Red Hat Enterprise Linux 8

GFS2 파일 시스템 구성

GFS2 파일 시스템 구성 및 관리 가이드

Last Updated: 2023-01-15

Red Hat Enterprise Linux 8 GFS2 파일 시스템 구성

GFS2 파일 시스템 구성 및 관리 가이드

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

법적 공지

초록

이 가이드에서는 Red Hat Enterprise Linux 8용 GFS2 파일 시스템 설정 및 관리에 대한 정보를 제공합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서 및 웹 속성에서 문제가 있는 언어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서에 대한 피드백에 감사드립니다. 어떻게 개선할 수 있는지 알려주십시오.

특정 문구에 대한 의견 제출

1. **Multi-page HTML** 형식으로 설명서를 보고 페이지가 완전히 로드된 후 오른쪽 상단 모서리에 **피드백** 버튼이 표시되는지 확인합니다.
2. 커서를 사용하여 주석 처리할 텍스트 부분을 강조 표시합니다.
3. 강조 표시된 텍스트 옆에 표시되는 **피드백 추가** 버튼을 클릭합니다.
4. 의견을 추가하고 **제출** 을 클릭합니다.

Bugzilla를 통해 피드백 제출(등록 필요)

1. Bugzilla 웹 사이트에 로그인합니다.
2. **버전** 메뉴에서 올바른 버전을 선택합니다.
3. **Summary** (요약) 필드에 설명 제목을 입력합니다.
4. **Description** (설명) 필드에 개선을 위한 제안을 입력합니다. 문서의 관련 부분에 대한 링크를 포함합니다.
5. **버그 제출**을 클릭합니다.

1장. GFS2 파일 시스템 배포 계획

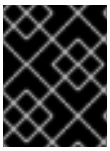
Red Hat Global File System 2(GFS2) 파일 시스템은 공유 이름 공간을 제공하고 공통 블록 장치를 공유하는 여러 노드 간에 일관성을 관리하는 64비트 대칭 클러스터 파일 시스템입니다. GFS2 파일 시스템은 로컬 파일 시스템에 최대한 가까운 기능 세트를 제공하는 동시에 노드 간에 전체 클러스터 일관성을 적용하는 데 사용됩니다. 이를 위해 노드는 파일 시스템 리소스에 클러스터 전체 잠금 체계를 사용합니다. 이 잠금 체계는 TCP/IP와 같은 통신 프로토콜을 사용하여 잠금 정보를 교환합니다.

예를 들어, Linux 파일 시스템 API는 GFS2의 클러스터형 특성을 완전히 투명하게 허용하지 않습니다. 예를 들어, GFS2에서 POSIX 잠금을 사용하는 프로그램은 클러스터의 다른 노드에서 **GETLK** 함수를 사용하지 않아야 합니다. 그러나 대부분의 경우 GFS2 파일 시스템의 기능은 로컬 파일 시스템의 기능과 동일합니다.

RHEL(Red Hat Enterprise Linux) 복구 스토리지 애드온은 GFS2를 제공하며, 이 애드온은 RHEL High Availability Add-On을 통해 GFS2에 필요한 클러스터 관리를 제공합니다.

gfs2.ko 커널 모듈은 GFS2 파일 시스템을 구현하고 GFS2 클러스터 노드에 로드됩니다.

GFS2에서 최상의 성능을 얻으려면 기본 설계에서 제공하는 성능 고려 사항을 고려해야 합니다. 로컬 파일 시스템과 마찬가지로, GFS2는 자주 사용되는 데이터의 로컬 캐싱으로 성능을 개선하기 위해 페이지 캐시를 사용합니다. 클러스터의 노드 전체에서 일관성을 유지하기 위해 *glock* 상태 시스템에서 캐시 제어를 제공합니다.



중요

Red Hat High Availability Add-On의 배포가 고객의 요구 사항을 충족하고 지원을 받을 수 있는지 확인하십시오. 인증 Red Hat 담당자에게 문의하여 배포 전에 구성을 확인하십시오.

1.1. 확인할 주요 GFS2 매개 변수

GFS2 파일 시스템을 설치하고 구성하기 전에 계획해야 하는 여러 가지 주요 GFS2 매개 변수가 있습니다.

GFS2 노드

클러스터에서 GFS2 파일 시스템을 마운트할 노드를 결정합니다.

파일 시스템 수

처음에 생성할 GFS2 파일 시스템 수를 결정합니다. 나중에 더 많은 파일 시스템을 추가할 수 있습니다.

파일 시스템 이름

각 GFS2 파일 시스템에는 고유한 이름이 있어야 합니다. 이 이름은 일반적으로 LVM 논리 볼륨 이름과 같으며, GFS2 파일 시스템이 마운트될 때 DLM 잠금 테이블 이름으로 사용됩니다. 예를 들어 이 가이드에서는 일부 예제 프로시저에서 파일 시스템 이름 **mydata1** 및 **mydata2** 를 사용합니다.

저널

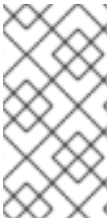
GFS2 파일 시스템의 저널 수를 결정합니다. GFS2에는 파일 시스템을 마운트해야 하는 클러스터의 각 노드에 대해 저널이 1개 있어야 합니다. 예를 들어 16노드 클러스터가 있지만 두 노드에서 파일 시스템만 마운트해야 하는 경우 저널 두 개만 있으면 됩니다. 추가 서버가 파일 시스템을 마운트할 때 **gfs2_jadd** 유틸리티를 사용하여 나중에 저널을 동적으로 추가할 수 있습니다.

스토리지 장치 및 파티션

파일 시스템에서 논리 볼륨(**lvmlockd**을 사용하여)을 생성하는 데 사용할 스토리지 장치 및 파티션을 결정합니다.

시간 프로토콜

GFS2 노드의 클럭이 동기화되었는지 확인합니다. Precision Time Protocol(PTP)을 사용하거나 구성에 필요한 경우 Red Hat Enterprise Linux 배포판과 함께 제공되는 NTP(Network Time Protocol) 소프트웨어를 사용하는 것이 좋습니다.
불필요한 inode 타임 스탬프 업데이트를 방지하려면 GFS2 노드의 시스템 클럭은 서로 몇 분 내에 있어야 합니다. 불필요한 inode 타임 스탬프 업데이트는 클러스터 성능에 심각한 영향을 미칩니다.



참고

여러 생성 및 삭제 작업이 동일한 디렉토리에 있는 두 개 이상의 노드에서 동시에 실행되는 경우 GFS2의 성능 문제가 발생할 수 있습니다. 이로 인해 시스템에서 성능 문제가 발생하는 경우 노드의 파일 생성 및 삭제를 가능한 한 해당 노드와 관련된 디렉토리로 로컬화해야 합니다.

1.2. GFS2 지원 고려 사항

Red Hat이 GFS2 파일 시스템을 실행하는 클러스터에 대해 지원을 받으려면 Red Hat의 지원 정책을 고려해야 합니다.

