使用方法

クォータの設定 (ハードリミット)

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

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

クォータの設定 (警告リミット)

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

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

User

制限または警告するユーザー ID。パスワードファイルからのユーザー名または UID 番号を使用できます。

Group

制限または警告するグループ ID。グループファイルからのグループ名または GID 番号を使用できます。

Size

制限または警告する値を指定します。この値はデフォルトではメガバイト単位になります。**-k**、**-s**、**-b** などのフラグを使用すると単位をキロバイト、セクター、ファイルシステムのブロック数などに変更できます。

MountPoint

動作を適用する GFS2 ファイルシステムです。

例

以下の例では、ファイルシステム **/mygfs2** 上のユーザー *Bert* に 1024 メガバイト (1 ギガバイト) のハードリミットを設定します。

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

以下の例では、ファイルシステム **/mygfs2** 上のグループ ID 21 に 50 キロバイトのソフトリミットを設定します。

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

A.2. GFS2_QUOTA コマンドを使用したクォータの上限と使用状況の表示

特定のユーザーやグループに関するクォータの上限と現在の使用状況を **gfs2_quota get** コマンドを使って表示します。クォータファイルの全内容については **gfs2_quota list** コマンドを使用すると表示できます。ハードリミット、ソフトリミット、値がゼロ以外に設定された全 ID が一覧表示されます。

使用方法

ユーザーのクォータ上限の表示

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

グループのクォータ上限の表示

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

クォータファイル全体の表示

```
gfs2_quota list -f MountPoint
```

User

特定のユーザーに関する情報を表示するためのユーザー ID。パスワードファイルからのユーザー名または UID 番号を使用できます。

Group

特定のグループに関する情報を表示するためのグループ ID。グループファイルからのグループ名または GID 番号を使用できます。

MountPoint

動作を適用する GFS2 ファイルシステムです。

コマンドの出力

gfs2_quota コマンドを使用すると、GFS2 クォータ情報は、次のように表示されます。

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

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

LimitSize、***WarnSize***、および ***Value*** の数 (値) は、デフォルトではメガバイト単位になっています。コマンドラインに **-k**、**-s**、または **-b** のフラグを追加すると、単位はそれぞれキロバイト、セクター、ファイルシステムブロックに変更されます。

User

データが関連付けされているユーザー名または ID。

Group

データが関連付けされているグループ名または ID。

LimitSize

ユーザーまたはグループに対して設定されるハードリミット。上限が設定されていない場合には、この値はゼロになります。

Value

ユーザーまたはグループが使用するディスク領域の実際のサイズ。

コメント

クォータ情報を表示する時に、**-n** オプションをコマンドラインに追加すると、**gfs2_quota** コマンドは、UID や GID を名前に解決しません。

コマンドラインに **-d** オプションを追加すると、GFS2 の隠しファイルに割り当てられている領域が root の UID および GID の表示値から除外されます。**gfs2_quota** の数値と **du** コマンドの結果とを照合する場合に便利です。

例

以下の例は、ファイルシステム **/mygfs2** 上のユーザーおよびグループで上限が設定されているかディスク領域を使用している全ユーザーおよびグループのクォータ情報が表示されます。

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

以下の例は、ファイルシステム **/mygfs2** 上のグループ **users** のクォータ情報をセクター単位で表示します。

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

A.3. GFS2_QUOTA コマンドを使用したクォータの同期

GFS2 ではクォータ情報はすべてディスク上にある独自の内部ファイルに格納されます。GFS2 ノードはファイルシステムの書き込みのたびにこのクォータファイルを更新するのではなく、デフォルトでは 60 秒ごとにクォータファイルの更新を行うようになっています。これはノード間でクォータファイルへの書き込み競合が発生しパフォーマンスの低下を招かないようにするため不可欠になります。

ユーザーまたはグループのクォータが上限に近づくと、GFS2 は、クォータファイルの更新間隔を動的に短縮して上限の超過を防止します。クォータ同期の通常の間隔は、調整可能なパラメーター **quota_quantum** です。表4.2「GFS2 固有のマウントオプション」 で説明しているように、**quota_quantum=** マウントオプションを使用するとデフォルト値の 60 秒を変更することができます。**quota_quantum** パラメーターはファイルシステムがマウントされるたび各ノードに設定しなければなりません。**quota_quantum** に行った変更はアンマウント後は維持されません。**quota_quantum** 値は **mount -o remount** を使用して更新することができます。

