Red Hat Enterprise Linux 8

GFS2 ファイルシステムの設定

高可用性クラスターでの GFS2 ファイルシステムの計画、管理、トラブルシューティング、および設定

Last Updated: 2023-09-19

Red Hat Enterprise Linux 8 GFS2 ファイルシステムの設定

高可用性クラスターでの GFS2 ファイルシステムの計画、管理、トラブルシューティング、および設定

法律上の通知

概要

Red Hat Enterprise Linux (RHEL) Resilient Storage Add-Onは、共通のブロックデバイスを共有する複数のノード間の一貫性を管理するクラスターファイルシステムである Red Hat Global File System 2 (GFS2) を提供します。本書では、GFS2 ファイルシステムのデプロイメントの計画に関する情報と、GFS2 ファイルシステムの設定、トラブルシューティング、およびチューニングの手順について説明します。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ を参照してください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

特定の文章に関するコメントの送信

1. **Multi-page HTML** 形式でドキュメントを表示し、ページが完全にロードされてから右上隅に **Feedback** ボタンが表示されていることを確認します。
2. カーソルを使用して、コメントを追加するテキスト部分を強調表示します。
3. 強調表示されたテキストの近くに表示される **Add Feedback** ボタンをクリックします。
4. フィードバックを追加し、**Submit** をクリックします。

Jira からのフィードバック送信（アカウントが必要）

1. Jira の Web サイト にログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 GFS2 ファイルシステムのデプロイメントの計画

Red Hat Global File System 2 (GFS2) ファイルシステムは、64 ビットの対称クラスターファイルシステムで、共有名前空間を提供し、一般的なブロックデバイスを共有する複数のノード間の一貫性を管理します。GFS2 ファイルシステムは、ローカルファイルシステムに可能な限り近い機能セットを提供すると同時に、ノード間でクラスターの完全な整合性を強制することを目的としています。これを実現するため、ノードはファイルシステムリソースにクラスター全体のロックスキームを使用します。このロックスキームは、TCP/IP などの通信プロトコルを使用して、ロック情報を交換します。

場合によっては、Linux ファイルシステム API では、GFS2 のクラスター化された性質を完全に透過的にすることができません。たとえば、GFS2 で POSIX ロックを使用しているプログラムは、**GETLK** の使用を回避する必要があります。なぜなら、クラスター環境では、プロセス ID が、クラスター内の別のノードに対するものである可能性があるためです。ただし、ほとんどの場合、GFS2 ファイルシステムの機能は、ローカルファイルシステムのものと同じです。

Red Hat Enterprise Linux (RHEL) Resilient Storage Add-On は GFS2 を提供します。GFS2 が必要とするクラスター管理の提供は RHEL High Availability Add-On により提供されます。

gfs2.ko カーネルモジュールは GFS2 ファイルシステムを実装し、GFS2 クラスターノードに読み込まれます。

GFS2 環境を最大限に利用するためにも、基礎となる設計に起因するパフォーマンス事情を考慮することが重要です。GFS2 では、ローカルファイルシステムと同様、ページキャッシュで、頻繁に使用されるデータのローカルキャッシングを行ってパフォーマンスを向上します。クラスターのノード間で一貫性を維持するために、**glock** ステートマシンでキャッシュ制御が提供されます。

重要

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

1.1. 確認する主な GFS2 パラメーター

GFS2 ファイルシステムのインストールおよび設定前に計画する必要がある主要な GFS2 パラメーターが複数あります。

GFS2 ノード

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

ファイルシステムの数

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

ファイルシステム名

各 GFS2 ファイルシステムには一意の名前を付ける必要があります。この名前は通常、LVM 論理ボリューム名と同じで、GFS2 ファイルシステムがマウントされたときに DLM ロックテーブル名として使用されます。たとえば、このガイドでの手順では、ファイルシステム名に **mydata1** および **mydata2** を使用します。

ジャーナル

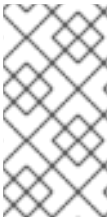
GFS2 ファイルシステムのジャーナル数を決定します。GFS2 では、ファイルシステムのマウントを必要とするクラスターの各ノードにジャーナルが必要になります。たとえば、16 ノードのクラスターがあり、2 つのノードからファイルシステムのみをマウントする必要がある場合は、2 つのジャーナルのみが必要になります。GFS2 では、追加サーバーがファイルシステムをマウントするため、後で **gfs2_jadd** ユーティリティーを使用して、ジャーナルを動的に追加できます。

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 (**lvmlockd** を使用して) 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。

時間プロトコル

GFS2 ノードのクロックが同期されていることを確認します。Red Hat Enterprise Linux ディストリビューションで提供されている Precision Time Protocol (PTP)、または Network Time Protocol (NTP) ソフトウェア (設定に必要な場合) を使用することが推奨されます。
不要な inode 時間スタンプの更新を防ぐには、GFS2 ノード内のシステムクロックの時間差が数分以内になるように設定する必要があります。inode のタイムスタンプの更新を不要に行うと、クラスターのパフォーマンスに大きな影響が及びます。



注記

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

1.2. GFS2 サポートに関する考慮事項

GFS2 ファイルシステムを実行するクラスターで Red Hat からのサポートを受けるには、GFS2 ファイルシステムのサポートポリシーを考慮する必要があります。

1.2.1. ファイルシステムおよびクラスターの最大サイズ

次の表で、GFS2 が現在対応しているファイルシステムの最大サイズと最大ノード数をまとめています。

表1.1 GFS2 サポート制限

パラメーター	最大
ノードの数	16 (x86、PowerVM の Power8) 4 (z/VM の s390x)
ファイルシステムのサイズ	すべての対応アーキテクチャー上の 100TB

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

ファイルシステムのサイズを決定する際に、復旧のニーズを考慮する必要があります。大規模なファイルシステムで **fsck.gfs2** コマンドを実行すると、時間がかかって大量のメモリーを消費する可能性があります。また、ディスクまたはディスクサブシステムに障害が発生した場合、復旧時間はバックアップメディアの速度により異なります。fsck.gfs2 コマンドが必要とするメモリー量の詳細は、**Determining required memory for running fsck.gfs2** を参照してください。

1.2.2. クラスターの最小サイズ

GFS2 ファイルシステムはスタンドアロンシステムで実装したり、クラスター設定の一部として実装したりできますが、以下を除いて、Red Hat はシングルノードファイルシステムとしての GFS2 の使用をサポートしていません。

- Red Hat は、たとえばバックアップの目的で、必要に応じてクラスターファイルシステムのスナップショットをマウントする、単一ノードの GFS2 ファイルシステムを引き続きサポートします。

- GFS2 ファイルシステム (DLM を使用) をマウントする単一ノードクラスターは、セカンダリーサイトのディザスターリカバリー (DR) ノードの目的でサポートされています。この例外は DR のみを目的としており、メインクラスターのワークロードをセカンダリーサイトに転送するためのものではありません。
  たとえば、プライマリーサイトがオフラインの場合に、セカンダリーサイトにマウントされたファイルシステムからデータをコピーします。ただし、プライマリーサイトから単一ノードクラスターのセカンダリーサイトに直接ワークロードを移行することはサポートされていません。完全なワーク負荷を単一ノードのセカンダリーサイトに移行する必要がある場合は、セカンダリーサイトのサイズがプライマリーサイトと同じである必要があります。

  Red Hat では、シングルノードクラスターに GFS2 ファイルシステムをマウントするときは、**errors=panic** マウントオプションを指定して、ファイルシステムのエラーが発生した場合に単一ノードのクラスターが自身でフェンシングできないようにするため、GFS2 が撤退するときにパニックを起こさせます。

Red Hat は、シングルノード向けに最適化され、一般的にクラスターファイルシステムよりもオーバーヘッドが小さい多くの高パフォーマンスのシングルノードファイルシステムをサポートします。また、Red Hat は、シングルノードがファイルシステムをマウントする必要がある場合に限り、GFS2 ではなく高パフォーマンスのシングルノードファイルシステムを使用することが推奨されます。Red Hat Enterprise Linux 9 がサポートするファイルシステムの詳細は、ファイルシステムの管理 を参照してください。

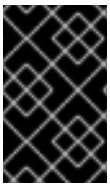
1.2.3. 共有ストレージに関する留意事項

GFS2 ファイルシステムは LVM 以外で使用できますが、Red Hat は、共有 LVM 論理ボリュームに作成されている GFS2 ファイルシステムのみをサポートします。

GFS2 ファイルシステムをクラスターファイルシステムとして設定する場合は、クラスター内のすべてのノードが共有ストレージにアクセスできることを確認する必要があります。共有ストレージにアクセスできるノードとサポート対象外のノードがある非対称クラスター設定はサポートされません。ただし、すべてのノードが実際に GFS2 ファイルシステム自体をマウントする必要はありません。

1.3. GFS2 フォーマットに関する考慮事項

GFS2 ファイルシステムを保存してパフォーマンスを最適化するには、以下のような推奨事項を考慮する必要があります。



重要

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

ファイルシステムサイズ: 小さい方が好ましい
GFS2 は、理論的には 8 EB のファイルシステムに対応できる 64 ビットアーキテクチャーに基づいています。ただし、64 ビットハードウェア用で現在対応している GFS2 ファイルシステムの最大サイズは 100 TB です。

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

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

- 各ファイルシステムのバックアップ所要時間が短縮されます。
- **fsck.gfs2** コマンドを使用してファイルシステムをチェックする必要がある場合は、それほど時間がかかりません。
- **fsck.gfs2** コマンドを使用してファイルシステムを確認する必要がある場合は、必要になるメモリーが少なくなります。

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

当然ながら、GFS2 ファイルシステムのサイズを小さくしすぎると、容量が不足し、それ自体に影響が及ぶ可能性があります。サイズを決定する前に、どのように使用されるかを考慮してください。

ブロックサイズ: デフォルト (4K) ブロックを推奨

mkfs.gfs2 コマンドは、デバイストポロジーに基づいて最適なブロックサイズを推定しようとします。通常、4K ブロックがブロックサイズとして推奨されます。これは、4K が Red Hat Enterprise Linux のデフォルトページサイズ (メモリー) であるためです。その他の一部のファイルシステムとは異なり、GFS2 は 4K カーネルバッファーを使用して多くの操作を実行します。ブロックサイズが 4K であれば、カーネルによるバッファー操作の作業数が減ります。

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

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と、ジャーナルのサイズを指定できます。サイズを指定しないと、デフォルトの 128 MB になります。これは、ほとんどのアプリケーションに最適です。

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

通常、デフォルトのジャーナルサイズである 128 MB を使用することが推奨されます。ファイルシステムが非常に小さい (例: 5GB) 場合、128 MB のジャーナルでは非現実的である可能性があります。より大きなファイルシステムがあり、容量に余裕があれば、256 MB のジャーナルを使用するとパフォーマンスが向上する可能性があります。

リソースグループのサイズおよび数

mkfs.gfs2 コマンドで GFS2 ファイルシステムを作成すると、ストレージが、リソースグループと呼ばれる均一なスライスに分割されます。最適なリソースグループサイズ (32MB から 2GB) の推定値を計算しようとします。**mkfs.gfs2** コマンドに **-r** オプションを指定すると、デフォルトを上書きできます。

最適なリソースグループのサイズは、ファイルシステムの使用方法によって異なります。どの程度いっぱいになるか、または著しく断片化されるかどうかを考慮してください。

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

ファイルシステムに含まれるリソースグループの数が多過ぎて、各リソースグループが小さすぎると、ブロックの割り当てにより、空きブロックに対する何万ものリソースグループの検索に時間がかかりすぎることがあります。ファイルシステムが満杯になるほど、検索されるリソースグループが増え、そのすべてにクラスター全体のロックが必要になります。その結果、パフォーマンスが低下します。

ただし、ファイルシステムのリソースグループの数が少なすぎて、各リソースグループが大き過ぎると、同じリソースグループロックへのブロックの割り当てが頻繁に競合する可能性があり、パフォーマンスにも影響が及びます。たとえば、2 GB の 5 つのリソースに分けられた 10GB のファイルがある場合、クラスター内のノードは、同じファイルシステムが 32 MB の 320 個のリソースグループに分けられている場合よりも頻繁に 5 つのリソースグループを奪おうとします。空きブロックを持つリソースグループを見つける前に、各ブロックの割り当てが複数のリソースグループを参照する必要があるため、ファイルシステムがほぼ満杯になると、この問題は悪化します。GFS2 は、次のいずれかの方法でこの問題を軽減しようとします。

- まず、リソースグループが完全にいっぱいになると、ブロックが解放されるまで、そのリソースグループが記憶され、今後の割り当てで確認が行われなくなります。ファイルを削除しなければ、競合は少なくなります。ただし、アプリケーションがブロックを継続的に削除し、ほぼ満杯のファイルシステムに新しいブロックを割り当てる場合は、競合が非常に高くなり、パフォーマンスに深刻な影響を及ぼします。

- 次に、既存ファイルに新しいブロックが追加されると、GFS2 は、(たとえば追加することで) そのファイルと同じリソースグループ内で、新しいブロックのグループ化を試みます。これは、パフォーマンスを向上させるために行われます。回転ディスクでは、物理的に相互に近い場合に、シーク操作の所要時間が短くなります。

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

1.4. クラスター内の GFS2 に関する考慮事項

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

クラスターファイルシステムをデプロイすることは、単一ノードデプロイメントのドロップインには代わりません。Red Hat では、システムをテストして必要なパフォーマンスレベルで機能させるためにも、新規インストールにおいて約 8 〜12 週のテストを行うことを推奨しています。この期間中は、パフォーマンスまたは機能の問題を処理できます。また、すべてのクエリーは Red Hat サポートチームに転送する必要があります。

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

1.5. ハードウェアに関する検討事項

GFS2 ファイルシステムをデプロイする場合に、ハードウェアについて以下の考慮事項を考慮してください。

- より高品質なストレージオプションを使用する
  GFS2 は、iSCSI や FCoE (Fibre Channel over Ethernet) などの安価な共有ストレージオプションで動作しますが、キャッシュ容量が大きい高品質のストレージを購入するとパフォーマンスが向上します。Red Hat は、ファイバーチャネルの相互接続の SAN ストレージで、品質、健全性、パフォーマンスに関する多くのテストを行っています。原則として、最初にテストしたものは常にデプロイすることが推奨されます。

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

第2章 GFS2 使用に関する推奨事項

GFS2 ファイルシステムをデプロイする場合、さまざまな一般的な推奨事項を考慮する必要があります。

2.1. ATIME 更新の設定

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

- **ctime** - inode のステータスが最後に変更した時刻
- **mtime** - ファイル (またはディレクトリー) のデータが最後に変更した時刻
- **atime** - ファイル (またはディレクトリー) のデータに最後にアクセスした時刻