1.2.1. 최대 파일 시스템 및 클러스터 크기

다음 표에는 현재 최대 파일 시스템 크기와 GFS2에서 지원하는 노드 수가 요약되어 있습니다.

표 1.1. GFS2 지원 제한

매개변수	최대
노드 수	16 (x86, Power8 on PowerVM) 4 (z/VM 아래의 s390x)
파일 시스템 크기	지원되는 모든 아키텍처에서 100TB

GFS2는 64비트 아키텍처를 기반으로 하며, 이 아키텍처는 이론적으로 8 EB 파일 시스템을 수용할 수 있습니다. 현재 지원되는 것보다 더 큰 GFS2 파일 시스템이 필요한 경우 Red Hat 서비스 담당자에게 문의하십시오.

파일 시스템의 크기를 결정할 때는 복구 요구 사항을 고려해야 합니다. 매우 큰 파일 시스템에서 **fsck.gfs2** 명령을 실행하면 시간이 오래 걸릴 수 있으며 많은 양의 메모리를 사용할 수 있습니다. 또한 디스크 또는 디스크 하위 시스템 오류가 발생하는 경우 복구 시간은 백업 미디어의 속도에 따라 제한됩니다. **fsck.gfs2** 명령에 필요한 메모리 크기에 대한 자세한 내용은 fsck.gfs2를 실행하는 데 필요한 메모리 확인을 참조하십시오.

1.2.2. 최소 클러스터 크기

독립 실행형 시스템이나 클러스터 구성의 일부로 GFS2 파일 시스템을 구현할 수 있지만 Red Hat은 다음과 같은 예외를 제외하고 단일 노드 파일 시스템으로 GFS2 사용을 지원하지 않습니다.

- Red Hat은 백업 목적으로 필요에 따라 클러스터 파일 시스템의 스냅샷을 마운트하기 위해 단일 노드 GFS2 파일 시스템을 지원합니다.

- DLM을 사용하는 단일 노드 클러스터 마운트는 보조 사이트 재해 복구(DR) 노드의 목적으로 지원됩니다. 이 예외는 DR 용도로만 사용되며 기본 클러스터 워크로드를 보조 사이트로 전송하는 것은 아닙니다.
  예를 들어 기본 사이트는 오프라인 상태인 동안 보조 사이트에 마운트된 파일 시스템에서 데이터를 복사합니다. 그러나 기본 사이트에서 단일 노드 클러스터 보조 사이트로 직접 워크로드를 마이그레이션하는 것은 지원되지 않습니다. 전체 작업 부하를 단일 노드 보조 사이트로 마이그레이션해야 하는 경우 보조 사이트는 기본 사이트와 동일한 크기여야 합니다.

  단일 노드 클러스터에 GFS2 파일 시스템을 마운트할 때 파일 시스템 오류가 발생할 때 single-node 클러스터가 자체적으로 펜싱되지 않으므로 single-node 클러스터가 발생할 때 단일 노드 클러스터가 중단되도록 **errors=panic** 마운트 옵션을 지정하는 것이 좋습니다.

Red Hat은 단일 노드에 최적화된 다수의 고성능 단일 노드 파일 시스템을 지원하므로 일반적으로 클러스터 파일 시스템보다 오버헤드가 낮습니다. 단일 노드에서 파일 시스템을 마운트해야 하는 경우에는 이러한 파일 시스템을 GFS2에 사용하는 것이 좋습니다. Red Hat Enterprise Linux 8에서 지원하는 파일 시스템에 대한 자세한 내용은 파일 시스템 관리를 참조하십시오.

1.2.3. 공유 스토리지 고려 사항

Red Hat은 LVM 외부에서 GFS2 파일 시스템을 사용할 수 있지만 Red Hat은 공유 LVM 논리 볼륨에서 생성되는 GFS2 파일 시스템만 지원합니다.

GFS2 파일 시스템을 클러스터 파일 시스템으로 구성하는 경우 클러스터의 모든 노드가 공유 스토리지에 액세스할 수 있는지 확인해야 합니다. 일부 노드가 공유 스토리지에 액세스할 수 있고 다른 노드는 지원되지 않는 클러스터 구성입니다. 모든 노드가 실제로 GFS2 파일 시스템 자체를 마운트할 필요는 없습니다.

1.3. GFS2 포맷 고려 사항

GFS2 파일 시스템을 포맷하여 성능을 최적화하려면 다음 권장 사항을 고려해야 합니다.

중요

Red Hat High Availability Add-On의 배포가 고객의 요구 사항을 충족하고 지원을 받을 수 있는지 확인하십시오. 인증 Red Hat 담당자에게 문의하여 배포 전에 구성을 확인하십시오.

파일 시스템 크기: 작은 Is better
GFS2는 64비트 아키텍처를 기반으로 하며, 이 아키텍처는 이론적으로 8 EB 파일 시스템을 수용할 수 있습니다. 그러나 64비트 하드웨어에 대해 지원되는 현재 GFS2 파일 시스템의 최대 크기는 100TB입니다.

대규모 파일 시스템을 사용할 수 있지만 이러한 파일 시스템을 사용하지 않는 것이 좋습니다. 대부분의 경우 GFS2에서 사용하는 것이 작아집니다. 10TB 파일 시스템을 1TB 이상의 파일 시스템을 사용하는 것이 좋습니다.

GFS2 파일 시스템을 작게 유지해야 하는 몇 가지 이유가 있습니다.

- 각 파일 시스템을 백업하는 데 시간이 단축됩니다.
- **fsck.gfs2** 명령을 사용하여 파일 시스템을 확인해야 하는 경우 시간이 단축됩니다.
- **fsck.gfs2** 명령을 사용하여 파일 시스템을 확인해야 하는 경우 메모리가 줄어듭니다.

또한 유지 관리를 위해 리소스 그룹 수가 적으면 성능이 향상됩니다.

물론 GFS2 파일 시스템을 너무 작게 만드는 경우 공간이 부족하여 고유한 결과가 발생할 수 있습니다. 크기를 결정하기 전에 자신의 사용 사례를 고려해야 합니다.

블록 크기: 기본 (4K) 블록은 Preferred입니다.

mkfs.gfs2 명령은 장치 토폴로지에 따라 최적의 블록 크기를 추정하려고 합니다. 일반적으로 4K 블록은 Red Hat Enterprise Linux의 기본 페이지 크기(메모리)이기 때문에 기본 블록 크기입니다. 다른 파일 시스템과 달리, GFS2는 4K 커널 버퍼를 사용하여 대부분의 작업을 수행합니다. 블록 크기가 4K인 경우 커널은 버퍼를 조작하기 위해 작업을 적게 수행해야 합니다.

성능이 가장 높은 기본 블록 크기를 사용하는 것이 좋습니다. 많은 매우 작은 파일을 효율적으로 저장해야 하는 경우에만 다른 블록 크기를 사용해야 할 수도 있습니다.