GFS2 で実行される自動的な更新の合間にノードのクォータ情報をディスク上のクォータファイルに同期する場合は **gfs2_quota sync** コマンドを使用します。

使用方法

クォータ情報の同期

```
gfs2_quota sync -f MountPoint
```

MountPoint

動作を適用する GFS2 ファイルシステムです。

同期の間隔の調整

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

MountPoint

動作を適用する GFS2 ファイルシステムです。

secs

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

例

以下の例では、コマンドが実行されるノードからファイルシステム **/mygfs2** にクォータ情報を同期します。

```
# gfs2_quota sync -f /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) に変更しています。

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

A.4. クォータ施行を有効/無効にする

GFS2 ファイルシステムでは、クォータ施行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施行を有効にするには **quota=on** オプションを指定してファイルシステムをマウントします。

使用方法

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

クォータ施行を無効にしてファイルシステムをマウントする場合は **quota=off** 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。これがデフォルトの設定です。

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

-o quota={on|off}

ファイルシステムをマウントする際にクォータの施行を有効または無効にします。

BlockDevice

GFS2 ファイルシステムが存在するブロックデバイスを入力します。

MountPoint

GFS2 ファイルシステムのマウント先となるディレクトリーを入力します。

例

以下の例では、**/dev/vg01/lvol0** にある GFS2 ファイルシステムを **/mygfs2** ディレクトリーにマウントし、クォータ施行を有効にしています。

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

A.5. クォータアカウントを有効にする

上限や警告の値は施行せずにディスクの使用状況を追跡して、ユーザーおよびグループのクォータアカウント情報を維持管理することができます。**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。

使用方法

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

-o quota=account

クォータの上限を施行せずユーザーおよびグループの使用状況に関する統計をファイルシステムで維持管理する場合に指定します。

BlockDevice

GFS2 ファイルシステムが存在するブロックデバイスを入力します。

MountPoint

GFS2 ファイルシステムのマウント先となるディレクトリーを入力します。

例

以下の例では、**/dev/vg01/lvol0** にある GFS2 ファイルシステムが **/mygfs2** ディレクトリーにマウントされ、クォータアカウントが有効になります。

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

付録B GFS から GFS2 へのファイルシステム変換

Red Hat Enterprise Linux 6 リリースは GFS ファイルシステムには対応していません。既存の GFS ファイルシステムはすべて **gfs2_convert** コマンドを使って GFS2 ファイルシステムにアップグレードする必要があります。まず先に Red Hat Enterprise Linux 5 システム上で変換手順を行った後、Red Hat Enterprise Linux 6 にアップグレードする必要があるため注意してください。

警告

GFS ファイルシステムを変換する前に必ずバックアップを取っておいてください。変換処理は元に戻すことはできません。また変換中にエラーが発生するとプログラムが突然終了してファイルシステムが使用できなくなる恐れがあります。

GFS ファイルシステムを変換する前に、**gfs_fsck** コマンドを使用してファイルシステムをチェックしエラーを修正しておいてください。

GFS から GFS2 への変換が停電などによる問題で中断された場合は変換ツールを再起動してください。変換が完了するまでは絶対に **fsck.gfs2** コマンドを実行しないでください。

全ファイルシステムまたはほぼ全ファイルシステムを変換する場合は、GFS2 ファイルシステムの全データ構造を格納するのに十分な領域がない可能性があります。このような場合は、空き領域にすべてが収まるよう全ジャーナルのサイズが均一に縮小されます。

B.1. コンテキスト依存のパス名の変換

GFS2 ファイルシステムではコンテキスト依存のパス名 (CDPN) には対応していません (コンテキスト依存のパス名を使用すると可変のファイルやディレクトリーをポイントするシンボリックリンクを作成することが可能)。GFS2 では、**mount** コマンドの **bind** オプションを使用すると同様の動作を行わせることができます。