デフォルトで GFS2 などの Linux ファイルシステムにあるため、**atime** 更新が有効な場合は、ファイルが読み込まれるたびにその inode を更新する必要があります。

ほとんどのアプリケーションは、**atime** により提供される情報を使用しないため、このような更新では、不要な書き込みのトラフィックとファイルロックのトラフィックが大量に必要になる可能性があります。そのトラフィックにより、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、電源を切るか、**atime** 更新頻度を減らすことが推奨されます。

atime 更新の影響を軽減するには、以下の方法を利用できます。

- **relatime** (relative atime) でマウントします。これは、以前の **atime** 更新が **mtime** または **ctime** の更新よりも古い場合に **atime** を更新します。これは、GFS2 ファイルシステムのデフォルトのマウントオプションです。
- **noatime** または **nodiratime** でマウントします。**noatime** でのマウントは、そのファイルシステムのファイルやディレクトリーの両方の **atime** 更新を無効にします。**nodiratime** でのマウントは、そのファイルシステムのディレクトリーに対してのみ **atime** 更新を無効にします。一般的に、**noatime** マウントオプションまたは **nodiratime** マウントオプションを指定して GFS2 ファイルシステムをマウントすることが推奨されています。この場合は、アプリケーションがこれを許可する **noatime** が優先されます。GFS2 ファイルシステムパフォーマンスにおけるこのような引数の効果は、GFS2 Node Locking を参照してください。

次のコマンドを使用して、**noatime** Linux マウントオプションを指定して GFS2 ファイルシステムをマウントします。

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

BlockDevice
GFS2 ファイルシステムを置くブロックデバイスを指定します。

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

この例では、GFS2 ファイルシステムは **/dev/vg01/lvol0** に置かれ、**atime** 更新がオフの状態で **/mygfs2** にマウントされます。

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

2.2. VFS チューニングオプション: リサーチと実験

すべての Linux ファイルシステムと同様に、GFS2 は仮想ファイルシステム (VFS) と呼ばれるレイヤーの上にあります。VFS は多くのワークロードに対するキャッシュ設定に適したデフォルトを提供しますが、ほとんどの場合で変更する必要はありません。ただし、効率的に実行していないワークロードがある場合 (たとえば、キャッシュが大きすぎる、あるいは小すぎる場合など) は、**sysctl**(8) コマンドを使用して **/proc/sys/vm** ディレクトリーの **sysctl** ファイルの値を調整することでパフォーマンスを向上できます。ファイルのドキュメンテーションは、カーネルソースツリーの **Document/sysctl/vm.txt** で参照できます。

たとえば、**dirty_background_ratio** および **vfs_cache_pressure** の値は、状況に応じて調整できます。現在の値を取得するには、次のコマンドを使用します。

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

次のコマンドは、値を調整します。

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

/etc/sysctl.conf ファイルを編集すると、このパラメーター値を永久的に変更できます。

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

2.3. GFS2 での SELINUX

GFS2 で Security Enhanced Linux (SELinux) を使用すると、パフォーマンスが多少低下します。これを回避するには、強制モードで SELinux を動作させているシステム上であっても、GFS2 で SELinux を使用するべきではありません。GFS2 ファイルシステムをマウントする場合は、man ページの **mount**(8) で説明されているように **context** オプションのいずれかを使用して SELinux が各ファイルシステムオブジェクトの **seclabel** 要素の読み取りを試行しないようにしてください。SELinux は、ファイルシステムのすべての内容が、**context** マウントオプションによる **seclabel** 要素でラベル付けされると想定します。また、これにより、**seclabel** 要素を含む拡張属性ブロックの別のディスク読み取りが回避され、処理が高速化されます。

たとえば、SELinux が強制モードになっているシステムで、ファイルシステムに Apache コンテンツが含まれる場合は、次の **mount** コマンドを使用して GFS2 ファイルシステムをマウントできます。このラベルはファイルシステム全体に適用されます。メモリー内に残り、ディスクには書き込まれません。

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

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

Pacemaker クラスターでは、GFS2 ファイルシステムの管理には常に Pacemaker を使用する必要があります。GFS2 ファイルシステムリソースを作成する際にマウントオプションを指定できます。

2.4. GFS2 での NFS の設定

GFS2 ロッキングサブシステムとそのクラスターの複雑性を考慮し、GFS2 での NFS の設定には多くの予防措置が必要になります。

警告

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

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

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

- Red Hat は、以下のような特性を持つアクティブ/パッシブ設定のロックを備えた NFSv3 を使用する Red Hat High Availability Add-On 設定のみをサポートしています。この設定では、ファイルシステムで高可用性 (HA) が実現し、システムの停止時間が短縮されます。これは、失敗したノードが、あるノードから別のノードへの NFS サーバーに障害が発生したときに **fsck** コマンドの実行が必須にならないためです。
    - バックエンドファイルシステムは、2〜16 のノードクラスターで稼働している GFS2 ファイルシステムです。
    - NFSv3 サーバーは、単一クラスターノードから GFS2 ファイルシステム全体を一度にエクスポートするサービスとして定義されます。
    - NFS サーバーは、1 つのクラスターノードから他のクラスターノードへのフェイルオーバーが可能です (アクティブ/パッシブ設定)。
    - NFS サーバー経由を **除いて**、GFS2 ファイルシステムへのアクセスは許可されていません。これには、ローカルの GFS2 ファイルシステムアクセスと、Samba または クラスター

化 Samba によるアクセスの両方が含まれます。マウント元のクラスターノードからローカルにファイルシステムにアクセスすると、データが破損する可能性があります。

    - システムで NFS クォータはサポートされていません。

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

2.5. GFS2 での SAMBA (SMB または WINDOWS) ファイルサービス

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

Samba の外部から Samba 共有のデータへの同時アクセスには対応していません。現在、GFS2 クラスターリースはサポート外で、Samba ファイル処理の速度が低下します。Samba のサポートポリシーの詳細は、RHEL Resilient Storage のサポートポリシー - ctdb の一般的なポリシー および RHEL Resilient Storage のサポートポリシー - 他のプロトコルを介した gfs2 コンテンツのエクスポート を参照してください。

2.6. GFS2 用仮想マシンの設定

仮想マシンで GFS2 ファイルシステムを使用する場合は、キャッシュを強制的にオフにするために各ノードの仮想マシンのストレージ設定を適切に設定することが重要です。たとえば、**libvirt** ドメインで **cache** および **io** の設定を追加すると、GFS2 が期待通りに動作するようになります。

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

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

2.7. ブロック割り当て

通常、データの書き込みのみを行うアプリケーションでも、ブロックの割り当て方法や割り当て場所は重要ではありません。ただし、ブロック割り当ての仕組みに関するある程度の知識は、パフォーマンスの最適化に役立ちます。

2.7.1. ファイルシステムで空き領域の確保

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

こういった理由で、(ワークロードにより数値は変わってきますが) 85% 以上が使用済みのファイルシステムを実行しないことが推奨されます。

2.7.2. 可能な場合は各ノードで独自のファイルを割り当て

GFS2 ファイルシステムで使用するアプリケーションを開発する場合は、可能であれば独自のファイルを各ノードに割り当てることが推奨されます。全ファイルが 1 つのノードにより割り当てられ、その他のノードがこれらのファイルにブロックを追加する必要がある場合には、分散ロックマネージャー (DLM) の動作の仕組みが原因となって、ロック競合が多くなります。

これまでロックマスターという用語を使用して、クラスター内のリモートノードまたはローカルから送信されるロック要求の現在のコーディネーターノードを表していました。ロック要求コーディネーターというこの用語は、ロック要求のキューへの追加、許可または拒否の面からすると実際にはリソース (DLM 用語) を指すので若干誤解を招きます。DLM はピアツーピアシステムであるため、DLM で使用する用語では、リーダーという意味で使用されるべきです。

Linux カーネルの DLM 実装では、ロックを最初に使用するノードが、ロック要求のコーディネーターになり、その時点から、コーディネーターは変更されません。これは、Linux カーネル DLM の実装の説明で、一般的に DLM のプロパティーではありません。今後の更新により、特定のロック要求を連携することでノード間の移行が可能になります。

ロック要求が連携される場所は、ロック要求のイニシエーターに対して透過的です。ただし、要求のレイテンシーからの影響がある場合を除きます。現在の実装で考えられる 1 つの結果として、(I/O コマンドを実行する他のノードよりも前に、1 つのノードがファイルシステム全体をスキャンする場合など) 最初のワークロードでバランスが取れていない場合に、ファイルシステムを最初にスキャンしたノードと比べると、クラスター内の他のノードのロックでレイテンシーが発生する可能性が高くなります。

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

2.7.3. 可能な場合には事前に割り当て

ファイルを事前に割り当てた場合は、ブロックの割り当てを完全に回避し、ファイルシステムをより効率的に実行できます。GFS2 には、**fallocate(1)** システムコールが含まれます。これは、データブロックの割り当てに使用できます。

第3章 GFS2 ファイルシステムの管理

GFS2 ファイルシステムの作成、マウント、拡張、および管理に使用するコマンドやオプションには様々なものがあります。

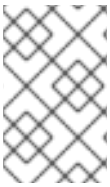
3.1. GFS2 ファイルシステムの作成

mkfs.gfs2 コマンドで、GFS2 ファイルシステムを作成できます。ファイルシステムは、アクティブ化された LVM ボリュームに作成されます。

3.1.1. GFS2 mkfs コマンド

mkfs.gfs2 コマンドを実行してクラスター化した GFS2 ファイルシステムを作成するには、以下の情報が必要です。

- ロックプロトコル/モジュール名 (クラスターの **lock_dlm**)
- クラスター名
- ジャーナルの数 (ファイルシステムをマウントするノード 1 つに対してジャーナルが 1 つ必要)



注記

mkfs.gfs2 コマンドで GFS2 ファイルシステムを作成すると、ファイルシステムのサイズは縮小できなくなります。ただし、**gfs2_grow** コマンドで、既存のファイルシステムのサイズを大きくすることは可能です。

クラスター化 GFS2 ファイルシステムを作成する形式を以下に示します。Red Hat は、シングルノードのファイルシステムとしての GFS2 の使用には対応していないことに注意してください。

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

必要に応じて、**mkfs** コマンドを使用して GFS2 ファイルシステムを作成できます。そのとき、**-t** パラメーターで **gfs2** タイプのファイルシステムを指定し、その後に GFS2 ファイルシステムオプションを指定します。

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

警告

ClusterName:FSName パラメーターが適切に指定されていないと、ファイルシステムまたはロック領域が破損する可能性があります。

ClusterName

GFS2 ファイルシステムが作成されているクラスターの名前。

FSName

ファイルシステムの名前。1 ~ 16 文字まで指定できます。名前は、クラスター内のすべての **lock_dlm** ファイルシステムで固有にする必要があります。

NumberJournals

mkfs.gfs2 コマンドにより作成されるジャーナルの数を指定します。ファイルシステムをマウントするノードごとに、ジャーナルが 1 つ必要になります。GFS2 ファイルシステムの場合は、ファイルシステムを拡張せずにジャーナルを後で追加できます。

BlockDevice

論理または他のブロックデバイスを指定します

以下の表では、**mkfs.gfs2** コマンドオプション (フラグおよびパラメーター) を説明します。

表3.1 コマンドオプション: mkfs.gfs2

フラグ	パラメーター	説明
-c	**Megabytes**	各ジャーナルのクォータ変更ファイルの初期サイズを **メガバイト** に設定します。
-D		デバッグの出力を有効にします。
-h		ヘルプ。使用可能なオプションを表示します。
-J	**Megabytes**	ジャーナルのサイズをメガバイトで指定します。デフォルトのジャーナルサイズは 128 メガバイトです。最小サイズは 8 メガバイトです。ジャーナルのサイズが大きいほどパフォーマンスが向上しますが、使用するメモリーが、小さいジャーナルよりも多くなります。
-j	**Number**	**mkfs.gfs2** コマンドにより作成されるジャーナルの数を指定します。ファイルシステムをマウントするノードごとに、ジャーナルが 1 つ必要になります。このオプションを指定しない場合は、ジャーナルが 1 つ作成されます。GFS2 ファイルシステムでは、ファイルシステムを拡張せずに、後でジャーナルを追加できます。
-O		**mkfs.gfs2** コマンドが、ファイルシステムに書き込む前に確認を求めないようにします。

フラグ	パラメーター	説明
-p	**LockProtoName**	* 使用するロックプロトコルの名前を指定します。認識されるロックプロトコルには次のものがあります。 * **lock_dlm** - 標準のロックモジュール。クラスターファイルシステムに必要です。 * **lock_nolock** - GFS2 がローカルファイルシステムとして機能している場合に使用します (1 ノードのみ)。
-q		Quiet モード。何も表示しません。
-r	**Megabytes**	リソースグループのサイズをメガバイト単位で指定します。リソースグループの最小サイズは 32 メガバイトです。リソースグループの最大サイズは 2048 メガバイトです。リソースグループのサイズが大きくなると、非常に大きなファイルシステムでのパフォーマンスが向上することがあります。これが指定されていない場合は、**mkfs.gfs2** が、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を選択します。平均的なサイズのファイルシステムには 256 メガバイトのリソースグループがあり、ファイルシステムが大きくなると、パフォーマンスを向上させるためにリソースグループが大きくなります。

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にロックテーブルフィールドを指定する一意の識別子。**lock_nolock** プロトコルは、このパラメーターを使用しません。 * このパラメーターは、**ClusterName:FSName** のように、コロンで区切られた 2 つの要素 (スペースなし) で設定されます。 * **ClusterName** は、GFS2 ファイルシステムが作成されているクラスターの名前です。このクラスターのメンバーだけが、このファイルシステムを使用できます。 * **FSName** ファイルシステム名です。名前は、長さが 1 文字から 16 文字までで、クラスター内のすべてのファイルシステムで一意でなければなりません。
-V		コマンドのバージョン情報を表示します。

3.1.2. GFS2 ファイルシステムの作成

以下の例では、GFS2 ファイルシステムを 2 つ作成します。この両方のファイルシステムでは、lock_dlm がクラスターファイルシステムであるため、ファイルシステムが使用するロッキングプロトコルになります。両方のファイルシステムは、**alpha** という名前のクラスターで使用できます。

最初のファイルシステムの場合、ファイルシステム名は **mydata1** です。これには 8 個のジャーナルが含まれ、**/dev/vg01/lvol0** に作成されます。2 番目のファイルシステムの場合、このファイルシステム名は **mydata2** になります。これには 8 個のジャーナルが含まれ、**/dev/vg01/lvol1** に作成されます。