비트맵 크기: 기본값(128MB)은 일반적으로 Optimal입니다.

mkfs.gfs2 명령을 실행하여 GFS2 파일 시스템을 생성할 때 저널 크기를 지정할 수 있습니다. 크기를 지정하지 않으면 기본적으로 대부분의 애플리케이션에 최적이어야 하는 128MB가 설정됩니다.

일부 시스템 관리자는 128MB가 과도한 것으로 간주하고 저널 크기를 최소 8MB 또는 더 보수적으로 32MB로 줄일 수 있습니다. 이 경우 효과가 있을 수 있지만 성능에 심각한 영향을 미칠 수 있습니다. 많은 저널링 파일 시스템과 마찬가지로, GFS2가 메타데이터를 쓸 때마다 메타데이터가 저널에 커밋됩니다. 이렇게 하면 시스템이 충돌하거나 전원이 끊어지면 저널이 마운트 시 자동으로 재생될 때 모든 메타데이터를 복구할 수 있습니다. 그러나 8MB의 저널을 채우기 위해 파일 시스템 활동이 많이 필요하지 않으며, 저널이 가득 차면 GFS2가 스토리지에 쓰기를 기다려야 하기 때문에 성능이 느려집니다.

일반적으로 기본 저널 크기 128MB를 사용하는 것이 좋습니다. 파일 시스템이 매우 작은 경우(예: 5GB) 128MB의 저널이 비현실될 수 있습니다. 더 큰 파일 시스템이 있고 공간을 절약할 수 있는 경우 256MB의 저널을 사용하면 성능이 향상될 수 있습니다.

리소스 그룹의 크기 및 수

mkfs.gfs2 명령을 사용하여 GFS2 파일 시스템을 생성하면 스토리지를 리소스 그룹이라는 균일 슬라이스로 나눕니다. 최적의 리소스 그룹 크기(32MB에서 2GB까지) 추정하려고 시도합니다. **mkfs.gfs2** 명령의 **-r** 옵션을 사용하여 기본값을 재정의할 수 있습니다.

최적의 리소스 그룹 크기는 파일 시스템을 사용하는 방법에 따라 다릅니다. 얼마나 넓을 지, 그리고 그것이 심하게 분할되는지 여부를 고려하십시오.

최적의 성능을 얻으려면 다양한 리소스 그룹 크기를 실험해야 합니다. GFS2를 전체 프로덕션에 배포하기 전에 테스트 클러스터를 테스트하는 것이 좋습니다.

파일 시스템에 너무 많은 리소스 그룹이 있는 경우 블록 할당이 너무 작으면 사용 가능한 블록을 위해 수십만 개의 리소스 그룹을 검색하는 데 너무 많은 시간이 낭비될 수 있습니다. 파일 시스템이 가득 차면 검색될 리소스 그룹이 많을수록 모두 클러스터 전체의 잠금이 필요합니다. 이로 인해 성능이 저하됩니다.

그러나 파일 시스템에 리소스 그룹이 너무 적으면 블록 할당이 너무 크면 동일한 리소스 그룹 잠금에 대해 블록 할당이 더 자주 제한될 수 있으므로 성능에도 영향을 미칠 수 있습니다. 예를 들어, 2GB의 리소스 그룹 5개로 구성된 10GB 파일 시스템이 있는 경우 동일한 파일 시스템이 32MB의 리소스 그룹 320으로 인한 경우 클러스터의 노드가 5개 리소스 그룹보다 더 자주 충돌합니다. 모든 블록 할당이 사용 가능한 블록 블록으로 하나를 찾기 전에 여러 리소스 그룹을 찾아야 할 수 있으므로 파일 시스템이 거의 가득하면 문제가 발생합니다. GFS2에서는 다음 두 가지 방법으로 이 문제를 해결하려고 합니다.

- 먼저 리소스 그룹이 완전히 가득 차면 블록이 해제될 때까지 향후 할당에 대해 이를 확인하지 않도록 합니다. 파일을 삭제하지 않으면 경합이 덜 심각합니다. 그러나 애플리케이션이 지속적으로 블록을 삭제하고 주로 전체 파일 시스템에 새 블록을 할당하는 경우 경합이 매우 높아지고 성능에 심각한 영향을 미칩니다.

- 두 번째, 기존 파일(예: 추가) 새 블록이 추가된 경우, GFS2에서 파일과 동일한 리소스 그룹에 새 블록을 그룹화하려고 합니다. 이는 성능을 높이기 위해 수행됩니다: 회전 디스크에서는 물리적으로 함께 닫을 때 검색 작업에 시간이 단축됩니다.

가장 안 좋은 시나리오는 모든 노드가 파일을 생성하는 중앙 디렉터리가 있는 경우입니다. 모든 노드가 동일한 리소스 그룹을 잠그기 위해 지속적으로 충돌할 것이기 때문입니다.

1.4. 클러스터의 GFS2 고려 사항

시스템에 포함할 노드 수를 결정할 때 고가용성과 성능 사이에 절충이 있습니다. 노드 수가 점점 많아지면 워크로드를 확장하기가 점점 어려워지고 있습니다. 따라서 Red Hat은 16개 이상의 클러스터 파일 시스템 배포에는 GFS2 사용을 지원하지 않습니다.

클러스터 파일 시스템을 배포하는 것은 단일 노드 배포를 위한 "디드랍"이 아닙니다. Red Hat은 시스템을 테스트하고 필요한 성능 수준에서 작동하는지 확인하기 위해 새로운 설치 시 약 8-12주 간의 테스트 기간을 허용할 것을 권장합니다. 이 기간 동안 모든 성능 또는 기능 문제가 발생할 수 있으며 모든 쿼리는 Red Hat 지원 팀에 문의해야 합니다.

클러스터 배포를 고려하는 고객은 나중에 가능한 지원 문제를 방지하기 위해 배포 전에 Red Hat 지원팀에서 구성을 검토할 것을 권장합니다.

1.5. 하드웨어 고려 사항

GFS2 파일 시스템을 배포할 때 다음 하드웨어 고려 사항을 고려하십시오.

- 더 높은 품질의 스토리지 옵션 사용
  GFS2는 iSCSI 또는 FCoE(Fibre Channel over Ethernet)와 같은 저렴한 공유 스토리지 옵션에서 작동할 수 있지만 캐싱 용량이 더 높은 품질 스토리지를 구입하면 성능이 향상됩니다. Red Hat은 파이버 채널 상호 연결을 통해 SAN 스토리지에서 최상의 품질, 온전성 및 성능 테스트를 수행합니다. 일반적으로 테스트를 먼저 테스트한 항목을 배포하는 것이 항상 좋습니다.

- 배포 전에 네트워크 장치 테스트
  높은 품질, 더 빠른 네트워크 장치를 사용하면 클러스터 통신 및 GFS2가 더 높은 안정성으로 더 빠르게 실행됩니다. 그러나 가장 값 비싼 하드웨어를 구입할 필요가 없습니다. 가장 비용이 많이 드는 네트워크 스위치에는 **fcntl** 잠금(flock)을 전달하는 데 사용되는 멀티 캐스트 패킷을 전달하는 데 문제가 있는 반면 저렴한 상용 네트워크 스위치는 더 빠르고 안정적입니다. Red Hat은 전체 프로덕션에 배포하기 전에 장비를 사용해 보는 것이 좋습니다.