gfs2_convert コマンドはコンテキスト依存のパス名を特定し、同じ名前の空のディレクトリーに置き換えます。ただし、コンテキスト依存のパス名を bind を使ったマウントで行わせるようにする場合は、対象のコンテキスト依存のパス名を使ってリンクしている先のフルパスが必要となります。ファイルシステムを変換する前に、**find** コマンドを使ってリンクを特定しておきます。

以下のコマンドは、**hostname** のコンテキスト依存のパス名をポイントしているシンボリックリンクを一覧表示します。

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

同様に、他のコンテキスト依存のパス名 (**mach**、**os**、**sys**、**uid**、**gid**, **jid**) にも **find** コマンドを実行します。コンテキスト依存のパス名は **@hostname** または **{hostname}** の形式をとる可能性があるためそれぞれに **find** コマンドを実行する必要がある点に注意してください。

GFS2 ファイルシステムでの複数マウントの結合とコンテキスト依存のパス名についての詳細は 「複数マウントの結合とコンテキスト依存のパス名」 を参照してください。

B.2. GFS から GFS2 への変換手順

次の手順にしたがって、GFS ファイルシステムを GFS2 ファイルシステムに変換します。

1. Red Hat Enterprise Linux システムで既存の GFS ファイルシステムのバックアップを作成します。
2. クラスター内の全ノードから GFS ファイルシステムをアンマウントします。
3. GFS ファイルシステム上で **gfs_fsck** コマンドを実行して、ファイルシステムに破損がないことを確認します。
4. **gfs2_convert *gfsfilesystem*** を実行します。***gfsfilesystem*** を GFS2 に変換する前には、警告と確認のためのプロンプトが表示されます。
5. Red Hat Enterprise Linux 6 にアップグレードします。

以下の例では、ブロックデバイス **/dev/shell_vg/500g** 上の GFS ファイルシステムを GFS2 ファイルシステムに変換します。

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

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

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

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

本付録では glock の **debugfs** インターフェースおよび GFS2 のトレースポイントの両方について説明しています。ファイルシステムの内部について十分に理解しているユーザーで GFS2 の設計および GFS2 固有の問題のデバッグ方法についての詳細をお探しの上級ユーザーを対象としています。

C.1. GFS2 トレースポイントのタイプ

GFS2 トレースポイントには、*glock* ("ジーロック" と発音) トレースポイント、*bmap* トレースポイント、および *log* トレースポイントの 3 つのタイプがあります。これらのトレースポイントは、実行中の GFS2 ファイルシステムのモニタリングに使用することができ、Red Hat Enterprise Linux の旧リリースで対応していたデバッグオプションで取得できる情報に加え更に詳細な情報を得ることができるようになります。ハングやパフォーマンス上の問題など、再現が可能なため問題となっている動作中にトレースポイントの出力が取得できるような場合にはトレースポイントは大変役に立つ情報になります。GFS2 では glocks が主要なキャッシュ制御メカニズムとなるため、GFS2 の主要となる部分のパフォーマンスを理解する場合に重要となります。bmap (block map) トレースポイントを使用するとブロック割り当てやブロックマッピング (ディスク上のメタデータツリー内に既に割り当てられたブロックの検索) を発生と同時にモニタリングし、アクセスの局所性に関連した問題がないかをチェックすることができます。log トレースポイントはジャーナルに書き込まれるデータおよびジャーナルから解放されるデータを追跡するため、GFS2 のその部分に関する有用な情報を得ることができます。

トレースポイントは可能な限り汎用的となるように設計されています。つまり、Red Hat Enterprise Linux 6 の間は API を変更する必要はないはずです。一方、このインターフェースを使用する上で、これが通常の Red Hat Enterprise Linux 6 API セットの一部ではなくデバッグ用のインターフェースであること、そのため GFS2 トレースポイントインターフェースが変更されないという保証はないことを認識しておく必要があります。

トレースポイントは Red Hat Enterprise Linux 6 の汎用機能であり、GFS2 だけでなく広い範囲を対象としています。特に、**blktrace** インフラストラクチャーの実装に使用され、また **blktrace** トレースポイントは GFS2 のトレースポイントとの併用によりシステムパフォーマンスの詳細な状況を把握することができます。トレースポイントが機能するレベルにより、極めて短時間で大量のデータを作成することができます。トレースポイントは有効化された時のシステムへの負荷が最小限となるように設計されていますが、ある程度の影響は避けられません。様々な方法でイベントをフィルタリングすることにより、データ量を削減し、特定の状況を理解するにあたって有用な情報のみの取得にフォーカスすることができます。