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

3.2. GFS2 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できるようにするには、そのファイルシステムが存在し、そのファイルシステムが存在するボリュームがアクティブな状態で、かつサポート対象のクラスタリングシステムとロッキングシステムが起動している必要があります。以上の要件を満たすと、Linux ファイルシステムと同様に GFS2 ファイルシステムをマウントできます。

注記

mount コマンドを使用して、手動でファイルシステムをマウントするのではなく、実稼働環境で GFS2 ファイルシステムを管理するには、常に Pacemaker を使用する必要があります。これは、手動でマウントすると、システムのシャットダウン時に問題が発生する可能性があるためです。

ファイルの ACL を操作するには、**-o acl** マウントオプションを指定してファイルシステムをマウントする必要があります。ファイルシステムが **-o acl** マウントオプションを指定せずにマウントされている場合は、(**getfacl** で) ACL を表示できますが、(**setfactl** で) それを設定することはできません。

3.2.1. オプションなしで GFS2 ファイルシステムのマウント

この例では、**/dev/vg01/lvol0** の 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ます。

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

3.2.2. マウントオプションを指定する GFS2 ファイルシステムのマウント

以下は、GFS2 ファイルシステムをマウントするコマンドの形式です。マウントオプションも指定します。

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

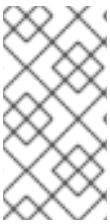
BlockDevice

GFS2 ファイルシステムを置くブロックデバイスを指定します。

MountPoint

GFS2 ファイルシステムがマウントされるディレクトリーを指定します。

-o option 引数は、GFS2 固有のオプションまたは許容できる標準の Linux **mount -o** オプション、もしくはその両方の組み合わせで設定されます。複数の **オプション** パラメーターはコンマで区切ります (スペースは使用しません)。



注記

mount コマンドは、Linux のシステムコマンドです。これらの GFS2 固有のオプションに加えて、他の標準の **mount** コマンドオプション (**-r** など) を使用することもできます。その他の Linux **mount** コマンドオプションは、Linux **mount** の man ページを参照してください。

次の表は、マウント時に GFS2 に渡すことができる GFS2 固有の **-o option** の値を説明します。

注記

この表では、ローカルファイルシステムでのみ使用されるオプションを説明します。ただし、Red Hat は、シングルノードのファイルシステムとして GFS2 を使用することはサポートしていないことに注意してください。Red Hat は、クラスターファイルシステムのスナップショットのマウントを目的 (例: バックアップ) とした、単一ノードの GFS2 ファイルシステムを引き続きサポートします。

表3.2 GFS2 固有のマウントオプション

オプション	説明
acl	ファイル ACL を操作できるようにします。ファイルシステムが **acl** マウントオプションなしでマウントされている場合は、(**getfacl** で) ACL を表示できますが、(**setfacl** で) 設定することはできません。
data=[ordered|writeback]	**data=ordered** が設定されていると、トランザクションにより変更したユーザーデータは、トランザクションがディスクにコミットされる前にディスクにフラッシュされます。これにより、クラッシュ後に、初期化されていないブロックがファイルに表示されなくなります。**data=writeback** モードが設定されていると、そのユーザーのデータがダーティーになるとディスクに書き込まれます。ここでは、**ordered** モードと同じような一貫性保証は行わないため、ワークロードによって少し速くなります。デフォルト値は **ordered** モードです。
ignore_local_fs **注意:** このオプションは、GFS2 ファイルシステムを共有しているとき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** フラグが自動的に有効になります。
localflocks **注意:** このオプションは、GFS2 ファイルシステムを共有しているときは使用しないでください。	VFS (仮想ファイルシステム) レイヤーが flock および fcntl をすべて実行するように、GFS2 に命令します。**localflocks** フラグは、**lock_nolock** により自動的に有効になります。
lockproto=LockModuleName	ユーザーがファイルシステムで使用するロックプロトコルを指定できるようにします。**LockModuleName** を指定しない場合、ロックプロトコル名は、ファイルシステムのスーパーブロックから読み取られます。
locktable=LockTableName	ユーザーがファイルシステムで使用するロックテーブルを指定できるようにします。
quota=[off/account/on]	ファイルシステムのクォータのオンとオフを切り替えます。クォータを **account** 状態に設定すると、UID ごとまたは GID ごとの使用統計がファイルシステムにより正しく管理され、limit と warn の値は無視されます。デフォルト値は **off** です。

オプション	説明
errors=panic|withdraw	**error=panic** を指定すると、ファイルシステムのエラーによりカーネルパニックが発生します。**errors=withdraw** を指定 (デフォルトの動作) は、ファイルシステムのエラーによりシステムはファイルシステムから撤退し、次にシステムを再起動するまでアクセスできなくなります。場合によっては、システムは稼働したままになります。
discard/nodiscard	GFS2 が、解放されたブロックの破棄 I/O 要求を生成します。これを適切なハードウェアで使用して、シンプロビジョニングや同様のスキームを実装できます。
barrier/nobarrier	ジャーナルのフラッシュ時に GFS2 が I/O バリアを送信します。デフォルト値は **on** です。基となるデバイスが I/O バリアに対応していないと、このオプションは自動的に **off** になります。ブロックデバイスが、書き込みキャッシュの内容を失わないように設計されていない場合 (UPS 上にある場合や、書き込みキャッシュがない場合など) は常に、GFS2 で I/O バリアを使用することが強く推奨されます。
quota_quantum=secs	クォータ情報の変更が、クォータファイルに書き込まれる前にノードに留まる秒数を設定します。これは、このパラメーターの設定に推奨される方法です。この値には、ゼロよりも大きい秒数 (整数) を設定します。デフォルトは 60 秒です。短く設定すると、遅延クォータ情報の更新が速くなり、そのクォータを超過する可能性が低くなります。長く設定すると、クォータに関連するファイルシステムの操作が速くなり、より効率的になります。
statfs_quantum=secs	**statfs** の低速バージョンを設定するには、**statfs_quantum** を 0 に設定することが推奨されます。デフォルト値は 30 秒で、**statfs** 変更がマスターの **statfs** ファイルに同期されるまでの最大期間を設定します。速いけど正確さが劣るように、もしくは遅くて正確さが上げるように **statfs** の値を調整できます。このオプションを 0 に設定すると、**statfs** は、常に true 値を報告します。
statfs_percent=value	期間の有効期限が切れていない場合でも、マスター **statfs** ファイルに同期される前のローカルベースに関する **statfs** 情報の最大変化率の上限を提供します。**statfs_quantum** の設定が 0 の場合は、この設定が無視されます。

3.2.3. GFS2 ファイルシステムのマウント解除

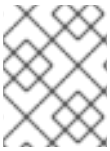
Pacemaker を介して自動的で行わずに、手動でマウントした GFS2 ファイルシステムは、システムのシャットダウン時にファイルシステムのマウントが解除されても、システムにより認識されません。これにより、GFS2 リソースエージェントでは GFS2 ファイルシステムのマウントが解除されなくなります。GFS2 リソースエージェントがシャットダウンすると、標準のシャットダウンプロセスがクラスターインフラストラクチャーを含む残りのユーザープロセスをすべて強制終了し、ファイルシステムのマウントを解除しようとします。このマウントの解除はクラスターインフラストラクチャーがないと失敗し、システムがハングします。

GFS2 ファイルシステムのマウントを解除する際にシステムをハングさせないようにするには、次のいずれかを行ってください。

- GFS2 ファイルシステムを管理する際に常に Pacemaker を使用します。
- GFS2 ファイルシステムが、**mount** コマンドを使用して手動でマウントされている場合は、システムを再起動またはシャットダウンする前に、**umount** コマンドを使用してそのファイルシステムのマウントを手動で解除します。

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

GFS2 ファイルシステムは、**umount** コマンドを使用すれば、他の Linux ファイルシステムと同じ方法でマウントを解除できます。



注記

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

用途

```
umount MountPoint
```

MountPoint

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

3.3. GFS2 ファイルシステムのバックアップ

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

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

バックアップの完了後にキャッシュを削除すると、他のノードがクラスターのロック/キャッシュの所有権を再取得するのに必要な時間が短縮されます。ただし、バックアッププロセスが開始する前に、キャッシュしていたデータのキャッシュを他のノードが停止するため、これは理想的ではありません。バックアップ完了後に次のコマンドを実行すると、キャッシュを削除できます。

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

タスクがノード間で分割されるように、クラスターの各ノードがそれ自体のファイルのバックアップを作成する方が高速です。これは、ノード固有のディレクトリーに **rsync** コマンドを使用するスクリプトで実現できます。

Red Hat では、SAN でハードウェアスナップショットを作成し、そのスナップショットを別のシステムに提供してそこにバックアップを行うことで、GFS2 バックアップを作成することを推奨しています。バックアップシステムはクラスターには含まれないため、**-o lockproto=lock_nolock** で、スナップショットをマウントする必要があります。

3.4. GFS2 ファイルシステムで動作の一時停止

ファイルシステムへの書き込み動作を一時停止するには、**dmestup suspend** コマンドを使用します。書き込みアクティビティーを停止することで、ハードウェアベースのデバイスのスナップショットを使用して、一貫性のある状態でファイルシステムをキャプチャーできます。**dmsetup resume** コマンドは、一時停止を終了します。

GFS2 ファイルシステムの動作を一時停止するコマンドの形式は、次のようになります。

```
dmsetup suspend MountPoint
```

この例では、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
```

GFS2 ファイルシステムで動作の一時停止を終了するコマンドの形式は、次のようになります。

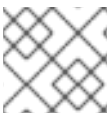
```
dmsetup resume MountPoint
```

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

```
# dmsetup resume /mygfs2
```

3.5. GFS2 ファイルシステムの拡張

gfs2_grow コマンドは、ファイルシステムが存在するデバイスが拡張された後に GFS2 ファイルシステムを拡張するのに使用されます。既存の GFS2 ファイルシステムで **gfs2_grow** コマンドを実行すると、ファイルシステムの現在の末尾とデバイスの末尾の間にある使用されていない領域がすべて、新しく初期化された GFS2 ファイルシステムの拡張子で埋められます。クラスター内の全ノードは、追加されたストレージ領域を使用できます。



注記

GFS2 ファイルシステムのサイズは縮小できません。

gfs2_grow コマンドは、マウントされたファイルシステムで実行する必要があります。次の手順では、/**mnt**/**gfs2** の マウントポイントで、論理ボリューム **shared_vg**/**shared_lv1** にマウントされているクラスターの GFS2 ファイルシステムのサイズが増大します。

手順

1. ファイルシステムでデータのバックアップを作成します。

2. 拡張するファイルシステムが使用している論理ボリュームが分からない場合は、**df mountpoint** コマンドを実行して確認できます。これにより、デバイス名が次の形式で表示されます。
/dev/mapper/vg-lv

   たとえば、デバイス名が **/dev/mapper/shared_vg-shared_lv1** の場合は、論理ボリュームが **shared_vg/shared_lv1** になります。

3. クラスターのノードの 1 つで、**lvextend** コマンドで、基礎となるクラスターボリュームを拡張します。RHEL 8.0 を実行している場合は、**--lockopt skiplv** オプションを使用して、通常の論理ボリュームロックを上書きします。これは、RHEL 8.1 以降を実行しているシステムでは必要ありません。
RHEL 8.1 以降の場合は、次のコマンドを使用します。

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

   RHEL 8.0 の場合は、次のコマンドを使用します。

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

4. RHEL 8.0 を実行している場合は、クラスターに追加した各ノードで論理ボリュームを更新して、そのノードでアクティブな論理ノードを更新します。RHEL 8.1 以降を実行しているシステムでは、論理ボリュームを拡張するとこの手順が自動的に行われるため、この手順は必要ありません。

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

5. クラスターのノードの 1 つで、GFS2 ファイルシステムのサイズを大きくします。論理ボリュームが全ノードで更新されていない場合はファイルシステムを拡張しないでください。この状態で拡張すると、ファイルシステムのデータがクラスター全体で利用できなくなる可能性があります。

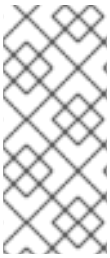
```
# gfs2_grow /mnt/gfs2
FS: Mount point:             /mnt/gfs2
FS: Device:                  /dev/mapper/shared_vg-shared_lv1
FS: Size:                    1310719 (0x13ffff)
DEV: Length:                 1572864 (0x180000)
The file system will grow by 1024MB.
gfs2_grow complete.
```

6. すべてのノードで **df** コマンドを実行して、そのファイルシステムで新しい領域が使用できるようになったことを確認します。すべてのノードで **df** コマンドを実行して同じファイルシステムのサイズを表示するには、最大 30 秒かかることに注意してください。

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

3.6. GFS2 ファイルシステムへのジャーナルの追加

GFS2 では、ファイルシステムのマウントが必要なクラスターのノードごとに、1 つのジャーナルが必要になります。追加のノードをクラスターに追加する場合は、**gfs2_jadd** コマンドで、ジャーナルを GFS2 ファイルシステムに追加できます。基となる論理ボリュームを拡張しなくても、ジャーナルはいつでも動的に GFS2 ファイルシステムに追加できます。**gfs2_jadd** コマンドは、マウントしたファイルシステムで実行する必要がありますが、クラスターのノードの 1 つでのみ実行する必要があります。その他のすべてのノードは、拡張が発生したことを検知します。



注記

GFS2 ファイルシステムが満杯になると、ファイルシステムを含む論理ボリュームが拡張されてファイルシステムよりも大きくなっている場合でも、**gfs2_jadd** コマンドは失敗します。これは、GFS2 ファイルシステムではジャーナルが埋め込みメタデータではなくプレーンファイルであるため、基となる論理ボリュームを拡張するだけではジャーナル用の領域が確保されないためです。

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

```
# gfs2_edit -p jindex /dev/sasdrives/scratch|grep journal
  3/3 [fc7745eb] 4/25 (0x4/0x19): File    journal0
  4/4 [8b70757d] 5/32859 (0x5/0x805b): File    journal1
  5/5 [127924c7] 6/65701 (0x6/0x100a5): File    journal2
```

GFS2 ファイルシステムにジャーナルを追加する基本的なコマンドの形式は次のようになります。