2장. GFS2 사용 권장 사항

GFS2 파일 시스템을 배포할 때 고려해야 할 다양한 일반적인 권장 사항이 있습니다.

2.1. 시간 업데이트 구성

각 파일 inode 및 디렉터리 inode에는 3개의 타임스탬프와 연관된 세 개의 타임스탬프가 있습니다.

- **ctime** - inode 상태가 마지막으로 변경된 시간입니다.
- **mtime** - 파일(또는 디렉터리) 데이터가 마지막으로 수정된 시간입니다.
- **atime** - 파일(또는 디렉터리) 데이터에 마지막으로 액세스한 시간입니다.

시간 업데이트가 GFS2 및 기타 Linux 파일 시스템에서 기본적으로 활성화되어 있는 경우 파일을 읽을 때마다 inode를 업데이트해야 합니다.

애플리케이션이 제공하는 정보를 사용하는 애플리케이션이 많지 않으므로 이러한 업데이트에 **는** 불필요한 쓰기 트래픽 및 파일 잠금 트래픽이 많이 필요할 수 있습니다. 이 트래픽은 성능을 저하시킬 수 있으므로 **시간** 업데이트 빈도를 끄거나 줄이는 것이 좋습니다.

시간 업데이트의 영향을 줄이는 다음 방법을 사용할 수 있습니다.

- **relatime** (relative atime)으로 마운트하면 이전 **atime** 업데이트가 **mtime** 또는 **ctime** 업데이트보다 오래된 경우 atime을 업데이트합니다. 이는 GFS2 파일 시스템의 기본 마운트 옵션입니다.
- **noatime** 또는 **nodiratime** 을 사용하여 마운트합니다. **no atime** 을 사용하여 마운트하면 해당 파일 시스템의 파일 및 디렉터리 모두에 대해 시간 업데이트가 비활성화됩니다. **nodiratime** 으로 마운트하면 해당 파일 시스템의 디렉터리에 대해서만 시간 업데이트를 비활성화하는 것이 좋습니다. 일반적으로 애플리케이션이 허용하는 **no atime** 또는 **nodiratime** 마운트 옵션을 사용하여 GFS2 파일 시스템을 **noatime** 마운트 옵션으로 마운트하는 것이 좋습니다. GFS2 파일 시스템 성능에 대한 이러한 인수의 영향에 대한 자세한 내용은 GFS2 노드 잠금 을 참조하십시오.

noatime Linux 마운트 옵션으로 GFS2 파일 시스템을 마운트하려면 다음 명령을 사용합니다.

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

BlockDevice

GFS2 파일 시스템이 상주하는 블록 장치를 지정합니다.

MountPoint

GFS2 파일 시스템을 마운트해야 하는 디렉터리를 지정합니다.

이 예에서 GFS2 파일 시스템은 **/dev/vg01/lvol0** 에 있으며 **시간** 업데이트가 꺼진 경우 **/mygfs2** 디렉토리에 마운트됩니다.

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

2.2. VGPU 튜닝 옵션: 연구 및 실험

모든 Linux 파일 시스템과 마찬가지로 GFS2는 VFS(가상 파일 시스템)라는 계층 위에 있습니다. Multus는 대부분의 워크로드에 대한 캐시 설정에 적합한 기본값을 제공하며 대부분의 경우 변경할 필요가 없습니다. 그러나 제대로 실행되지 않는 워크로드(예: 캐시가 너무 크거나 작은 경우) **sysctl**(8) 명령을 사용하여

/proc/sys/vm 디렉터리의 **sysctl** 파일 값을 조정하여 성능을 향상시킬 수 있습니다. 이러한 파일에 대한 문서는 커널 소스 트리 **Documentation/sysctl/vm.txt** 에서 확인할 수 있습니다.

예를 들어, **더러운_background_ratio** 및 **vfs_cache_pressure** 의 값은 상황에 따라 조정할 수 있습니다. 현재 값을 가져오려면 다음 명령을 사용합니다.

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

다음 명령은 값을 조정합니다.

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

/etc/sysctl.conf 파일을 편집하여 이러한 매개변수의 값을 영구적으로 변경할 수 있습니다.

사용 사례에 맞는 최적의 값을 찾으려면 전체 프로덕션에 배포하기 전에 다양한 vGPU 옵션을 조사하고 테스트 클러스터에서 실험하십시오.

2.3. GFS2의 SELINUX

GFS2에서 SELinux(Security Enhanced Linux)를 사용하면 약간의 성능 저하가 발생합니다. 이러한 오버헤드를 방지하기 위해 SELinux가 강제 모드인 시스템에서도 GFS2와 함께 SELinux를 사용하지 않도록 선택할 수 있습니다. GFS2 파일 시스템을 마운트할 때 SELinux가 **mount**(8) 도움말 페이지에 설명된 **컨텍스트** 옵션 중 하나를 사용하여 각 파일 시스템 오브젝트에서 weekly **label** 요소를 읽으려고 시도하지 않도록 할 수 있습니다. SELinux는 파일 시스템의 모든 콘텐츠가 **컨텍스트** 마운트 옵션에 제공된 weekly **label** 요소로 레이블이 지정되었다고 가정합니다. 이는 또한 euro **label 요소를 포함할 수 있는 확장 속성 블록의 다른 디스크 읽기를 방지할 수 있으므로 처리** 속도를 향상시킵니다.

예를 들어 SELinux가 강제 모드로 설정된 시스템에서 다음 **mount** 명령을 사용하여 파일 시스템에 Apache 콘텐츠를 포함할 경우 GFS2 파일 시스템을 마운트할 수 있습니다. 이 레이블은 전체 파일 시스템에 적용되며 메모리에 남아 있으며 디스크에 기록되지 않습니다.

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

파일 시스템에 Apache 콘텐츠가 포함될지 확실하지 않은 경우 **public_content_rw_t** 또는 **public_content_t** 레이블을 사용하거나 새 레이블을 완전히 정의하고 관련 정책을 정의할 수 있습니다.

Pacemaker 클러스터에서는 항상 Pacemaker를 사용하여 GFS2 파일 시스템을 관리해야 합니다. GFS2 파일 시스템 리소스를 생성할 때 마운트 옵션을 지정할 수 있습니다.

2.4. GFS2를 통한 NFS 설정

GFS2 잠금 하위 시스템의 복잡성과 클러스터형 특성의 추가 복잡성으로 인해 GFS2를 통해 NFS를 설정하는 데 많은 예방 조치가 필요합니다.