C.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。これは、**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としています。**events** サブディレクトリーには、指定可能な全トレーシングイベントが格納されています。また、**gfs2** モジュールがロードされている場合には、**gfs2** サブディレクトリーがあり、GFS2 イベントごとの更なるサブディレクトリーが格納されています。**/sys/kernel/debug/tracing/events/gfs2** の内容は、大体以下のように表示されます。

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

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

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

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

トレースポイントからの出力は、ASCII またはバイナリ形式で提供されます。本付録には、現在バイナリインターフェースについては記載していません。ASCII インターフェースは 2 つの方法で利用することができます。以下のコマンドを実行すると、現在のリングバッファーの内容を一覧表示することができます。

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

このインターフェースは、ある一定の期間に長時間実行されるプロセスを使用する場合や、何らかのイベントの後にバッファー内にキャプチャーされている最新の情報を確認したい場合に役立ちます。もう一つのインターフェースは **/sys/kernel/debug/tracing/trace_pipe** で、前出力が必要な場合に使用することができます。イベントは、発生すると同時にこのファイルから読み取ることができます。このインターフェースでは履歴情報は提供されません。出力の形式は両インターフェースとも同じで、本付録の後半で GFS2 イベント別に説明しています。

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

C.3. GLOCKS

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

表C.1 glock モードおよび DLM ロックモード

glock モード	DLM ロックモード	注記
UN	IV/NL	ロック解除されています (I フラグに依存する glock または NL ロックに関連付けされた DLM ロックはなし)
SH	PR	共有 (読み取り保護) ロック
EX	EX	排他ロック

glock モード	DLM ロックモード	注記
DF	CW	ダイレクト I/O およびファイルシステムフリーズで使用される遅延 (同時書き込み)

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

注記

このような DLM ロックの振る舞いの特定の側面は、Red Hat Enterprise Linux 5 から変更されています。Red Hat Enterprise Linux 5 では、glock にアタッチされた DLM ロックが完全に解除されてしまう場合があるため、必要な場合に LVB (Lock Value Block) が保持されるようにするための異なるメカニズムを採用しています。Red Hat Enterprise Linux 6 の新たなスキームは **`lock_dlm`** ロックモジュールを GFS2 にマージ (DLM 自体と混同されないように) することによって可能となりました。

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

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

注記

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

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

表C.2 glock のモードとデータタイプ

glock モード	キャッシュデータ	キャッシュメタデータ	ダーティーデータ	ダーティーメタデータ
UN	No	No	No	No
SH	Yes	Yes	No	No

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
DF	No	Yes	No	No
EX	Yes	Yes	Yes	Yes

C.4. GLOCK DEBUGFS インターフェース

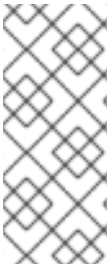
glock **debugfs** インターフェースは glock とホルダーの内部の状態を視覚化することができます。また、場合によっては、ロックされているオブジェクトのサマリー情報が含まれます。このファイルの各行は、インデントなしで G: (glock 自体のことを示す) から始まるか、1 文字分字下げされて他の文字で始まり、ファイル内の直前の行の glock に関連付けられた構造を示します (H: ホルダー、I: inode、R: リソースグループ)。以下は、このファイルの内容の一例です。