```
gfs2_jadd -j Number MountPoint
```

Number

新たに追加するジャーナルの数を指定します。

MountPoint

GFS2 ファイルシステムがマウントされているディレクトリーを指定します。

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

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

第4章 GFS2 のクォータ管理

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

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

注記

GFS2 は、標準の Linux クォータ機能に対応しています。これを使用するためには、**quota** RPM をインストールする必要があります。これは、GFS2 でクォータを管理するのに推奨される方法であるため、クォータを使用した GFS2 の新規デプロイメントには必ず使用してください。

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**

4.1. GFS2 ディスククォータの設定

GFS2 ファイルシステムのディスククォータを実装するには、3 つの手順を実行します。

ディスククォータを実装する手順は、以下のとおりです。

1. 強制またはアカウンティングモードでクォータを設定します。
2. 現在のブロック使用情報の入ったクォータデータベースファイルを初期化します。
3. クォータポリシーを割り当てます。(アカウンティングモードでは、このポリシーは強制されません。)

この各ステップは、以下のセクションで詳しく説明します。

4.1.1. 強制モードまたはアカウンティングモードでのクォータの設定

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

クォータが有効なファイルシステムをマウントするには、クラスターで GFS2 ファイルシステムリソースを作成するときに **options** 引数に **quota=off** を指定します。たとえば、次のコマンドは、作成している GFS2 の **Filesystem** リソースが、クォータが有効になっている状態でマウントされることを指定し

ます。

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

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

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

4.1.2. クォータデータベースファイルの作成

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

quotacheck コマンドは、クォータが有効になっているファイルシステムを調べ、ファイルシステムごとの現在のディスク使用量の表を作成します。次に、この表を使用して、オペレーティングシステムのディスク使用量を更新します。さらに、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** コマンドに **-u** オプションおよび **-g** オプションを指定します。ユーザーおよびグループのクォータを初期化するには、両方のオプションを指定する必要があります。たとえば、クォータが **/home** ファイルシステムに対して有効な場合は、**/home** ディレクトリーにファイルを作成します。

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

4.1.3. ユーザーごとにクォータの割り当て

最後の手順では、**edquota** コマンドでディスククォータを割り当てます。(**quota=account** オプションを指定して) ファイルシステムをアカウンティングモードでマウントした場合は、クォータが適用されないことに注意してください。

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

```
# edquota username
```

クォータが必要な各ユーザーに対して、この手順を実行します。たとえば、クォータが **/home** パーティションに対して有効になっている場合 (以下の例では **/dev/VolGroup00/LogVol02**) に **edquota testuser** コマンドを実行すると、システムのデフォルトとして設定されているエディターに次のように表示されます。

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

注記

EDITOR 環境変数により定義されたテキストエディターは、**edquota** により使用されます。エディターを変更するには、~/**.bash_profile** ファイルの **EDITOR** 環境変数を、使用するエディターのフルパスに設定します。

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

ソフトブロック制限は、使用可能な最大ディスク容量を定義します。

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

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

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

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

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

```
# quota testuser
```

setquota コマンドを使用して、コマンドラインからクォータを設定することもできます。**setquota** コマンドの詳細は、**setquota**(8) man ページを参照してください。

4.1.4. グループごとにクォータの割り当て

クォータは、グループごとに割り当てることもできます。(**account=on** オプションを指定して) ファイルシステムをアカウンティングモードでマウントした場合は、クォータが適用されません。

devel グループのグループクォータを設定するには (グループはグループクォータを設定する前に存在している必要があります)、次のコマンドを使用します。

```
# edquota -g devel
```

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

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

GFS2 ファイルシステムは inode のクォータを維持しないため、この列は GFS2 ファイルシステムには適用されず、空白になります。この制限を変更して、ファイルを保存します。

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

```
$ quota -g devel
```

4.2. GFS2 ディスククォータの管理

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

ユーザーが繰り返しクォータを超過したり、常にソフト制限に達している場合、システム管理者には、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 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決定できるように支援するか、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。

repquota ユーティリティーを実行して、ディスク使用量のレポートを作成できます。たとえば、**repquota /home** コマンドは次のような出力を生成します。

```
*** Report for user quotas on device /dev/mapper/VolGroup00-LogVol02
Block grace time: 7days; Inode grace time: 7days
   Block limits    File limits
User  used soft hard grace used soft hard grace
----------------------------------------------------------------------
root      --      36       0       0              4     0     0
kristin   --     540       0       0            125     0     0
testuser  --  440400  500000  550000          37418     0     0
```

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

```
# repquota -a
```

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

repquota コマンドが、基になるファイルシステムに関係なく、NFS ではサポートされないことに注意してください。

4.3. QUOTACHECK コマンドを使用した GFS2 ディスククォータの精度を保つ

クォータを無効にして実行してから一定期間後にファイルシステムでクォータを有効にする場合は、クォータファイルの作成、確認、修復を行う **quotacheck** コマンドを実行する必要があります。また、システムクラッシュ後にファイルシステムのマウントが正しく解除されていない場合に、クォータファイルが正確でない可能性があると思われる場合は、**quotacheck** コマンドを実行してください。

quotacheck コマンドの詳細は、**quotacheck(8)** の man ページを参照してください。

注記

ディスクアクティビティーが、計算するクォータ値に影響を与える可能性があるために、ファイルシステムがすべてのノードでアイドル状態になっている場合は、**quotacheck** を実行します。

4.4. QUOTASYNC コマンドを使用したクォータの同期

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

ユーザーまたはグループがクォータ制限に近づくと、GFS2 はクォータファイルの更新の間隔を動的に短縮し、制限を超えないようにします。クォータ同期の間の通常の期間は、調整可能なパラメーターである **quota_quantum** です。マウントオプションを指定する GFS2 ファイルシステムのマウント の GFS2 固有のマウントオプションの表に記載されているように、**quota_quantum=** マウントオプションを使用して、デフォルト値の 60 秒から変更できます。

quota_quantum パラメーターは、各ノードごとと、ファイルシステムをマウントするたびに設定する必要があります。**quota_quantum** パラメーターへの変更は、マウントが解除すると元に戻ります。**quota_quantum** の値は **mount -o remount** で更新できます。

quotasync コマンドを使用して、GFS2 が実行する自動更新の間に、ノードからディスク上のクォータファイルにクォータ情報を同期させます。**クォータ情報を同期する** 形式は以下の通りです。

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

u
ユーザーのクォータファイルを同期します。

g
グループのクォータファイルを同期します。

a
現在クォータが有効で、同期に対応するすべてのファイルシステムを同期します。-a を指定しない場合は、ファイルシステムのマウントポイントを指定する必要があります。

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

quota-quantum マウントオプションを指定して、同期の間隔を調整できます。

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

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

secs
GFS2 による通常のクォータファイル同期の新しい間隔を指定します。値を小さくすると競合が増え、パフォーマンスが低下する可能性があります。

次の例では、実行しているノードから、ファイルシステム **/mnt/mygfs2** のディスク上のクォータファイルに、キャッシュされているすべてのダーティークォータを同期します。

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

次の例では、ファイルシステムを論理ボリューム **/dev/volgroup/logical_volume** に再マウントする場合に、ファイルシステム **/mnt/mygfs2** でクォータファイルを更新する通常の間隔を、デフォルトから 1 時間 (3600 秒) に変更します。

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

第5章 GFS2 ファイルシステムの修復

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

重要

fsck.gfs2 コマンドは、すべてのノードからマウントが解除されているファイルシステムでのみ実行する必要があります。ファイルシステムが Pacemaker クラスターリソースとして管理されている場合は、ファイルシステムリソースを無効にしてファイルシステムのマウントを解除できます。**fsck.gfs2** コマンドを実行すると、ファイルシステムリソースが再び有効になります。**pcs resource disable** の **--wait** オプションで指定した **タイムアウト** 値は、秒単位の値になります。

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

ファイルシステムがリソースグループの一部である場合でも、暗号化されたファイルシステムのデプロイメントと同様に、ファイルシステムで fsck コマンドを実行するために、ファイルシステムリソースのみを無効にする必要があります。リソースグループ全体を無効にしないでください。

fsck.gfs2 コマンドが、システムの起動時に GFS2 ファイルシステムで実行しないようにするには、クラスターに GFS2 ファイルシステムリソースを作成する際に、**options** 引数の **run_fsck** パラメーターを設定できます。**run_fsck=no** を指定すると、**fsck** コマンドが実行しなくなります。

5.1. FSCK.GFS2 の実行に必要なメモリーの判定

fsck.gfs2 コマンドを実行すると、オペレーティングシステムとカーネルに使用されているメモリー以上のシステムメモリーを必要とする場合があります。特に大きなファイルシステムでは、このコマンドを実行するために追加のメモリーが必要になる場合があります。

次の表に、ブロックサイズが 4K で、GFS2 ファイルシステムのサイズが 1TB、10TB、および 100TB の場合に、**fsck.gfs2** ファイルシステムの実行に必要なメモリーの概算値を示します。

GFS2 ファイルシステムのサイズ	fsck.gfs2 の実行に必要な概算メモリー
1 TB	0.16 GB
10 TB	1.6 GB
100 TB	16 GB

ファイルシステムのブロックサイズが小さいほど、必要になるメモリーが大きくなることに注意してください。たとえば、ブロックサイズが 1K の GFS2 ファイルシステムには、この表で示されるメモリーの 4 倍が必要になります。

5.2. GFS2 ファイルシステムの修復

GFS2 ファイルシステムを修復する **fsck.gfs2** コマンドの形式は、以下のとおりです。

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

-y

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

BlockDevice

GFS2 ファイルシステムを置くブロックデバイスを指定します。

この例では、**/dev/testvg/testlv** ブロックデバイスにある GFS2 ファイルシステムが修復されます。修復するすべての質問に、自動的に **yes** と答えます。

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

第6章 GFS2 パフォーマンスの向上

GFS2 設定には、ファイルシステムのパフォーマンスを改善するために分析できる多くの側面があります。

High Availability Add-On および Red Hat Global File System 2 (GFS2) を使用する Red Hat Enterprise Linux クラスターのデプロイおよびアップグレードの一般的な推奨事項は、Red Hat カスタマーポータルの記事 Red Hat Enterprise Linux Cluster, High Availability, and GFS Deployment Best Practices を参照してください。

6.1. GFS2 ファイルシステムのデフラグ

Red Hat Enterprise Linux には GFS2 用のデフラグツールはありませんが、個々のファイルを **filefrag** ツールで識別して一時ファイルにコピーし、一時ファイルの名前を変更してオリジナルを置き換えると、個々のファイルをデフラグできます。

6.2. GFS2 のノードロック機能

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

GFS2 では各ノードに独自のページキャッシュがあり、ここにオンディスクデータの一部が含まれている場合があります。GFS2 は、**glocks** (ジーロックと発音します) と呼ばれるロック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、キャッシュ管理機能を提供します。これは、基になる通信層として **分散ロックマネージャー** (DLM) を使用して実装されます。

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

glock が排他モード (DLM ロックモード: EX) で許可されると、単一ノードのみがその glock でデータをキャッシュできます。このモードは、データを変更するすべての操作 (**write** システムコールなど) で使用されます。

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

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

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

共有 glock を必要とするのは、**read** (バッファー付き)、**stat**、**readdir** などの操作のみです。排他的な glock を必要とするのは、**write** (バッファー付き)、**mkdir**、**rmdir**、**unlink** などの操作のみです。ダイ

レクト I/O の読み書き操作では、割り当てが行われていない場合は DF (deffered) 状態の glock が必要です。また、書き込みに割り合てが必要な場合は (つまりファイルの拡張または穴埋めには)、EX (exclusive) 状態の glock が必要です。

この場合、パフォーマンスに関する主な考慮事項が 2 つがあります。まず、読み込み専用操作は各ノードで独立して実行できるため、クラスター全体で並列処理が極めてよく機能します。次に、同じ inode へのアクセスを競うノードが複数あると、排他 glock を必要とする操作によりパフォーマンスが低下する場合があります。たとえば、GFS2 ファイルシステムのバックアップ の説明にあるように、GFS2 ファイルシステムのパフォーマンスにおいては、ファイルシステムのバックアップを行う場合などには、各ノードのワーキングセットを考慮することが重要になります。

これに加え、可能な限り GFS2 で マウントオプションの **noatime** または **nodiratime** を指定することが推奨されます。この場合、アプリケーションがこれを許可する **noatime** が優先されます。これにより、read が **atime** タイムスタンプを更新する際に排他的なロックが必要なくなります。

ワーキングセットやキャッシュの効率を懸念している場合、GFS2 では、GFS2 ファイルシステムのパフォーマンスを監視できるツール (Performance Co-Pilot や GFS2 トレースポイント) を利用できます。

注記

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

- inode は、すべてのノードで読み取り専用で使用されます。
- inode は、1 つのノードからのみ書き込みまたは変更されます。

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

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

読み書きのマッピングがある GFS2 のファイルに **mmap**() を行い、そこからのみ読み込む場合のみ、これは読み込みとしてのみカウントされます。

noatime mount パラメーターを設定しないと、読み込みが、ファイルのタイムスタンプを更新するための書き込みにもなります。すべての GFS2 ユーザーは、**atime** に特定の要件がない限り、**noatime** を使用してマウントすることが推奨されます。

6.3. POSIX ロックの問題

POSIX ロックを使用する場合は、以下の点を考慮する必要があります。

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

6.4. GFS2 によるパフォーマンスチューニング

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

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

SMTP でメールが届くと、デフォルトでは特定ユーザーのメールを特定のノードに渡すように個別のノードを設定できます。デフォルトノードが起動していない場合は、受信側のノードにより、メッセージがユーザーのメールスプールに直接保存されます。この設計は、通常のケースで 1 つのノードにキャッシュされた特定のファイルセットを維持することを目的としていますが、ノードに障害が発生した時にはダイレクトアクセスを許可します。

この設定により、GFS2 のページキャッシュを最大限に活用することができ、また障害が発生してもアプリケーション (**imap** は **smtp**) に対して透過的になります。

バックアップも、扱いにくい分野です。繰り返しますが、可能であれば、各ノードのワーキングセットを、その特定の inode のセットをキャッシュしているノードから直接バックアップを作成することが強く推奨されます。通常の時点で実行するバックアップスクリプトがあり、GFS2 で実行しているアプリケーションの応答時間が急に増大した思われる場合は、クラスターがページキャッシュを最も効率的に使用していない可能性が高くなります。

当然ながら、バックアップを実行するためにアプリケーションを停止できる場合は、特に問題にはなりません。一方、バックアップが 1 つのノードのみから実行する場合は、バックアップ完了後にそのノード上にファイルシステムの大部分がキャッシュされ、他のノードからの後続アクセスのパフォーマンスが低下します。次のコマンドを実行すると、バックアップ完了後にバックアップノード上の VFS ページキャッシュをドロップすることで、パフォーマンスの低下をある程度緩和できます。

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

ただし、この方法は、各ノードのワーキングセットが共有されているか、大半がクラスター全体で読み取り専用であるか、1 つのノードから大部分がアクセスされるようにするのと比べると、あまり良い解決策ではありません。

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

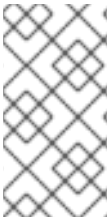
GFS2 キャシングの使用効率が悪いためにクラスターのパフォーマンスが影響を受けている場合は、I/O の待機時間が大幅に長くなることがあります。GFS2 のロックダンプ情報を利用すると、この問題の原因を特定できます。

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

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

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

debugfs ファイルを使用する場合は、**cat** コマンドを使用して、アプリケーションで問題が発生している間にファイル全体のコピーを取り (RAM が大きくて、キャッシュされた inode がある場合は時間がかかる場合があります)、後日、その結果得られたデータを調べることが最適な方法になります。



注記

debugfs ファイルのコピーを 2 部作成すると便利です。数秒または 1~2 分かかります。同じ glock 番号に関する 2 つのトレースの所有者情報を比較することで、ワークロードが進行中である (遅いだけ) か、それとも動かなくなったか (この場合は常にバグが原因であるため、Red Hat サポートに報告する必要があります) 判断できます。

debugfs ファイルで H: (ホルダー) で始まる行は、許可された、または許可されるのを待っているロック要求を表します。ホルダーの行 f: の flags フィールドは、W フラグが待機要求を参照し、H フラグが許可された要求を参照しています。待機要求が多数ある glock は、特定の競合が発生している可能性があります。

以下の表は、glock フラグおよび glock ホルダーフラグとその意味について紹介します。

表6.1 glock フラグ

フラグ	名前	意味
b	Blocking	ロックされたフラグが設定され、DLM から要求された操作がブロックされる可能性があることを示します。このフラグは、降格操作および try ロックに対して消去されます。このフラグの目的は、その他のノードがロックを降格する時間とは無関係に、DLM 応答時間の統計を収集できるようにすることです。
d	Pending demote	遅延している (リモートの) 降格要求
D	Demote	降格要求 (ローカルまたはリモート)
f	Log flush	この glock を解放する前にログをコミットする必要があります。
F	Frozen	リモートのノードからの返信が無視されます (復旧が進行中です)。このフラグは、異なるメカニズムを使用するファイルシステムのフリーズとは関係がなく、復元中にしか使用されません。
i	Invalidate in progress	この glock の下でページを無効にする過程です。

フラグ	名前	意味
I	Initial	DLM ロックがこの glock と関連付けられる場合に指定します。
l	Locked	glock は、状態を変更中です。
L	LRU	glock が LRU リストにあるときに設定します。
o	Object	glock がオブジェクトに関連付けられている (つまり、タイプ 2 の glock の場合は inode、タイプ 3 の glock の場合はリソースグループ) ときに設定されます。
p	Demote in progress	glock は、降格要求に応答中です。
q	Queued	ホルダーが glock にキューイングされると設定され、glock が保持されるとクリアされますが、残りのホルダーはありません。アルゴリズムの一部として使用され、glock の最小保持時間を計算します。
r	Reply pending	リモートノードから受信した返信の処理を待機中です。
y	Dirty	この glock を解放する前にデータをディスクにフラッシュする必要があります。

表6.2 glock ホルダーフラグ

フラグ	名前	意味
a	Async	glock の結果を待ちません (結果は後でポーリングします)。
A	Any	互換性のあるロックモードはすべて受け入れ可能です。
c	No cache	ロック解除時に DLM ロックを即時に降格します。

フラグ	名前	意味
e	No expire	後続のロック取り消し要求を無視します。
E	exact	完全一致するロックモードでなければなりません。
F	First	ホルダーがこのロックに最初に許可される場合に指定します。
H	Holder	要求したロックが許可されたことを示します。
p	Priority	キューの先頭にある待機ホルダー
t	Try	try ロックです。
T	Try 1CB	コールバックを送信する try ロックです。
W	Wait	要求完了の待機中にセットされます。

問題の原因になっている glock を特定したら、それがどの inode に関連しているかを調べることが次のステップになります。glock 番号 (G: 行の n:) はこれを示します。これは、**type/number** の形式になっており、**type** が 2 の場合、glock は inode glock となり、 **number** は inode の番号になります。inode を追跡するには、**find -inum number** を実行できます。ここでの **number** は、glock のファイルを 16 進数から 10 進数に変換した inode になります。

警告

ロックの競合が発生しているときにファイルシステムで **find** コマンドを実行すると、問題が悪化する可能性が高くなります。競合する inode を探す際に、アプリケーションを停止してから **find** コマンドを実行することを推奨します。

以下の表は、さまざまな glock タイプの意味を示しています。

表6.3 glock タイプ

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

タイプ番号	ロックタイプ	使用方法
3	Rgrp	リソースグループのメタデータ
4	Meta	スーパーブロック
5	Iopen	最後に閉じた inode の検出
6	Flock	**flock**(2) syscall
8	Quota	クォータ操作
9	Journal	ジャーナルミューテックス

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

リソースグループロックでキューに置かれている待機要求が多く表示され場合は、複数の理由が考えられます。1 つは、ファイルシステム内のリソースグループ数と比べ、多くのノードが存在するためです。または、ファイルシステムがほぼ満杯になっている可能性もあります (平均して、空きブロックの検索時間が長い場合があります)。どちらの場合も状況を改善するには、ストレージを追加し、**gfs2_grow** コマンドを使用してファイルシステムを拡張します。

6.6. データジャーナリングの有効化

通常、GFS2 はジャーナルにメタデータのみを書き込みます。ファイルの内容は、ファイルシステムのバッファーをフラッシュするカーネルの定期同期によって後続的にディスクに書き込まれます。ファイルで **fsync()** を呼び出すと、ファイルのデータがただちにディスクに書き込まれます。この呼び出しは、すべてのデータが安全に書き込まれたことをディスクに報告すると返されます。

ファイルデータは、メタデータとともにジャーナルにも書き込まれるため、データジャーナリングは、ファイルが非常に小さくなると **fsync()** が減少する可能性があります。ファイルサイズが増加すると、この利点が急速に低下します。中規模ファイルおよび大規模ファイルへの書き込みは、データジャーナリングが有効になっているとかなり遅くなります。

ファイルデータの同期に **fsync()** に依存しているアプリケーションは、データジャーナリングを使用してパフォーマンスが向上することがあります。データジャーナリングは、フラグ付きディレクトリー (およびそのすべてのサブディレクトリー) に作成されるすべての GFS2 ファイルに対して自動的に有効にできます。長さがゼロの既存のファイルも、データジャーナリングのオン/オフを切り替えることができます。

1 つのディレクトリー上でデータジャーナリングを有効にすると、そのディレクトリーは inherit jdata に設定され、その後そのディレクトリー内に作成されるファイルやディレクトリーはすべてジャーナル処理されることを示します。ファイルのデータジャーナリング機能は、**chattr** コマンドで有効にしたり無効にしたりできます。

次のコマンドでは、**/mnt/gfs2/gfs2_dir/newfile** ファイルに対するデータジャーナリングを有効にしてからフラグが正しく設定されているかどうかを確認します。

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

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

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

また、**chattr** コマンドを使用して、ディレクトリーに **j** フラグを設定できます。ディレクトリーにこのフラグを設定すると、その後にそのディレクトリー内に作成されたすべてのファイルとディレクトリーがジャーナリング処理されます。次の一連のコマンドは、**gfs2_dir** ディレクトリーに **j** フラグを設定し、フラグが正しく設定されたかどうかを確認します。この後、コマンドは、**/mnt/gfs2/gfs2_dir** ディレクトリーに、**newfile** という名前の新しいファイルを作成し、そのファイルに **j** フラグが設定されているかどうかを確認します。ディレクトリーに **j** フラグが設定されているため、**newfile** のジャーナリングも有効にする必要があります。

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

第7章 GFS2 ファイルシステムに伴う問題の診断と修正

以下の手順では、GFS2 の一般的な問題と、その対処方法に関する情報を説明します。

7.1. ノードに利用できない GFS2 ファイルシステム (GFS2 の WITHDRAW 機能)

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

以下は、GFS2 を無効にする可能性のある非整合の種類です。

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

GFS2 の無効を引き起こす (撤回) 可能性がある不一致の例は、ファイルの inode に対するブロック数が間違っています。GFS2 がファイルを削除すると、そのファイルが参照するすべてのデータおよびメタデータブロックがシステムにより削除されます。完了すると、inode のブロック数を確認します。ブロック数が 1 でない場合 (つまり、残りのすべてがディスクの inode 自体である場合)、inode のブロック数はファイルに使用されている実際のブロックと一致しないため、ファイルシステムが不整合であることを示します。

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

多くの場合、撤回した GFS2 ファイルシステムから復元する最善の方法は、ノードを再起動またはフェンスすることです。撤回した GFS2 ファイルシステムでは、クラスターの別のノードにサービスを再配置する機会が与えられます。サービスが再配置されれば、このコマンドを使用してノードを再起動するか、フェンスを強制的に実行できます。