주의

GFS2 파일 시스템이 NFS를 내보낸 경우 **localflocks** 옵션을 사용하여 파일 시스템을 마운트해야 합니다. **localflocks** 옵션을 사용하면 여러 위치에서 안전하게 GFS2 파일 시스템에 액세스할 수 없으며 여러 노드에서 동시에 GFS2를 내보내는 것은 불가능합니다. 이 구성을 사용할 때 한 번에 하나의 노드에만 GFS2 파일 시스템을 마운트해야 합니다. 이 방법은 각 서버의 POSIX 잠금을 로컬로 강제 적용하는 것입니다. 비 클러스터가 서로 독립적입니다. 이는 GFS2가 클러스터 노드에서 NFS에서 POSIX 잠금을 구현하려고 하는 경우 여러 가지 문제가 있기 때문입니다. NFS 클라이언트에서 실행되는 애플리케이션의 경우, 두 클라이언트가 서로 다른 서버에서 마운트되는 경우 두 클라이언트가 동일한 잠금을 동시에 보유할 수 있으므로 데이터 손상이 발생할 수 있습니다. 모든 클라이언트가 하나의 서버에서 NFS를 마운트하면 동일한 잠금을 부여하는 별도의 서버가 독립적으로 사라집니다. **localflocks** 옵션을 사용하여 파일 시스템을 마운트할지 확실하지 않은 경우 옵션을 사용하지 않아야 합니다. 데이터 손실을 방지하기 위해 즉시 Red Hat 지원팀에 문의하십시오. NFS를 통해 GFS2를 내보내는 것은 일부 상황에서 기술적으로 지원되는 것은 권장되지 않습니다.

기타 모든(NFS가 아닌) GFS2 애플리케이션의 경우 **localflocks** 를 사용하여 파일 시스템을 마운트하지 마십시오. 그러면 GFS2에서 클러스터의 모든 노드(클러스터 전체에서) 간에 POSIX 잠금 및 flock을 관리할 수 있습니다. **localflocks** 를 지정하고 NFS를 사용하지 않는 경우 클러스터의 다른 노드에는 서로의 POSIX 잠금 및 flock에 대한 지식이 없으므로 클러스터형 환경에서 안전하지 않습니다.

잠금 고려 사항 외에도 GFS2 파일 시스템에서 NFS 서비스를 구성할 때 다음 사항을 고려해야 합니다.

- Red Hat은 다음과 같은 특성을 가진 활성/수동 구성에서 NFSv3를 사용하여 Red Hat High Availability Add-On 구성만 지원합니다. 이 구성은 파일 시스템에 HA(고가용성)를 제공하고 실패한 노드가 한 노드에서 다른 노드로 NFS 서버가 실패할 때 **fsck** 명령을 실행하지 않아도 되므로 시스템 다운 타임을 줄입니다.
  - 백엔드 파일 시스템은 2~16개 노드 클러스터에서 실행되는 GFS2 파일 시스템입니다.
  - NFSv3 서버는 한 번에 전체 클러스터 노드에서 전체 GFS2 파일 시스템을 내보내는 서비스로 정의됩니다.
  - NFS 서버는 하나의 클러스터 노드에서 다른 클러스터 노드로 장애 조치(active/passive 구성)할 수 있습니다.
  - NFS 서버를 *제외한* GFS2 파일 시스템에 액세스할 수 없습니다. 여기에는 로컬 GFS2 파일 시스템 액세스와 Samba 또는 Clustered Samba를 통한 액세스가 모두 포함됩니다. 마운트된 클러스터 노드를 통해 로컬로 파일 시스템에 액세스하면 데이터가 손상될 수 있습니다.
  - 시스템에 NFS 할당량 지원이 없습니다.
- GFS2의 NFS 내보내기에는 **fsid=** NFS 옵션이 필요합니다.
- 클러스터에서 문제가 발생하는 경우(예: 클러스터가 정상 상태가 되고 펜싱이 완료되지 않음) 클러스터형 논리 볼륨과 GFS2 파일 시스템이 정지되고 클러스터가 정지될 때까지 액세스할 수 없습니다. 이 절차에서 정의된 것과 같은 간단한 장애 조치 솔루션이 시스템에 가장 적합한 솔루션인지 확인할 때는 이러한 가능성을 고려해야 합니다.

2.5. GFS2에서 제공하는 SAMBA(SMB 또는 WINDOWS) 파일

RuntimeClass와 함께 GFS2 파일 시스템에서 제공하는 Samba(SMB 또는 Windows) 파일을 사용하여 활성/활성 구성을 사용할 수 있습니다.

Samba 외부에서 Samba 공유의 데이터에 동시 액세스할 수 없습니다. 현재 Samba 파일 제공 속도가 느릴 때 GFS2 클러스터 리스가 지원되지 않습니다. Samba에 대한 지원 정책에 대한 자세한 내용은 RHEL Resilient Storage에 대한 지원 정책 - ctdb General Policies and Support Policies for RHEL Resilient Storage - Red Hat Customer Portal의 다른 프로토콜을 통해 gfs2 콘텐츠를 내보냅니다.

2.6. GFS2용 가상 머신 구성

가상 머신과 함께 GFS2 파일 시스템을 사용할 때 캐시를 강제로 사용하려면 각 노드의 VM 스토리지 설정을 올바르게 구성해야 합니다. 예를 들어 **libvirt** 도메인에 **cache** 및 io **에** 대한 이러한 설정을 포함하여 GFS2가 예상대로 작동하도록 허용해야 합니다.

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

또는 device 요소 내에서 **공유 가능한** 속성을 구성할 수 있습니다. 이는 해당 장치를 도메인 간에 공유할 것으로 예상됩니다(하이퍼바이저와 OS가 이를 지원하는 경우). **shareable** 을 사용하는 경우 해당 장치에 **cache='no'** 를 사용해야 합니다.

2.7. 블록 할당

데이터만 작성하는 애플리케이션에서는 일반적으로 블록 할당 방법이나 위치를 상관하지 않지만 블록 할당이 작동하는 방식에 대한 일부 지식이 성능을 최적화하는 데 도움이 될 수 있습니다.

2.7.1. 파일 시스템에 여유 공간 확보

GFS2 파일 시스템이 거의 가득 차면 블록 할당자가 새 블록을 할당할 공간을 찾기 어려울 수 있습니다. 결과적으로 allocator에 의해 제공되는 블록은 리소스 그룹 끝 또는 파일 조각화 가능성이 훨씬 더 많은 작은 조각으로 소커되는 경향이 있습니다. 이 파일 조각화로 인해 성능 문제가 발생할 수 있습니다. 또한 GFS2 파일 시스템이 거의 가득 차면, 모든 할당기가 여러 리소스 그룹을 통해 검색하는 데 더 많은 시간을 소비하고, 사용 가능한 공간이 충분한 파일 시스템에서 반드시 존재하지 않는 잠금 경합을 추가하는 것을 차단합니다. 이로 인해 성능 문제가 발생할 수 있습니다.

이러한 이유로 워크로드에 따라 다를 수 있지만 85 % 이상의 파일 시스템을 실행하지 않는 것이 좋습니다.

2.7.2. 가능한 경우 각 노드가 자신의 파일을 할당하도록 합니다.