```
G:  s:SH n:5/75320 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:EX n:3/258028 f:yI t:EX d:EX/0 a:3 r:4
 H: s:EX f:tH e:0 p:4466 [postmark] gfs2_inplace_reserve_i+0x177/0x780
[gfs2]
 R: n:258028 f:05 b:22256/22256 i:16800
G:  s:EX n:2/219916 f:yfI t:EX d:EX/0 a:0 r:3
 I: n:75661/219916 t:8 f:0x10 d:0x00000000 s:7522/7522
G:  s:SH n:5/127205 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:EX n:2/50382 f:yfI t:EX d:EX/0 a:0 r:2
G:  s:SH n:5/302519 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/313874 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/271916 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/312732 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
```

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

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

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



注記

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

ホルダーと glock 用のフラグの完全な一覧は 表C.4「glock のフラグ」 および 表C.5「Glock holder flags」 にまとめています。LVB の内容は現在 **debugfs** インターフェースで入手することはできません。

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

表C.3 glock のタイプ

タイプ番号	ロックタイプ	用途
1	trans	トランザクションのロック
2	inode	Inode のメタデータとデータ
3	rgrp	リソースグループのメタデータ
4	meta	スーパーブロック
5	iopen	最後にinode をクローズしたプロセスの検出
6	flock	**flock**(2) syscall
8	quota	クォータの操作
9	journal	ジャーナルミューテックス

重要な glock フラグの一つに l (locked) フラグがあります。glock の状態変更を実行する際に glock 状態へのアクセスを回避するために使用するビットロックです。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が間もなく送信する時に設定され、完全な操作が実行された後のみに消去されます。場合によっては、複数のロック要求が送信されて、その間に様々な無効化が発生していることを意味します。

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

表C.4 glock のフラグ

フラグ	名前	意味
d	Pending demote	遅延された (リモートの) 降格要求
D	Demote	降格要求 (ローカルまたはリモート)

フラグ	名前	意味
f	Log flush	この glock を解放する前にログのコミットが必要
F	Frozen	リモートのノードからの返信を無視 - 復元の進行中
i	Invalidate in progress	この glock のページの無効化が進行中
I	Initial	DLM がこの glock と関連付けられるとセットされる
l	Locked	glock は状態を変更中
L	LRU	LRU 一覧に glock が記載されるとセットされる
o	オブジェクト	glock がオブジェクトに関連付けられるとセットされる (type 2 glock の場合 inode、type 3 glock の場合リソースグループ)
p	Demote in progress	glock は降格要求に応答中
q	キュー待ち	glock に対してホルダーがキュー待ちになるとセットされる、glock が行われキュー待ちのホルダーがなくなると消去される、(glock 維持の最小時間を計算するアルゴリズムの一部として使用されます。)
r	Reply pending	リモートノードから受信した返信の処理の待機中
y	Dirty	この glock を解放する前にディスクへのデータのフラッシュが必要

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

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

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

C.5. GLOCK ホルダー

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

表C.5 Glock holder flags

フラグ	名前	意味
a	Async	glock の結果を待たない (結果を後でポールします)
A	Any	互換性のあるロックモードはすべて受け入れ可能
c	No cache	ロック解除時に DLM ロックを即時降格
e	No expire	後続のロック取り消し要求を無視
E	Exact	完全一致するロックモード
F	First	このロックにホルダーを最初に付与する必要がある場合にセットされる
H	Holder	要求したロックが付与されたことを示す
p	Priority	キューの先頭にある待機ホルダー
t	Try	「try」ロック
T	Try 1CB	コールバックを送信する「try」ロック
W	Wait	要求完了の待機中にセットされる

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

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