```
pcs stonith fence node
```

警告

umount コマンドと **mount** コマンドを使用してファイルシステムのマウントを解除して再マウントしないようにしてください。代わりに **pcs** コマンドを使用する必要があります。このコマンドを使用しないと、ファイルシステムサービスが消えたことを Pacemaker が検出し、ノードをフェンスしてしまうからです。

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

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

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

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

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

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

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

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

ファイルシステムサービスに **-o errors=panic** オプションを指定してファイルシステムをマウントすることで、GFS2 の withdraw 機能を無効にできます。

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

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

内部的には、GFS2 の withdraw 機能は、ロックプロトコルを切断することで機能し、それ以降のすべてのファイルシステム操作で I/O エラーが発生するようにします。その結果、withdraw が発生すると、通常は、システムログに報告されたデバイスマッパーデバイスからの I/O エラーが多数表示されるようになります。

7.2. GFS2 ファイルシステムがハングし、単一ノードのリブートが必要

GFS2 ファイルシステムがハングし、それに対して実行したコマンドを返さなくても、ある特定のノードをリブートするとシステムが正常な状態に戻る場合には、ロックの問題もしくはバグの兆候である可

能性があります。これが発生した場合は、トラブルシューティングに使用する GFS2 データの収集 の説明に従って、これらの発生時に GFS2 データを収集し、Red Hat サポートにサポートチケットを作成してください。

7.3. GFS2 ファイルシステムがハングし、全ノードのリブートが必要

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

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

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

  a. 撤回が発生したノードを再起動します。

  ```
  # /sbin/reboot
  ```

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

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

  c. **gfs2_edit savemeta...** コマンドでメタデータを取得します。ファイルに十分な空きがあることを確認する必要があります。この例では、メタデータは **/root** ディレクトリーのファイルに保存されています。

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

  d. **gfs2-utils** パッケージを更新します。

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

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

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

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

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

  g. Red Hat サポートに、サポートチケットを作成します。GFS2 の撤回が発生したことを伝え、**sosreports** コマンドおよび **gfs2_edit savemeta** コマンドで生成したログとデバッグ情報を添付してください。

GFS2 の撤回では、ファイルシステムまたはそのブロックデバイスにアクセスしようとしているコマンドがハングすることもあります。このような場合は、クラスターを再起動するにはハードリブートが必要です。

GFS2 の withdraw 機能の詳細は、ノードで利用できない GFS2 ファイルシステム (GFS2 の withdraw 機能) を参照してください。

- このエラーは、ロックの問題またはバグを示している可能性があります。トラブルシューティングに使用する GFS2 データの収集 に従ってこの問題の発生時にデータを収集し、Red Hat サポートにサポートチケットを開きます。

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

新しいノードをクラスターに追加し、そのノードで GFS2 ファイルシステムをマウントできない場合は、GFS2 ファイルシステムのジャーナル数が、GFS2 ファイルシステムへのアクセスを試行しているノードよりも少ない可能性があります。ファイルシステムをマウントする GFS2 ホストごとに、ジャーナルが 1 つ必要です (ただし、**spectator** マウントオプションを設定してマウントした GFS2 ファイルシステムはジャーナルを必要としないため除外されます)。GFS2 ファイルシステムへのジャーナルの追加 で説明されているように、**gfs2_jadd** コマンドを使用して、GFS2 ファイルシステムにジャーナルを追加できます。

7.5. 空のファイルシステムで使用中と表示される領域

空の GFS2 ファイルシステムがある場合は、**df** コマンドで、占有されている領域が存在することが示されます。これは、GFS2 ファイルシステムのジャーナルがディスク上で領域 (ジャーナルの数 * ジャーナルサイズ) を消費するためです。多数のジャーナルがある GFS2 ファイルシステムを作成した場合、または指定したジャーナルサイズが大きい場合に **df** コマンドを実行すると、(ジャーナルの数 * ジャーナルのサイズ) が使用されているものとして表示されます。大量のジャーナルや大規模なジャーナルを指定していない場合でも、小さな GFS2 ファイルシステム (1GB 以下の範囲) には、デフォルトの GFS2 ジャーナルサイズで、使用中の大容量の領域が表示されます。

7.6. トラブルシューティングに使用する GFS2 データ収集

GFS2 ファイルシステムがハングし、それに対して実行したコマンドを返さず、Red Hat サポートのチケットを作成する必要がある場合は、最初に以下のデータを収集してください。

- 各ノード上のファイルシステム用の GFS2 ロックダンプの場合:

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

- 各ノードのファイルシステム用の DLM ロックダンプの場合: この情報は、**dlm_tool** を使用して確認できます。