GFS2 파일 시스템과 함께 사용할 애플리케이션을 개발하는 경우 가능한 경우 각 노드가 고유 파일을 할당하는 것이 좋습니다. 분산 잠금 관리자(DLM)가 작동하는 방식 때문에 모든 파일을 한 노드에 의해 할당되고 다른 노드에서 해당 파일에 블록을 추가해야 하는 경우 더 많은 잠금 경합이 발생합니다.

"lock master"라는 용어는 현재 클러스터의 원격 노드에서 시작되는 잠금 요청의 코디네이터인 노드를 나타내는 데 사용되었습니다. 잠금 요청 코디네이터에 대한 이 용어는 잠금 요청이 대기 중, 부여 또는 감소하는 리소스와 관련하여 실제로 리소스(DLM 용어)이기 때문에 약간 잘못 표시됩니다. DLM에서 용어가 사용되는 것은 DLM이 피어 투 피어 시스템이므로 "동일한 것"을 참조해야 합니다.

Linux 커널 DLM 구현에서 잠금이 처음 사용되는 노드는 잠금 요청의 코디네이터가 되고 그 뒤에는 변경되지 않습니다. 이는 Linux 커널 DLM의 구현 세부 정보이며 일반적으로 DLM의 속성이 아닙니다. 향후 업데이트로 특정 잠금에 대한 잠금 요청이 노드 간에 이동하도록 허용할 수 있습니다.

잠금 요청이 조정되는 위치는 요청 대기 시간에 미치는 영향을 제외하고 잠금 요청의 이니시에이터에 투

명합니다. 현재 구현의 결과 중 하나는 초기 워크로드의 불균형이 있는 경우(예: 다른 노드에서 I/O 명령을 수행하기 전에 전체 파일 시스템을 스캔)가 있으면 파일 시스템의 초기 검사를 수행한 노드와 비교하여 클러스터의 다른 노드의 잠금 대기 시간이 더 높을 수 있습니다.

대부분의 파일 시스템에서와 마찬가지로, GFS2 할당자는 동일한 파일에 블록을 서로 가깝게 유지하여 디스크 헤드의 이동을 줄이고 성능을 향상시키려고 합니다. 파일에 블록을 할당하는 노드는 새 블록에 대해 동일한 리소스 그룹을 사용하고 잠길 필요가 있습니다(해당 리소스 그룹의 모든 블록이 사용 중이 아닌 경우). 파일이 포함된 리소스 그룹의 잠금 요청 코디네이터가 데이터 블록을 할당하면 파일 시스템이 더 빨리 실행됩니다. 파일이 처음 열린 노드가 새 블록을 모두 작성하는 것이 더 빠릅니다).

2.7.3. 가능한 경우 사전 할당

파일이 미리 할당되면 블록 할당을 완전히 방지할 수 있으며 파일 시스템을 보다 효율적으로 실행할 수 있습니다. GFS2에는 데이터 블록을 사전 할당하는 데 사용할 수 있는 **fallocate(1)** 시스템 호출이 포함되어 있습니다.

3장. GFS2 파일 시스템 관리

GFS2 파일 시스템을 생성, 마운트, 확장 및 관리하는 데 사용하는 다양한 명령과 옵션이 있습니다.

3.1. GFS2 파일 시스템 생성

mkfs.gfs2 명령을 사용하여 GFS2 파일 시스템을 생성합니다. 파일 시스템은 활성화된 LVM 볼륨에 생성됩니다.

3.1.1. GFS2 mkfs 명령

mkfs.gfs2 명령을 실행하여 클러스터형 GFS2 파일 시스템을 생성하려면 다음 정보가 필요합니다.

- 잠금 프로토콜/모듈 이름(클러스터용 **lock_dlm**
- 클러스터 이름
- 저널 수(파일 시스템을 마운트할 수 있는 각 노드에 필요한 저널 1개)

참고

mkfs.gfs2 명령을 사용하여 GFS2 파일 시스템을 생성한 후에는 파일 시스템의 크기를 줄일 수 없습니다. 그러나 **gfs2_grow** 명령을 사용하여 기존 파일 시스템의 크기를 늘릴 수 있습니다.

클러스터형 GFS2 파일 시스템을 생성하는 형식은 다음과 같습니다. Red Hat은 GFS2를 단일 노드 파일 시스템으로 사용할 수 없습니다.

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

원하는 경우 **-t** 매개변수와 함께 **mkfs** 명령을 사용하여 **gfs2** 유형의 파일 시스템을 지정한 다음 GFS2 파일 시스템 옵션을 사용하여 GFS2 파일 시스템을 생성할 수 있습니다.

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

주의

ClusterName:FSName 매개변수를 잘못 지정하면 파일 시스템 또는 잠금 공간 손상이 발생할 수 있습니다.

ClusterName

GFS2 파일 시스템을 생성할 클러스터의 이름입니다.

FSName

파일 시스템 이름은 1~16자일 수 있습니다. 이 이름은 클러스터의 모든 **lock_dlm** 파일 시스템에 대해 고유해야 합니다.

NumberJournals

mkfs.gfs2 명령으로 생성할 저널 수를 지정합니다. 파일 시스템을 마운트하는 각 노드에 저널 1개가 필요합니다. GFS2 파일 시스템의 경우 파일 시스템을 확장하지 않고도 더 많은 저널을 추가할 수 있습니다.

BlockDevice

논리 또는 기타 블록 장치를 지정합니다.

다음 표에서는 **mkfs.gfs2** 명령 옵션(플래그 및 매개변수)에 대해 설명합니다.

표 3.1. 명령 옵션:mkfs.gfs2

플래그	매개변수	설명
-c	**메가바이트**	각 저널 할당량 변경 파일의 초기 크기를 **메가 바이트로** 설정합니다.
-D		디버깅 출력을 활성화합니다.
-h		도움말. 사용 가능한 옵션을 표시합니다.
-J	**메가바이트**	저널 크기(MB)를 지정합니다. 기본 저널 크기는 128MB입니다. 최소 크기는 8MB입니다. 더 큰 저널은 소규모 저널보다 더 많은 메모리를 사용하지만 성능이 향상됩니다.
-j	**숫자**	**mkfs.gfs2** 명령으로 생성할 저널 수를 지정합니다. 파일 시스템을 마운트하는 각 노드에 저널 1개가 필요합니다. 이 옵션을 지정하지 않으면 저널 1개가 생성됩니다. GFS2 파일 시스템의 경우 파일 시스템을 확장하지 않고도 나중에 저널을 추가할 수 있습니다.
-O		파일 시스템을 작성하기 전에 **mkfs.gfs2** 명령을 확인하지 않도록 합니다.
-p	**LockProtoName**	사용할 잠금 프로토콜의 이름을 지정합니다. 인식된 잠금 프로토콜은 다음과 같습니다. **lock_dlm** - 클러스터형 파일 시스템에 필요한 표준 잠금 모듈입니다. * **lock_nolock** - GFS2가 로컬 파일 시스템 역할을 할 때만 사용됩니다.
-q		quiet. 아무것도 표시하지 마십시오.