glock のサブシステムは、2 種類の "try" ロックをサポートしています。これらのロックは、(適切なバックオフとリエントリで) 通常の規則に反したロックの取得を可能にするのに加えて、他のノードが使用中のリソースを回避するのに役立つので、いずれも有用です。通常の t (try) ロックは、基本的にはその名前の示すとおりの "try" ロックで、特別なことは何もしません。一方、T (**`Try 1CB`**) ロックは、DLM が単一のコールバックを互換性のないロックホルダーに送信する点以外は t ロックと同じです。T (**`Try 1CB`**) ロックの用途の一つに、**`iopen`** ロックとの併用があります。このロックは、inode の **`i_nlink`** カウントがゼロの時にノードを判別して、どのノードが割り当て解除の原因となっているかを判断するのに使用します。**`iopen`** glock は通常共有の状態で保持されますが、**`i_nlink`** カウントがゼロになって **`->delete_inode`**() が呼び出されると、排他的ロックを T (**`try 1CB`**) に設定するように要求します。ロックが許可された場合には、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には、ロックの許可を阻止しているノードで glock に D (Demote) フラグが付きます。このフラグは、割り当て解除が忘れられないように **`->drop_inode`**() の時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。

C.6. GLOCK のトレースポイント

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

gfs2_glock_state_change トレースポイントは、最も重要な理解すべきトレースポイントです。このトレースポイントは初期作成から、**gfs2_glock_put** で終わる最終降格、最終の NL からロック解除に至るまでの遷移の全状態をトラッキングします。l (locked) glock フラグは、状態の変更が発生する前に必ず設定され、完了するまではクリアされません。状態の変更中には、許可されるホルダーはありません (H glock ホルダーのフラグ)。キューに配置されたホルダーがある場合には、常に W (waiting) 状態となります。状態の変更が完了すると、ホルダーが許可されることが可能となります。これは、l glock フラグがクリアされる前の最後の操作です。

gfs2_demote_rq トレースポイントは、ローカルおよびリモートの両方の降格要求を追跡します。ノードに十分なメモリーがあるとすれば、ローカルの降格要求が発生するのはまれです。降格要求はほとんどの場合 umount や時折のメモリー再利用によって作成されます。リモートの降格要求数はノード間における特定の inode やリソースグループに対する競合の一つの尺度です。

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

C.7. BMAP トレースポイント

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

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

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

C.8. LOG トレースポイント

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

役立ちます。

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

gfs2_ail_flush トレースポイント (Red Hat Enterprise Linux 6.2 以降) は、AIL リストのフラッシュの開始と終了を追跡する **gfs2_log_flush** トレースポイントと似ています。AIL リストには、ログを通過済みであるが所定の場所に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追加のログ領域を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。

C.9. GLOCK の統計

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

GFS2 では 2 種類のカウンターを維持しています。

- **dcount**、要求された DLM 動作の回数をカウントします。平均／分散の計算に使用されたデータ量を示します。
- **qcount**、要求された **syscall** レベルの動作回数をカウントします。一般的には **qcount** は **dcount** と同数かそれ以上になります。

また、GFS2 では 3 種類の平均／分散の組み合わせを維持します。平均／分散の組み合わせは平滑化指数関数推定値であり、使用されるアルゴリズムはネットワークコードで往復時間の計算に使用されるアルゴリズムになります。GFS2 で維持される平均と分散の組み合わせは整数のナノ秒単位で表されます。

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

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

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

統計は 2 種類の **sysfs** ファイルに格納されます。

- **glstats** ファイル、1 glock に付き 1 行の統計値が格納される点以外は **glocks** ファイルと同じです。作成される glock の glock タイプの「per cpu」データでデータの初期化が行われます (ゼロとなるカウンターとは別)。非常に大きなサイズのファイルになる可能性があります。
- **lkstats** ファイル、各 glock タイプの「per cpu」統計値が含まれます。1 統計に 1 行、コラムごと 1 cpu コアを示します。glock タイプごと 8 行あります。

C.10. 参考文献

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