```
dlm_tool lockdebug -sv lsname
```

このコマンドでは、**lsname** は、問題のファイルシステムに対して DLM が使用するロックスペース名です。**group_tool** コマンドの出力で、次の内容を確認できます。

- **sysrq -t** コマンドの出力
- **/var/log/messages** ファイルの内容

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

第8章 クラスター内の GFS2 ファイルシステム

Red Hat 高可用性クラスターで GFS2 ファイルシステムを設定するには、次の管理手順を使用します。

8.1. クラスターに GFS2 ファイルシステムを設定

次の手順で、GFS2 ファイルシステムを含む Pacemaker クラスターをセットアップできます。この例では、2 ノードクラスター内の 3 つの論理ボリューム上に 3 つの GFS2 ファイルシステムを作成します。

前提条件

- 両方のクラスターノードにクラスターソフトウェアをインストールして起動し、基本的な 2 ノードクラスターを作成している。
- クラスターのフェンシングを設定している。

Pacemaker クラスターの作成とクラスターのフェンシングの設定については、Pacemaker を使用した Red Hat High Availability クラスターの作成 を参照してください。

手順

1. クラスター内の両方のノードで、システムアーキテクチャーに対応する Resilient Storage のリポジトリーを有効にします。たとえば、x86_64 システムの Resilient Storage リポジトリーを有効にするには、以下の **subscription-manager** コマンドを入力します。

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

   Resilient Storage リポジトリーは、High Availability リポジトリーのスーパーセットであることに注意してください。Resilient Storage リポジトリーを有効にする場合は、High Availability リポジトリーを有効にする必要はありません。

2. クラスターの両方のノードで、**lvm2-lockd** パッケージ、**gfs2-utils** パッケージ、および **dlm** パッケージをインストールします。AppStream チャンネルおよび Resilient Storage チャンネルにサブスクライブして、これらのパッケージをサポートする必要があります。

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

3. クラスターの両方のノードで、**/etc/lvm/lvm.conf** ファイルの **use_lvmlockd** 設定オプションを **use_lvmlockd=1** に設定します。

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

4. グローバル Pacemaker パラメーター **no-quorum-policy** を **freeze** に設定します。

注記

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

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

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

5. **dlm** リソースをセットアップします。これは、クラスター内で GFS2 ファイルシステムを設定するために必要な依存関係です。この例では、**dlm** リソースを作成し、リソースグループ **locking** に追加します。

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

6. リソースグループがクラスターの両方のノードでアクティブになるように、**locking** リソースグループのクローンを作成します。

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

7. **locking** リソースグループの一部として **lvmlockd** リソースを設定します。

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

8. クラスターのステータスを確認し、クラスターの両方のノードで **locking** リソースグループが起動していることを確認します。

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

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

Full list of resources:

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

9. クラスターの 1 つのノードで、2 つの共有ボリュームグループを作成します。一方のボリュームグループには GFS2 ファイルシステムが 2 つ含まれ、もう一方のボリュームグループには GFS2 ファイルシステムが 1 つ含まれます。

注記

LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように設定することを推奨します。Pacemaker クラスターによって使用されるリモート物理ボリュームの起動順序の設定については、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

以下のコマンドは、共有ボリュームグループ **shared_vg1** を **/dev/vdb** に作成します。

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

以下のコマンドは、共有ボリュームグループ **shared_vg2** を **/dev/vdc** に作成します。

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

10. クラスター内の 2 番目のノードで以下を実行します。

    a. RHEL 8.5 以降で対応している LVM デバイスファイルを使用している場合は、共有デバイスを devices ファイルに追加します。

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

    b. 共有ボリュームグループごとにロックマネージャーを起動します。

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

11. クラスター内の 1 つのノードで、共有論理ボリュームを作成し、ボリュームを GFS2 ファイルシステムでフォーマットします。ファイルシステムをマウントするノードごとに、ジャーナルが 1 つ必要になります。クラスター内の各ノードに十分なジャーナルを作成してください。ロックテーブル名の形式は、**ClusterName:FSName** です。**ClusterName** は、GFS2 ファイルシステムが作成されているクラスターの名前です。**FSname** はファイルシステム名です。これは、クラスター経由のすべての **lock_dlm** ファイルシステムで一意である必要があります。

```
[root@z1 ~]# lvcreate --activate sy -L5G -n shared_lv1 shared_vg1
```

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

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

12. すべてのノードで論理ボリュームを自動的にアクティブにするために、各論理ボリュームに **LVM が有効** なリソースを作成します。

    a. ボリュームグループ **shared_vg1** の論理ボリューム **shared_lv1** に、**LVM が有効** なリソース **sharedlv1** を作成します。このコマンドは、リソースを含むリソースグループ **shared_vg1** も作成します。この例のリソースグループの名前は、論理ボリュームを含む共有ボリュームグループと同じになります。

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

    b. ボリュームグループ **shared_vg1** の論理ボリューム **shared_lv2** に、**LVM が有効** なリソース **sharedlv2** を作成します。このリソースは、リソースグループ **shared_vg1** に含まれます。

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

    c. ボリュームグループ **shared_vg2** の論理ボリューム **shared_lv1** に、**LVM が有効** なリソース **sharedlv3** を作成します。このコマンドは、リソースを含むリソースグループ **shared_vg2** も作成します。

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

13. リソースグループのクローンを新たに 2 つ作成します。

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

14. **dlm** リソースおよび **lvmlockd** リソースを含む **locking** リソースグループが最初に起動するように、順序の制約を設定します。

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

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

15. コロケーション制約を設定して、**vg1** および **vg2** のリソースグループが **locking** リソースグループと同じノードで起動するようにします。

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

16. クラスターの両ノードで、論理ボリュームがアクティブであることを確認します。数秒の遅延が生じる可能性があります。

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

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

17. ファイルシステムリソースを作成し、各 GFS2 ファイルシステムをすべてのノードに自動的にマウントします。
このファイルシステムは Pacemaker のクラスターリソースとして管理されるため、**/etc/fstab** ファイルには追加しないでください。マウントオプションは、**options=options** を使用してリソース設定の一部として指定できます。すべての設定オプションを確認する場合は、**pcs resource describe Filesystem** コマンドを実行します。

    以下のコマンドは、ファイルシステムのリソースを作成します。これらのコマンドは各リソースを、そのファイルシステムの論理ボリュームを含むリソースグループに追加します。

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

検証手順

1. GFS2 ファイルシステムが、クラスターの両方のノードにマウントされていることを確認します。

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

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

2. クラスターのステータスを確認します。

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

Full list of resources:

 smoke-apc      (stonith:fence_apc):    Started z1.example.com
 Clone Set: locking-clone [locking]
     Resource Group: locking:0
         dlm    (ocf::pacemaker:controld):      Started z2.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z2.example.com
     Resource Group: locking:1
         dlm    (ocf::pacemaker:controld):      Started z1.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z1.example.com
     Started: [ z1.example.com z2.example.com ]
 Clone Set: shared_vg1-clone [shared_vg1]
     Resource Group: shared_vg1:0
         sharedlv1      (ocf::heartbeat:LVM-activate):  Started z2.example.com
         sharedlv2      (ocf::heartbeat:LVM-activate):  Started z2.example.com
         sharedfs1      (ocf::heartbeat:Filesystem):    Started z2.example.com
         sharedfs2      (ocf::heartbeat:Filesystem):    Started z2.example.com
     Resource Group: shared_vg1:1
         sharedlv1      (ocf::heartbeat:LVM-activate):  Started z1.example.com
         sharedlv2      (ocf::heartbeat:LVM-activate):  Started z1.example.com
         sharedfs1      (ocf::heartbeat:Filesystem):    Started z1.example.com
         sharedfs2      (ocf::heartbeat:Filesystem):    Started z1.example.com
     Started: [ z1.example.com z2.example.com ]
 Clone Set: shared_vg2-clone [shared_vg2]
     Resource Group: shared_vg2:0
         sharedlv3      (ocf::heartbeat:LVM-activate):  Started z2.example.com
         sharedfs3      (ocf::heartbeat:Filesystem):    Started z2.example.com
     Resource Group: shared_vg2:1
         sharedlv3      (ocf::heartbeat:LVM-activate):  Started z1.example.com
         sharedfs3      (ocf::heartbeat:Filesystem):    Started z1.example.com
     Started: [ z1.example.com z2.example.com ]

...
```

関連情報

- GFS2 ファイルシステムの設定
- Microsoft Azure での Red Hat High Availability クラスターの設定
- AWS での Red Hat High Availability クラスターの設定
- Google Cloud Platform での Red Hat High Availability クラスターの設定

- Configuring Shared Block Storage for a Red Hat High Availability Cluster on Alibaba Cloud

8.2. クラスターでの暗号化 GFS2 ファイルシステムの設定

次の手順で、LUKS で暗号化した GFS2 ファイルシステムを含む Pacemaker クラスターを作成できます。この例では、論理ボリュームに 1 つの GFS2 ファイルシステムを作成し、そのファイルシステムを暗号化します。暗号化された GFS2 ファイルシステムは、LUKS 暗号化に対応する **crypt** リソースエージェントを使用してサポートされます。

この手順は、以下の 3 つの部分で設定されます。

- Pacemaker クラスター内で共有論理ボリュームを設定する
- 論理ボリュームを暗号化して **crypt** リソースを作成する
- GFS2 ファイルシステムで暗号化された論理ボリュームをフォーマットしてクラスター用のファイルシステムリソースを作成する

8.2.1. Pacemaker クラスター内での共有論理ボリュームの設定

前提条件

- 2 つのクラスターノードにクラスターソフトウェアをインストールして起動し、基本的な 2 ノードクラスターを作成している。
- クラスターのフェンシングを設定している。

Pacemaker クラスターの作成とクラスターのフェンシングの設定については、Pacemaker を使用した Red Hat High Availability クラスターの作成 を参照してください。

手順

1. クラスター内の両方のノードで、システムアーキテクチャーに対応する Resilient Storage のリポジトリーを有効にします。たとえば、x86_64 システムの Resilient Storage リポジトリーを有効にするには、以下の **subscription-manager** コマンドを入力します。

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

   Resilient Storage リポジトリーは、High Availability リポジトリーのスーパーセットであることに注意してください。Resilient Storage リポジトリーを有効にする場合は、High Availability リポジトリーを有効にする必要はありません。

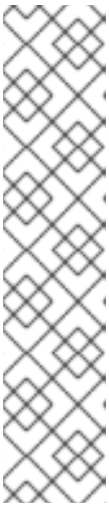
2. クラスターの両方のノードで、**lvm2-lockd** パッケージ、**gfs2-utils** パッケージ、および **dlm** パッケージをインストールします。AppStream チャンネルおよび Resilient Storage チャンネルにサブスクライブして、これらのパッケージをサポートする必要があります。

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

3. クラスターの両方のノードで、**/etc/lvm/lvm.conf** ファイルの **use_lvmlockd** 設定オプションを **use_lvmlockd=1** に設定します。

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

4. グローバル Pacemaker パラメーター **no-quorum-policy** を **freeze** に設定します。



注記

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

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

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

5. **dlm** リソースをセットアップします。これは、クラスター内で GFS2 ファイルシステムを設定するために必要な依存関係です。この例では、**dlm** リソースを作成し、リソースグループ **locking** に追加します。

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

6. リソースグループがクラスターの両方のノードでアクティブになるように、**locking** リソースグループのクローンを作成します。

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

7. **lvmlockd** リソースを、**locking** グループに追加します。

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

8. クラスターのステータスを確認し、クラスターの両方のノードで **locking** リソースグループが起動していることを確認します。

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

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

Full list of resources:

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

```
     dlm    (ocf::pacemaker:controld):      Started z2.example.com
     lvmlockd       (ocf::heartbeat:lvmlockd):      Started z2.example.com
 Started: [ z1.example.com z2.example.com ]
```

9. クラスターの 1 つのノードで、共有ボリュームグループを作成します。

注記

LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように設定することを推奨します。Pacemaker クラスターによって使用されるリモート物理ボリュームの起動順序の設定については、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

以下のコマンドは、共有ボリュームグループ **shared_vg1** を **/dev/sda1** に作成します。

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

10. クラスター内の 2 番目のノードで以下を実行します。

    a. RHEL 8.5 以降で対応している LVM デバイスファイルを使用している場合は、共有デバイスを devices ファイルに追加します。

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

    b. 共有ボリュームグループのロックマネージャーを起動します。

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

11. クラスター内の 1 つのノードで、共有論理ボリュームを作成します。

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

12. すべてのノードで論理ボリュームを自動的にアクティブにするために、論理ボリュームに **LVM が有効** なリソースを作成します。
以下のコマンドは、ボリュームグループ **shared_vg1** の論理グループ **shared_lv1** に、名前が **sharedlv1** で、**LVM が有効な** リソースを作成します。このコマンドは、リソースを含むリソースグループ **shared_vg1** も作成します。この例のリソースグループの名前は、論理ボリュームを含む共有ボリュームグループと同じになります。

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

13. 新しいリソースグループのクローンを作成します。

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

14. **dlm** および **lvmlockd** リソースを含む **locking** リソースグループが最初に起動するように、順序の制約を設定します。

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

15. コロケーション制約を設定して、**vg1** および **vg2** のリソースグループが **locking** リソースグループと同じノードで起動するようにします。

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

検証手順

クラスターの両ノードで、論理ボリュームがアクティブであることを確認します。数秒の遅延が生じる可能性があります。

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

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

8.2.2. 論理ボリュームの暗号化および暗号化リソースの作成

前提条件

- Pacemaker クラスターに共有論理ボリュームを設定している。

手順

1. クラスター内の 1 つのノードで、crypt キーを含めて新しいファイルを作成し、ファイルにパーミッションを設定して root でのみ読み取りできるようにします。

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

2. crypt キーを作成します。

```
[root@z1 ~]# dd if=/dev/urandom bs=4K count=1 of=/etc/crypt_keyfile
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000306202 s, 13.4 MB/s
[root@z1 ~]# scp /etc/crypt_keyfile root@z2.example.com:/etc/
```

3. **-p** パラメーターを使用して設定したパーミッションを保持した状態で、cript キーファイルをクラスター内の他のノードに配布します。

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

4. LVM ボリュームに暗号化デバイスを作成して、暗号化された GFS2 ファイルシステムを設定します。

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

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

5. **shared_vg1** ボリュームグループの一部として crypt リソースを作成します。

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

検証手順

crypt リソースが crypt デバイスを作成していることを確認します。この例では crypt デバイスは **/dev/mapper/luks_lv1** です。

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

8.2.3. GFS2 ファイルシステムで暗号化された論理ボリュームをフォーマットしてクラスター用のファイルシステムリソースを作成します。

前提条件

- 論理ボリュームを暗号化し、crypt リソースを作成している。

手順

1. クラスター内の 1 つのノードで、GFS2 ファイルシステムを使用してボリュームをフォーマットします。ファイルシステムをマウントするノードごとに、ジャーナルが 1 つ必要になります。クラスター内の各ノードに十分なジャーナルを作成してください。ロックテーブル名の形式は、**ClusterName:FSName** です。**ClusterName** は、GFS2 ファイルシステムが作成されているクラスターの名前です。**FSname** はファイルシステム名です。これは、クラスター経由のすべての **lock_dlm** ファイルシステムで一意である必要があります。

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

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

2. ファイルシステムリソースを作成し、GFS2 ファイルシステムをすべてのノードに自動的にマウントします。
ファイルシステムは Pacemaker のクラスターリソースとして管理されるため、**/etc/fstab** ファイルには追加しないでください。マウントオプションは、**options=options** を使用してリソース設定の一部として指定できます。すべての設定オプションを確認する場合は、**pcs resource describe Filesystem** コマンドを実行します。

   以下のコマンドは、ファイルシステムのリソースを作成します。このコマンドは、対象のファイルシステムの論理ボリュームリソースを含むリソースグループに、リソースを追加します。

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

検証手順

1. GFS2 ファイルシステムが、クラスターの両方のノードにマウントされていることを確認します。

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

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

2. クラスターのステータスを確認します。

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

Full list of resources:

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

```
  Clone Set: shared_vg1-clone [shared_vg1]
    Resource Group: shared_vg1:0
         sharedlv1      (ocf::heartbeat:LVM-activate):  Started z2.example.com
         crypt      (ocf::heartbeat:crypt) Started z2.example.com
         sharedfs1      (ocf::heartbeat:Filesystem):    Started z2.example.com
    Resource Group: shared_vg1:1
         sharedlv1      (ocf::heartbeat:LVM-activate):  Started z1.example.com
         crypt      (ocf::heartbeat:crypt)  Started z1.example.com
         sharedfs1      (ocf::heartbeat:Filesystem):    Started z1.example.com
       Started:  [z1.example.com z2.example.com ]
...
```

関連情報

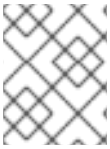
- GFS2 ファイルシステムの設定

8.3. RHEL7 から RHEL8 へ GFS2 ファイルシステムの移行

GFS2 ファイルシステムを含む RHEL 8 クラスターを設定する際に、既存の Red Hat Enterprise 7 論理ボリュームを使用できます。

Red Hat Enterprise Linux 8 では、LVM は、**clvmd** の代わりに LVM ロックデーモン **lvmlockd** を使用して、アクティブ/アクティブのクラスターで共有ストレージデバイスを管理します。これにより、アクティブ/アクティブのクラスターに共有論理ボリュームとして使用する必要がある論理ボリュームを設定する必要があります。また、これにより、**LVM が有効** なリソースを使用して LVM ボリュームを管理し、**lvmlockd** リソースエージェントを使用して **lvmlockd** デーモンを管理する必要があります。共有論理ボリュームを使用して、GFS2 ファイルシステムを含む Pacemaker クラスターを設定する手順は、クラスターに GFS2 ファイルシステムを設定 を参照してください。

GFS2 ファイルシステムを含む RHEL8 クラスターを設定する際に、既存の Red Hat Enterprise Linux 7 論理ボリュームを使用する場合は、RHEL8 クラスターから以下の手順を実行します。この例では、クラスター化された RHEL 7 論理ボリュームが、ボリュームグループ **upgrade_gfs_vg** に含まれます。



注記

既存のファイルシステムを有効にするために、RHEL8 クラスターの名前は、GFS2 ファイルシステムに含まれる RHEL7 クラスターと同じになります。

手順

1. GFS2 ファイルシステムを含む論理ボリュームが現在非アクティブであることを確認してください。すべてのノードがボリュームグループを使用して停止した場合にのみ、この手順は安全です。

2. クラスター内の 1 つのノードから、強制的にボリュームグループをローカルに変更します。

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

3. クラスター内の 1 つのノードから、ローカルボリュームグループを共有ボリュームグループに変更します。

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

4. クラスター内の各ノードで、ボリュームグループのロックを開始します。

```
[root@rhel8-01 ~]# vgchange --lockstart upgrade_gfs_vg
  VG upgrade_gfs_vg starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
[root@rhel8-02 ~]# vgchange --lockstart upgrade_gfs_vg
  VG upgrade_gfs_vg starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

この手順を実行すると、各論理ボリュームに、**LVM が有効** なリソースを作成できます。

第9章 GFS2 トレースポイントと GLOCK DEBUGFS インターフェイス

GFS 2 トレースポイントと glock **debugfs** インターフェイス両方に関する本書は、GFS2 の設計や GFS2 固有の問題のデバッグ方法の確認し、ファイルシステムの内部に精通している上級ユーザーを対象とします。

次のセクションでは、GFS2 トレースポイントと GFS2 **glocks** ファイルについて説明します。

9.1. GFS2 トレースポイントタイプ

現在、GFS2 トレースポイントには、**glock** (gee-lock と発音します) トレースポイント、**bmap** トレースポイント、および **ログ** トレースポイントがあります。これは、実行中の GFS2 ファイルシステムを監視するために使用できます。トレースポイントは、ハングやパフォーマンスの問題が再現可能で、問題のある操作中にトレースポイント出力を取得できる場合に特に役立ちます。GFS2 では、glock は主要なキャッシュ制御メカニズムで、GFS2 のコアのパフォーマンスを理解するときの鍵となります。bmap (ブロックマップ) トレースポイントを使用して、ブロック割り当てとブロックマッピング (ディスク上のメタデータツリーですでに割り当てられているブロックのルックアップ) を監視し、アクセスの局所性に関する問題を確認できます。ログトレースポイントは、ジャーナルに書き込まれ、ジャーナルから公開されるデータを追跡し、GFS2 の対象部分に関する有用な情報を提供できます。

トレースポイントは、できるだけ汎用性が保たれるように設計されています。つまり、Red Hat Enterprise Linux 8 を使用している場合には API を変更する必要がありません。一方、このインターフェイスを使用している場合は、これがデバッグインターフェイスであり、通常の Red Hat Enterprise Linux 8 API セットの一部ではないことを認識する必要があります。そのため、Red Hat では、GFS2 トレースポイントのいんたターフェースが変更されないことを保証できません。

トレースポイントは、Red Hat Enterprise Linux の汎用機能で、トレースポイントの対象範囲は GFS2 をはるかに超えています。特に、トレースポイントは、**blktrace** インフラストラクチャーの実装に使用されており、**blktrace** トレースポイントを GFS2 のトレースポイントと組み合わせて使用して、システムパフォーマンスの全体像を把握できます。トレースポイントの動作レベルでは、非常に短い時間で大量のデータを生成できます。トレースポイントは、有効になったときにシステムにかかる負荷が最小限になるように設計されていますが、何らかの影響は避けられません。さまざまな方法でイベントをフィルタリングすることで、データ量が減り、特定の状況を理解するのに有用な情報だけを取得することに集中できます。

9.2. トレースポイント

debugfs が、**/sys/kernel/debug** ディレクトリーの標準的な場所にマウントされていると仮定すると、トレースポイントは **/sys/kernel/debug/tracing/** ディレクトリーの下にあります。**events** サブディレクトリーには、指定可能なすべてのトレーシングイベントが格納されます。**gfs2** モジュールが読み込まれると、各 GFS2 イベントごとにサブディレクトリーを含む **gfs2** サブディレクトリーが現れます。**/sys/kernel/debug/tracing/events/gfs2** ディレクトリーの内容は以下のようになります。

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

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

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

特定のトレースポイントを有効にするために、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。これは、各イベントまたはイベントのセットにイベントフィルターを設定するために使用できる **filter** ファイルにも当てはまります。各イベントの意味は、以下で詳しく説明します。

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

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

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

トレースポイントのデータの読み取りには、**trace-cmd** と呼ばれるユーティリティーを利用できます。このユーティリティーの詳細は、 http://lwn.net/Articles/341902/ を参照してください。**trace-cmd** ユーティリティーは **strace** ユーティリティーと同様に使用することができ、さまざま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できます。

9.3. GLOCK

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

表9.1 Glock モードおよび DLM ロックモード

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

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

最初の要求時に glock に関連付けられます。この要求が成功すると、I(initial) フラグが glock に設定されます。glock debugfs インターフェイス の Glock Flags の表は、さまざまな glock フラグの意味を示しています。DLM が glock に関連付けられていると、DLM ロックは、少なくとも解放されるまで常に NL (Null) ロックモードのままになります。NL からロック解除への DLM ロックの降格は、常に glock の有効期間時の最後の操作になります。

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

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

注記

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

以下の表は、各 glock モードでキャッシュされる状態と、キャッシュされた状態がダーティーである可能性があるかどうかを示しています。これは、inode ロックとリソースグループロックの両方に適用されますが、リソースグループロック用のデータコンポーネントはなく、メタデータのみです。

表9.2 Glock モードおよびデータタイプ

Glock モード	キャッシュデータ	キャッシュメタデータ	ダーティーデータ	ダーティーメタデータ
UN	×	×	×	×
SH	○	○	×	×
DF	×	はい	×	×
EX	○	○	○	○

9.4. GLOCK DEBUGFS インターフェイス

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

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

```
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:EX n:2/50382 f:yfI t:EX d:EX/0 a:0 r:2
G:  s:SH n:5/302519 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/313874 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/271916 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:SH n:5/312732 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
```

上記の例は、1 つのノードの GFS2 ファイルシステムでポストマークのベンチマークの実行時に、**cat /sys/kernel/debug/gfs2/unity:myfs/glocks >my.lock** コマンドにより生成された (約 18MB ファイルからの) 一連の抜粋です。この図の glock は、glock ダンプのより興味深い機能のいくつかを示すために選択されています。

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

N: フィールド (数値) は、各項目に関連付けられた番号を示します。glock の場合、タイプ番号の後に glock 番号が続くため、上記の例では最初の glock は n:5/75320 となり、inode 75320 に関連する **iopen** glock を示します。inode と **iopen** glock の場合、glock 番号は常に inode のディスクブロック番号と同じになります。

注記

debugfs glock ファイルの glock 番号 (n: フィールド) は 16 進法ですが、トレースポイントの出力には 16 進法で表示されます。これには、これまでの経緯が原因です。glock 番号は常に 16 進法で書かれていますが、トレースポイントに 10 進数が選択されたため、この数字はその他のトレースポイント出力 (**blktrace** など) や **stat**(1) の出力と簡単に比較できます。

ホルダーと glock の両方に対するフラグの完全一覧は、Glock Flags の表と、Glock ホルダー の Glock Holder Flags の表に記載されています。ロック値のブロックの内容は、現在 **debugfs** インターフェイスからは利用できません。

以下の表は、さまざまな glock タイプの意味を示しています。

表9.3 glock の種類

タイプ番号	ロックタイプ	使用方法
1	trans	トランザクションのロック
2	inode	inode のメタデータとデータ
3	rgrp	リソースグループのメタデータ
4	meta	スーパーブロック

タイプ番号	ロックタイプ	使用方法
5	iopen	最後に閉じた inode の検出
6	flock	**flock**(2) syscall
8	quota	クォータ操作
9	journal	ジャーナルミューテックス

重要な glock フラグの 1 つは、l (locked) フラグです。これは、状態変更が行われるときに glock 状態へのアクセスを調整するのに使用されるビットロックです。これは、ステートマシンが DLM を介してリモートロック要求を送信するときに設定され、完全な操作が実行された場合にのみ消去されます。これは、複数のロック要求が送信され、その間にさまざまな無効化が発生することを意味します。

以下の表は、さまざまな glock フラグの意味を示しています。

表9.4 glock フラグ

フラグ	名前	意味
d	Pending demote	遅延している (リモートの) 降格要求
D	Demote	降格要求 (ローカルまたはリモート)
f	Log flush	この glock を解放する前にログをコミットする必要があります。
F	Frozen	リモートのノードからの返信が無視されます (復旧が進行中です)。
i	Invalidate in progress	この glock の下でページを無効にする過程です。
I	Initial	DLM ロックがこの glock と関連付けられる場合に指定します。
l	Locked	glock は、状態を変更中です。
L	LRU	glock が LRU リストにあるときに設定します。
o	Object	glock がオブジェクトに関連付けられている (つまり、タイプ 2 の glock の場合は inode、タイプ 3 の glock の場合はリソースグループ) ときに設定されます。

フラグ	名前	意味
p	Demote in progress	glock は、降格要求に応答中です。
q	Queued	ホルダーが glock にキューイングされると設定され、glock が保持されるとクリアされますが、残りのホルダーはありません。アルゴリズムの一部として使用され、glock の最小保持時間を計算します。
r	Reply pending	リモートノードから受信した返信の処理を待機中です。
y	Dirty	この glock を解放する前にデータをディスクにフラッシュする必要があります。

ローカルノードで保持されているノードと競合するモードでロックを取得する必要のあるノードからリモートコールバックを受け取ると、D (降格) または d (降格保留) のフラグのいずれかが設定されます。特定のロックの競合が発生した場合の枯渇状態を防ぐために、各ロックには最小保持時間が割り当てられます。最小保持時間の間にロックされていないノードは、その期間が経過するまでロックを保持できます。

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

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

9.5. GLOCK ホルダー

以下の表は、さまざまな glock ホルダーフラグの意味を示しています。

表9.5 Glock ホルダーフラグ

フラグ	名前	意味
a	Async	glock の結果を待ちません (結果は後でポーリングします)。
A	Any	互換性のあるロックモードはすべて受け入れ可能です。
c	No cache	ロック解除時に DLM ロックを即時に降格します。

フラグ	名前	意味
e	No expire	後続のロック取り消し要求を無視します。
E	Exact	完全一致するロックモードでなければなりません。
F	First	ホルダーがこのロックに最初に許可される場合に指定します。
H	Holder	要求したロックが許可されたことを示します。
p	Priority	キューの先頭にある待機ホルダー
t	Try	try ロックです。
T	Try 1CB	コールバックを送信する try ロックです。
W	Wait	要求完了の待機中にセットされます。

前述のように、それぞれ許可されたロック要求とキューに追加されたロック要求に設定されるため、ホルダーフラグで最も重要となるのは H (ホルダー) および W (待機) です。リスト内でのホルダーの順序は重要です。許可されたホルダーがある場合は常にキューの先頭にあり、その後にキューに追加されているホルダーが続きます。

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

glock サブシステムは、2 種類の try ロックに対応しています。これらは、(適切なバックオフと再試行により) 通常の順序からロックを解除でき、他のノードで使用されているリソースを回避するために使用できるため役立ちます。通常の t (試行) ロックは、その名前が示すとおりです。特別なことは行わない試行ロックです。一方、T (**try 1CB**) ロックは、DLM が現在互換性のないロックホルダーにコールバックを 1 つ送信するのを除き、t ロックと同じです。T (**try 1CB**) ロックには、**iopen** ロックがあります。これは、inode の **i_nlink** 数がゼロの場合にノード間の調整に使用されます。また、inode の割り当てに対応するノードを決定します。**iopen** glock は通常、共有状態で保持されますが、**i_nlink** リンク数がゼロになり、→**evict_inode**() が呼び出されると、T (**try 1CB**) セットによる排他的ロックが要求されます。ロックが許可されると、inode の割り当て解除が継続されます。ロックが許可されていない場合は、ロックの付与を妨げていたノードが glock を D (降格) フラグでマークします。これは、割り当て解除が忘れられていないことを確認するために、→**drop_inode**() 時間で確認されます。

つまり、リンク数がゼロであるがまだ開いている inode が、最後の **close**() が発生したノードによって割り当て解除されます。また、inode のリンク数もゼロにデクリメントされますが、inode はリンク数がゼロの特別な状態にあるとマークされますが、引き続きリソースグループのビットマップで使用されています。これは ext3 ファイルシステム 3 の孤立リストと同様に機能し、ビットマップの後続のリーダーは、再利用される可能性のある領域があることを認識し、再利用を試みることができます。

9.6. GLOCK トレースポイント

また、トレースポイントは、**blktrace** 出力と組み合わせたり、ディスク上のレイアウトの知識を組み合わせたりして、キャッシュ制御の正確さを確認できるように設計されています。次に、特定の I/O が発行され、正しいロックの下で完了したこと、および競合が存在しないことを確認できます。

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

gfs2_demote_rq トレースポイントは、ローカルおよびリモートの両方の降格要求を追跡します。ノードに十分なメモリーがある場合、ローカルの降格要求はほとんど発生せず、多くの場合は **umount** または場合によってはメモリーの再取得によって作成されます。リモート降格要求の数は、特定の inode またはリソースグループのノード間の競合の測定値です。

gfs2_glock_lock_time トレースポイントは、DLM への要求に要した時間に関する情報を提供します。このトレースポイントと組み合わせて使用するために、ブロック (**b**) フラグが glock に導入されました。

ホルダーにロックが付与されると、**gfs2_promote** が呼び出されます。これは、glock 状態が適切なモードのロックをすでにキャッシュしているため、状態変更の最終段階、または即時に付与できるロックが要求されたときに発生します。ホルダーがこの glock に付与される最初のホルダーである場合は、f (最初の) フラグがそのホルダーに設定されます。これは現在、リソースグループでのみ使用されます。

9.7. BMAP トレースポイント

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

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

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

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

ダイレクト I/O (**iomap**) は、ディスクとユーザーのバッファー間でファイルデータ転送を直接実行することができる別のキャッシュポリシーです。これは、キャッシュヒット率が低いと予想される状況で利点があります。**gfs2_iomap_start** トレースポイントおよび **gfs2_iomap_end** トレースポイントの両方がこれらの操作を追跡し、ダイレクト I/O、および操作タイプでダイレクト I/O のファイルシステム上の位置を追跡するのに使用できます。

9.8. ログトレースポイント

このサブシステムのトレースポイントは、ジャーナル (**gfs2_pin**) に追加されたブロックや削除されたブロックと、ログ (**gfs2_log_flush**) へのトランザクションのコミットにかかった時間を追跡します。これは、ジャーナリングのパフォーマンス問題をデバッグしようとする際に非常に便利です。

gfs2_log_blocks トレースポイントは、ログ内の予約ブロックを追跡し、たとえば、ログがワークロードに対して小さすぎるかどうかを表示するのに役立ちます。

gfs2_ail_flush トレースポイントは、AIL リストのフラッシュの開始と終了を追跡する点で **gfs2_log_flush** トレースポイントと似ています。AIL リストには、ログが通過したけど書き込まれていないバッファーが含まれています。これは、ファイルシステムで使用できるようにログ領域を解放するため、またはプロセスが **sync** または **fsync** を要求したときに定期的にフラッシュされます。

9.9. GLOCK の統計

GFS2 は、ファイルシステム内で何が発生しているかを追跡するのに役立つ統計を維持します。これにより、パフォーマンスの問題を特定できます。

GFS2 は、2 つのカウンターを維持します。

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

また、GFS2 では、平均と分散のペアが維持されます。平均/分散ペアは平滑化された指数予測であり、使用されるアルゴリズムはネットワークコードで往復時間を計算するために使用されるものです。

GFS2 で維持される平均と分散のペアは見積もられませんが、整数のナノ秒で表されます。

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

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

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

統計は **sysfs** ファイルに保存されます。

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

9.10. リファレンス

トレースポイントと GFS2 **glocks** ファイルに関する詳細情報は、以下の資料を参照してください。

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

第10章 PCP (PERFORMANCE CO-PILOT) を使用した GFS2 ファイルシステムの監視および分析

Performance Co-Pilot (PCP) は、GFS2 ファイルシステムの監視および分析に役立ちます。PCP での GFS2 ファイルシステムの監視は、Red Hat Enterprise Linux の GFS2 PMDA モジュールにより提供されます。これは、**pcp-pmda-gfs2** パッケージから入手できます。

GFS2 PMDA は、**debugfs** サブシステムで提供される GFS2 統計により提供されるメトリックを多数提供します。PMDA をインストールすると、**glocks** ファイル、**glstats** ファイル、および **sbstats** ファイルで指定した値が公開されます。このレポートでは、マウントされた各 GFS2 ファイルシステムの統計セットが報告されます。PMDA は、カーネル機能トレーサー (**ftrace**) が公開する GFS2 カーネルトレースポイントも利用します。

10.1. GFS2 PMDA のインストール

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

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

GFS2 PMDA は、デフォルトのインストールでは有効になっていません。PCP で GFS2 メトリックの監視を使用するには、インストール後に有効にする必要があります。

以下のコマンドを実行して PCP をインストールし、GFS2 PMDA を有効にします。PMDA インストールスクリプトは root で実行する必要があることに注意してください。

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

10.2. PMINFO ツールは、利用可能なパフォーマンスメトリックに関する情報を表示します。

pminfo ツールは、利用可能なパフォーマンスメトリックに関する情報を表示します。以下の例は、このツールで表示可能なさまざまな GFS2 メトリックを示しています。

10.2.1. 各ファイルシステムに現在存在している glock 構造の数を調べる

GFS2 glock メトリックは、現在マウントされた各 GFS2 ファイルシステムとそのロック状態について、glock 構造の数を示します。GFS2 では、glock は、DLM とキャッシングを 1 台のステートマシンにまとめるデータ構造です。各 glock には、1 つの DLM ロックと 1:1 の関係があり、そのロック状態のキャッシュを提供します。これにより、1 つのノードから実行される反復操作が DLM を繰り返し呼び出す必要がないため、不要なネットワークトラフィックを削減できます。

以下の **pminfo** コマンドは、ロックモードで、マウントされた GFS2 ファイルシステムごとの glock 数のリストを表示します。