플래그	매개변수	설명
-r	**메가바이트**	리소스 그룹의 크기(MB)를 지정합니다. 최소 리소스 그룹 크기는 32MB입니다. 최대 리소스 그룹 크기는 2048MB입니다. 대규모 리소스 그룹 크기가 매우 큰 파일 시스템에서 성능을 향상시킬 수 있습니다. 이를 지정하지 않으면 **mkfs.gfs2** 는 파일 시스템의 크기에 따라 리소스 그룹 크기를 선택합니다. 평균 크기 파일 시스템에는 256 메가바이트 리소스 그룹이 있으며 더 큰 파일 시스템은 성능 향상을 위해 더 큰 RG를 갖습니다.
-t	**LockTableName**	**lock_dlm** 프로토콜을 사용할 때 lock_dlm 프로토콜을 지정하는 고유 식별자입니다. **lock_nolock** 프로토콜은 이 매개변수를 사용하지 않습니다. * 이 매개변수에는 다음과 같이 콜론(공백 없음)으로 구분된 두 개의 파트가 있습니다. **clustername:FSName**. * **cluster** name은 GFS2 파일 시스템을 생성할 클러스터의 이름입니다. 이 클러스터의 멤버만 이 파일 시스템을 사용할 수 있습니다. * **FSName**, 파일 시스템 이름은 길이가 1~16자일 수 있으며, 클러스터의 모든 파일 시스템에서 이름은 고유해야 합니다.
-V		명령 버전 정보를 표시합니다.

3.1.2. GFS2 파일 시스템 생성

다음 예제에서는 두 개의 GFS2 파일 시스템을 생성합니다. 이러한 두 파일 시스템의 경우 lock_dlm은 파일 시스템이 사용하는 잠금 프로토콜입니다. 두 파일 시스템 모두 **alpha** 라는 클러스터에서 사용할 수 있습니다.