- 本付録に記載した情報の一部は、2009 年の Linux シンポジウム で Steve Whitehouse 氏が発表した論文の内容を一部取り入れています。この情報は http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/filesystems/gfs2-glocks.txt;h=0494f78d87e40c225eb1dc1a1489acd891210761;hb=HEAD で参照してください。

- glock のライブロックルールに関する詳細は http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/filesystems/gfs2-glocks.txt;h=0494f78d87e40c225eb1dc1a1489acd891210761;hb=HEAD を参照してください。

- イベントトレーシングに関する詳細は http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/trace/events.txt;h=09bd8e9029892e4e1d48078de4d076e24eff3dd2 を参照してください。

- **trace-cmd** ユーティリティに関する詳細は http://lwn.net/Articles/341902/ を参照してください。

付録D 改訂履歴

改訂 7.1-3.2 **Wed Apr 1 2015** **Noriko Mizumoto**
翻訳、査読、修正完了

改訂 7.1-3.1 **Wed Apr 1 2015** **Noriko Mizumoto**
翻訳ファイルを XML ソースバージョン 7.1-3 と同期

改訂 7.1-3 **Tue Dec 16 2014** **Steven Levine**
RHEL 6 スプラッシュページを更新し、sort_order を実装

改訂 7.0-9 **Wed Oct 8 2014** **Steven Levine**
6.6 GA リリースバージョン

改訂 7.0-8 **Thu Aug 7 2014** **Steven Levine**
6.6 Beta リリースバージョン

改訂 7.0-4 **Thu Jul 17 2014** **Steven Levine**
バグを修正: #1102591
Pacemaker クラスターでの GFS2 の設定手順を追加

改訂 7.0-3 **Wed Jul 16 2014** **Steven Levine**
バグを修正: #1035119
Glock フラグ表の更新、Glock 統計に関するセクションを追記

改訂 7.0-1 **Thu Jun 5 2014** **Steven Levine**
6.6 リリース初回ドラフト版

改訂 6.0-6 **Wed Nov 13 2013** **Steven Levine**
6.5 GA リリース向けのバージョン

改訂 6.0-5 **Fri Sep 27 2013** **Steven Levine**
6.5 ベータリリース向けのバージョン

改訂 6.0-3 **Fri Sep 27 2013** **Steven Levine**
バグを修正: #960841
GFS2 ファイルシステムの場合には SELinux に対応しない点を明記

改訂 6.0-1 **Fri Sep 06 2013** **Steven Levine**
Samba および GFS2 に関する注意事項を追記

改訂 5.0-7 **Mon Feb 18 2013** **Steven Levine**
6.4 GA リリース向けバージョン

改訂 5.0-5 **Mon Nov 26 2012** **Steven Levine**
6.4 ベータリリース向けのバージョン

改訂 5.0-4 **Tue Nov 13 2012** **Steven Levine**
バグを修正: #860324
GFS2 設定と操作における考慮点の章を更新、明確化

バグを修正: #807057
導入前に Red Hat 認定の担当者と構成の検証を行うことを推奨する記載を追加

改訂 5.0-1 **Mon Oct 15 2012** **Steven Levine**
操作における考慮点を変更

改訂 4.0-2 **Thu Mar 28 2012** **Steven Levine**
6.3 GA リリース向けバージョン

改訂 4.0-1 **Thu Mar 28 2012** **Steven Levine**
バグを修正: #782482、#663944
GFS2 の設定および操作の考慮事項についての新たな章を追加

バグを修正: #757742
GFS2 を CLVM で使用する必要性を明記

バグを修正: #786621
マイナーな誤字、脱字を修正

改訂 3.0-2 **Thu Dec 1 2011** **Steven Levine**
Red Hat Enterprise Linux 6.2 の GA リリース

改訂 3.0-1 **Mon Sep 19 2011** **Steven Levine**
Red Hat Enterprise Linux 6.2 Beta リリースに向けた初回改訂

バグを修正: #704179
tunegfs2 コマンドの対応について記載

バグを修正: #712390
GFS2 トレースポイントについての新たな付録を追加

バグを修正: #705961
マイナーな誤字・脱字を修正

改訂 2.0-1 **Thu May 19 2011** **Steven Levine**
Red Hat Enterprise Linux 6.1 の初期リリース

バグを修正: #549838
Red Hat Enterprise Linux 6.1 での Linux 標準のクォータ機能の対応について記載

バグを修正: #608750
GFS2 の withdraw 関数についての説明を明記

バグを修正: #660364
GFS2 ファイルシステムの最大サイズに関する訂正

バグを修正: #687874
GFS2 トラブルシューティングに関する新しい章を追加

バグを修正: #664848
GFS から GFS2 に変換する前にコンテキスト依存のパス名を検索する方法についての記載を追加

改訂 1.0-1 **Wed Nov 15 2010** **Steven Levine**
Red Hat Enterprise Linux 6 初期リリース

索引

シンボル

A

D

F

G

M

P