-

```
# pminfo -f gfs2.glocks

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

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

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

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

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

10.2.2. タイプ別にファイルシステムごとに存在する glock 構造の数を調べる

GFS2 glstats メトリックは、ystem の各ファイルに存在する各タイプの glock のカウントを提供します。これらの多くは、通常 inode (inode およびメタデータ) またはリソースグループ (リソースグループメタデータ) タイプのいずれかになります。

以下の **pminfo** コマンドは、マウントされた GFS2 ファイルシステムごとに各タイプの Glock 数のリストを表示します。

```
# pminfo -f gfs2.glstats

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

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

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

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

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

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

```
    inst [1 or "afc_cluster:bin"] value 16

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

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

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

10.2.3. 待機状態にある glock 構造の数の確認

最も重要なホルダーフラグは H (holder: は要求されたロックが許可されたことを示します) および W (wait: は要求が完了するのを待つ間に設定されます) です。このフラグは、許可されたロック要求と、キューに格納されたロック要求にそれぞれ設定されます。

以下の **pminfo** コマンドは、マウントされた各 GFS2 ファイルシステムの Wait (W) ホルダーフラグを持つ glock 数のリストを表示します。

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

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

リソースグループロックでキューに置かれている待機要求が多く表示され場合は、複数の理由が考えられます。1 つは、ファイルシステム内のリソースグループ数と比べ、多くのノードが存在するためです。または、ファイルシステムがほぼ満杯になっている可能性もあります (平均して、空きブロックの検索時間が長い場合があります)。どちらの場合も状況を改善するには、ストレージを追加し、**gfs2_grow** コマンドを使用してファイルシステムを拡張します。

10.2.4. カーネルトレースポイントベースのメトリックを使用したファイルシステム操作レイテンシーの確認

GFS2 PMDA は、GFS2 カーネルトレースポイントからメトリックの収集をサポートします。デフォルトでは、これらのメトリックの読み取りは無効になっています。これらのメトリックをアクティベートすると、メトリック値を設定するためにメトリックが収集されるときに、GFS2 カーネルトレースポイントが有効になります。これらのカーネルトレースポイントメトリックが有効になっていると、パフォーマンススループットに若干影響する可能性があります。

PCP は **pmstore** ツールを提供します。これにより、メトリック値に基づいて PMDA 設定が変更できます。**gfs2.control.*** メトリックを使用すると、GFS2 カーネルトレースポイントを集約できます。以下の例は、**pmstore** コマンドを使用して、GFS2 カーネルトレースポイントをすべて有効にします。

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

このコマンドは、**debugfs** ファイルシステム内のすべての GFS2 トレースポイントで PMDA スイッチを切り替えます。PCP における GFS2 で利用可能なメトリックの完全リスト のメトリックの完全リストの表で は、各制御トレースポイントとその使用方法について説明しています。また、各トレースポイ

ントの効果や利用可能なオプションについては、**pminfo** のヘルプスイッチからも入手できます。

GFS2 のプロモートメトリックは、ファイルシステムでのプロモート要求の数をカウントします。これらのリクエストは、最初の試行時に発生したリクエスト数と、初回のプロモート要求後に付与されるその他によって分離されます。最初のプロモート数の減少とその他のプロモートの増加は、ファイル競合の問題を示している可能性があります。

昇格要求メトリックなどの GFS2 降格要求メトリックは、ファイルシステムで発生する降格要求の数をカウントします。ただし、これは、現在のノードからの要求と、システムの他のノードからの要求の間にも分割されます。リモートノードからの降格要求が多くなると、特定のリソースグループに対する 2 つのノード間の競合を示すことができます。

pminfo ツールは、利用可能なパフォーマンスメトリックに関する情報を表示します。この手順では、マウントされた各 GFS2 ファイルシステムに対して Wait (W) ホルダーフラグが付いた glock の数をリスト表示します。以下の **pminfo** コマンドは、マウントされた各 GFS2 ファイルシステムの Wait (W) ホルダーフラグを持つ glock 数のリストを表示します。

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

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

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

これらの値が通常の範囲と異なる場合にパフォーマンスの変化に気付くことができるように、ワークロードが問題なく実行されているときに観察される一般的な値を決定することは良い考えです。

たとえば、最初の試行で完了するのではなく、完了するのを待っているプロモートリクエスト数が変化するのに気付く場合があります。これは、次のコマンドの出力で判断できます。

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

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

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

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

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

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

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

次のコマンドの出力により、リモート降格リクエストの大幅な増加を特定できます (特に他のクラスターノードからの場合)。

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

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

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

以下のコマンドの出力は、ログフラッシュの予期せぬ増加を示しています。

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

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

10.3. PCP の GFS2 に使用できるメトリックの完全なリスト

次の表は、GFS2 ファイルシステム用の **pcp-pmda-gfs2** パッケージが提供するパフォーマンスメトリックの全リストを説明します。

表10.1 完全なメトリックリスト

メトリック名	説明
gfs2.glocks.*	現在システムにマウントされている GFS2 ファイルシステムごとに、現在存在する各状態の glock の数をカウントする glock 統計ファイル (**glocks**) から収集された情報に関するメトリック。
gfs2.glocks.flags.*	指定された glock フラグで存在する glock の数をカウントするメトリックの範囲
gfs2.holders.*	glock 統計ファイル (**glock**) から収集された情報に関するメトリック。システムで現在マウントされている各 GFS2 ファイルシステムに現存している各ロック状態のホルダーで glock の数をカウントします。
gfs2.holders.flags.*	指定のホルダーフラグで glock ホルダーの数をカウントするメトリックの範囲
gfs2.sbstats.*	システムで現在マウントされている各 GFS2 ファイルシステムのスーパーブロック統計ファイル (**sbstats**) から収集された情報に関するタイミングメトリック。

メトリック名	説明
gfs2.glstats.*	現在システムにマウントされている GFS2 ファイルシステムごとに、現在存在する glock のタイプの数をカウントする glock 統計ファイル (**glstats**) から収集された情報に関するメトリック。
gfs2.latency.grant.*	マウントされた各ファイルシステムに対して glock 付与要求が完了するまでの平均レイテンシーをマイクロ秒単位で計算するために、**gfs2_glock_queue** トレースポイントおよび **gfs2_glock_state_change** トレースポイントのデータを使用する派生メトリック。このメトリックは、付与レイテンシーが増加するときにファイルシステムの低速化の可能性を検出するのに役立ちます。
gfs2.latency.demote.*	マウントされた各ファイルシステムに対して glock 降格要求が完了するまでの平均レイテンシーをマイクロ秒単位で計算するために、**gfs2_glock_state_change** トレースポイントおよび **gfs2_demote_rq** トレースポイントのデータを使用する派生メトリック。このメトリックは、降格レイテンシーが増加するときにファイルシステムの低速化の可能性を検出するのに役立ちます。
gfs2.latency.queue.*	マウントされた各ファイルシステムに対して glock キュー要求が完了するまでの平均レイテンシーをマイクロ秒単位で計算するため、**gfs2_glock_queue** トレースポイントのデータを使用する派生メトリック。
gfs2.worst_glock.*	マウントされた各ファイルシステムで認識された現在最も悪い glock を計算する **gfs2_glock_lock_time** トレースポイントからデータを使用する派生メトリック。このメトリックは、同じロックが複数回推奨される場合に、潜在的なロック競合を検出し、ファイルシステムの速度が低下する場合に役立ちます。
gfs2.tracepoints.*	システムで現在マウントされている各ファイルシステムの GFS2 **debugfs** トレースポイントからの出力に関するメトリック。これらのメトリックの各サブタイプ (各 GFS2 トレースポイントのいずれか) は、制御メトリックを使用してオンまたはオフを個別に制御できます。
gfs2.control.*	PMDA でメトリック録画のオンまたはオフを切り替えるのに使用される設定メトリック。コントロールメトリックは、**pmstore** ツールを使用して切り替えられます。

10.4. ファイルシステムデータを収集するための最小限の PCP 設定の実行

以下の手順では、Red Hat Enterprise Linux で統計を収集するために最小限の PCP 設定をインストールする方法を説明します。この設定は、詳細な分析のためにデータを収集するために必要な、実稼働システムに最低限のパッケージを追加します。

結果として得られる **pmlogger** 出力の **tar.gz** アーカイブは、追加の PCP ツールを使用して分析でき、他のソースのパフォーマンス情報と比較できます。

手順

1. 必要な PCP パッケージをインストールします。

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

2. PCP の GFS2 モジュールをアクティブにします。

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

3. **pmcd** サービスおよび **pmlogger** サービスを起動します。

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

4. GFS2 ファイルシステムで操作を実行します。

5. **pmcd** サービスおよび **pmlogger** サービスを停止します。

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

6. 出力を集めて、ホスト名と現在の日時に基づいて名前が付けられた **tar.gz** ファイルに保存します。

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

10.5. 関連情報

- GFS2 トレースポイントと glock debugfs インターフェイス
- Performance Co-Pilot によるパフォーマンスの監視
- Performance Co-Pilot (PCP) の記事、ソリューション、チュートリアル、ホワイトペーパーのインデックス