첫 번째 파일 시스템의 경우 파일 시스템 이름은 **mydata1** 입니다. 여기에는 8개의 저널이 포함되어 **/dev/vg01/lvol0** 에 생성됩니다. 두 번째 파일 시스템의 경우 파일 시스템 이름은 **mydata2** 입니다. 여기에는 8개의 저널이 포함되어 있으며 **/dev/vg01/lvol1** 에서 생성됩니다.

```
# 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 파일 시스템 마운트

GFS2 파일 시스템을 마운트하려면 파일 시스템이 있어야 하며, 파일 시스템이 존재하는 볼륨을 활성화해야 하며 지원 클러스터링 및 잠금 시스템을 시작해야 합니다. 이러한 요구 사항이 충족되면 모든 Linux 파일 시스템과 마찬가지로 GFS2 파일 시스템을 마운트할 수 있습니다.

참고

마운트 명령으로 파일 시스템을 수동으로 마운트 하지 않고 프로덕션 환경에서 GFS2 파일 시스템을 관리하려면 시스템 종료 시 문제가 발생할 수 있으므로 항상 Pacemaker를 사용하여 GFS2 파일 시스템을 관리해야 합니다.

파일 ACL을 조작하려면 **-o acl** 마운트 옵션을 사용하여 파일 시스템을 마운트해야 합니다. **-o acl** 마운트 옵션 없이 파일 시스템을 마운트하면 사용자는 ACL(**getfacl**)을 볼 수 있지만 **setfacl**을 사용하여 설정할 수는 없습니다.

3.2.1. 옵션이 지정되지 않은 상태에서 GFS2 파일 시스템 마운트

이 예에서는 **/dev/vg01/lvol0** 의 GFS2 파일 시스템이 **/mygfs2** 디렉터리에 마운트됩니다.

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

3.2.2. 마운트 옵션을 지정하는 GFS2 파일 시스템 마운트

다음은 마운트 옵션을 지정하는 GFS2 파일 시스템을 마운트하기 위한 명령 형식입니다.

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

BlockDevice

GFS2 파일 시스템이 상주하는 블록 장치를 지정합니다.

MountPoint

GFS2 파일 시스템을 마운트해야 하는 디렉터리를 지정합니다.

-o 옵션 인수는 GFS2 특정 옵션 또는 허용 가능한 표준 Linux **마운트 -o** 옵션 또는 두 가지 모두의 조합으로 구성됩니다. 여러 **옵션** 매개변수는 쉼표로 구분되고 공백 없이 구분됩니다.

참고

mount 명령은 Linux 시스템 명령입니다. 이 섹션에 설명된 특정 옵션 사용 외에도 기타 표준, **마운트** 명령 옵션(예: **-r**)을 사용할 수 있습니다. 다른 Linux **마운트** 명령 옵션에 대한 자세한 내용은 Linux **마운트** 도움말 페이지를 참조하십시오.

다음 표에는 마운트 시 GFS2에 전달할 수 있는 사용 가능한 GFS2-specific **-o 옵션** 값이 설명되어 있습니다.

참고

이 표에는 로컬 파일 시스템에서만 사용되는 옵션에 대한 설명이 포함되어 있습니다. 그러나 Red Hat은 단일 노드 파일 시스템으로 GFS2 사용을 지원하지 않습니다. Red Hat은 클러스터 파일 시스템의 스냅샷을 마운트하기 위해 단일 노드 GFS2 파일 시스템을 계속 지원합니다(예: 백업 목적으로).

표 3.2. GFS2-Specific 마운트 옵션

옵션	설명
acl	파일 ACL을 조작할 수 있습니다. 파일 시스템이 **acl** 마운트 옵션 없이 마운트된 경우 사용자는 ACL(**getfacl**) 을 볼 수 있지만 해당 파일(**setfacl**을 사용하여) 설정할 수는 없습니다.
data=[ordered|writeback]	**data=ordered** 가 설정되면 트랜잭션이 수정한 사용자 데이터가 디스크에 플러시됩니다. 이렇게 하면 충돌 후 사용자가 파일에 초기화되지 않은 블록이 표시되지 않도록 해야 합니다. **data=writeback** 모드가 설정되면 사용자 데이터는 dirtied 후 언제든지 디스크에 기록됩니다. 이는 **순서가 지정된** 모드와 동일한 일관성 보장을 제공하지 않지만 일부 워크로드에서는 약간 더 빠르게 해야 합니다. 기본값은 **ordered** mode입니다.
ignore_local_fs **주의 사항:** 이 옵션은 GFS2 파일 시스템을 공유할 때 사용해서는 안 됩니다.	파일 시스템을 다중 호스트 파일 시스템으로 처리하도록 GFS2를 강제 적용합니다. 기본적으로 **lock_nolock** 을 사용하면 **localflocks** 플래그를 자동으로 켭니다.
localflocks **주의 사항:** 이 옵션은 GFS2 파일 시스템을 공유할 때 사용해서는 안 됩니다.	모든 flock 및 fcntl이 모든 flock과 fcntl을 수행할 수 있도록 GFS2에 대해 (가상 파일 시스템) 계층에 알립니다. **localflocks** 플래그는 **lock_nolock** 에 의해 자동으로 켜집니다.
lockproto=LockModuleName	사용자가 파일 시스템과 함께 사용할 잠금 프로토콜을 지정할 수 있습니다. **LockModuleName** 을 지정하지 않으면 잠금 프로토콜 이름을 파일 시스템 수퍼 블록에서 읽습니다.
locktable=LockTableName	사용자가 파일 시스템과 함께 사용할 잠금 테이블을 지정할 수 있습니다.
quota=[off/account/on]	파일 시스템의 할당량을 켜거나 끕니다. 할당량을 **계정** 상태에 있게 설정하면 파일 시스템에서 UID/GID 사용량 통계가 올바르게 유지 관리됩니다. 제한 및 warn 값은 무시됩니다. 기본값은 **OFF** 입니다.
errors=panic|withdraw	**errors=panic** 이 지정되면 파일 시스템 오류로 인해 커널 패닉이 발생합니다. Error **=withdraw** 가 지정되고 기본 동작으로 인해 파일 시스템 오류로 인해 시스템이 파일 시스템에서 인출되어 다음 재부팅까지 액세스할 수 없게 됩니다.
discard/nodiscard	이로 인해 GFS2에서 해제된 블록에 대해 "discard" I/O 요청을 생성하게 됩니다. 이는 적합한 하드웨어에서 씬 프로비저닝 및 유사한 체계를 구현하는 데 사용할 수 있습니다.

옵션	설명
barrier/nobarrier	저널을 플러시할 때 GFS2에서 I/O 장벽을 보냅니다. 기본값 **은 입니다.** 기본 장치가 I/O 장벽을 지원하지 않는 경우 이 옵션이 자동으로 **꺼집니다**. GFS2에서 I/O 장벽을 사용하는 것은 블록 장치가 쓰기 캐시 콘텐츠를 손실할 수 없도록 설계되지 않는 한 항상 사용하는 것이 좋습니다(예: UPS에 있거나 쓰기 캐시가 없는 경우).
quota_quantum=*secs*	할당량 정보에 변경 사항이 할당량 파일에 기록되기 전에 한 노드에 있을 수 있는 시간(초)을 설정합니다. 이 매개 변수를 설정하는 것이 좋습니다. 값은 0보다 큰 정수 초입니다. 기본값은 60초입니다. 더 짧은 설정으로 인해 지연 할당량 정보 업데이트 속도가 빨라지고 사람이 할당량을 초과할 가능성이 줄어듭니다. 더 긴 설정에서는 할당량과 관련된 파일 시스템 작업을 더 빠르고 효율적으로 수행할 수 있습니다.
statfs_quantum=*secs*	**statfs_quantum** 을 0으로 설정하는 것이 느린 버전의 **statfs** 를 설정하는 데 선호되는 방법입니다. 기본값은 **statfs** 변경이 마스터 **statfs** 파일과 동기화되기 전에 최대 기간을 설정하는 30유입니다. 보다 빠르고 정확한 **statfs** 값을 허용하거나 더 정확한 값을 더 느릴 수 있도록 조정할 수 있습니다. 이 옵션을 0으로 설정하면 **statfs** 는 항상 true 값을 보고합니다.
statfs_percent=*value*	시간이 만료되지 않은 경우에도 마스터 **statfs** 파일과 다시 동기화되기 전에 **statfs** 정보의 최대 백분율 변경으로 바인딩됩니다. **statfs_quantum** 설정이 0이면 이 설정이 무시됩니다.

3.2.3. GFS2 파일 시스템 마운트 해제

Pacemaker를 통해 자동으로 마운트되지 않고 수동으로 마운트된 GFS2 파일 시스템은 시스템 종료 시 파일 시스템을 마운트 해제할 때 시스템에서 인식되지 않습니다. 결과적으로 GFS2 리소스 에이전트에서 GFS2 파일 시스템을 마운트 해제하지 않습니다. GFS2 리소스 에이전트가 종료되면 표준 종료 프로세스에서 클러스터 인프라를 포함한 나머지 사용자 프로세스를 모두 종료하고 파일 시스템을 마운트 해제하려고 시도합니다. 클러스터 인프라 없이는 이 마운트 해제에 실패하고 시스템이 중단됩니다.

GFS2 파일 시스템을 마운트 해제할 때 시스템이 정지되지 않도록 하려면 다음 중 하나를 수행해야 합니다.

- 항상 Pacemaker를 사용하여 GFS2 파일 시스템을 관리합니다.
- **mount** 명령을 사용하여 수동으로 GFS2 파일 시스템을 마운트한 경우 시스템을 재부팅하거나 종료하기 전에 **umount** 명령을 사용하여 파일 시스템을 수동으로 마운트 해제합니다.

이러한 상황에서 시스템 종료 중에 파일 시스템을 마운트 해제하는 동안 정지한 경우 하드웨어 재부팅을 수행합니다. 시스템 종료 프로세스 초기에 파일 시스템이 동기화되므로 데이터가 손실되지 않습니다.

umount 명령을 사용하여 모든 Linux 파일 시스템과 동일한 방식으로 GFS2 파일 시스템을 마운트 해제할 수 있습니다.

참고

umount 명령은 Linux 시스템 명령입니다. 이 명령에 대한 정보는 Linux **umount** 명령 도움말 페이지에서 확인할 수 있습니다.

사용법

```
umount MountPoint
```

MountPoint

GFS2 파일 시스템이 현재 마운트되어 있는 디렉터리를 지정합니다.

3.3. GFS2 파일 시스템 백업

파일 시스템의 크기에 관계없이 긴급한 경우 GFS2 파일 시스템을 정기적으로 백업해야 합니다. 많은 시스템 관리자는 RAID, 다중 경로, 미러링, 스냅샷 및 기타 형태의 중복성에 의해 보호되고 있기 때문에 안전하게 보호된다고 생각하지만, 충분히 안전한 것은 없습니다.

노드 또는 노드 세트를 백업하는 프로세스에서 일반적으로 전체 파일 시스템을 순서대로 읽는 작업이므로 백업을 생성하는 것이 문제가 될 수 있습니다. 단일 노드에서 이 작업을 수행하면 클러스터의 다른 노드가 잠금을 요청할 때까지 해당 노드는 캐시의 모든 정보를 유지합니다. 클러스터가 작동하는 동안 이러한 유형의 백업 프로그램을 실행하면 성능에 부정적인 영향을 미칩니다.

백업이 완료되면 캐시를 삭제하면 다른 노드에서 클러스터 잠금/캐너스의 소유권을 되찾는 데 필요한 시간이 줄어듭니다. 그러나 다른 노드에서 백업 프로세스가 시작되기 전에 캐싱된 데이터 캐싱이 중지되므로 이 방법은 여전히 적합하지 않습니다. 백업이 완료된 후 다음 명령을 사용하여 캐시를 삭제할 수 있습니다.

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

클러스터의 각 노드가 자체 파일을 백업하여 작업이 노드 간에 분할되도록 하는 것이 더 빠릅니다. 노드별 디렉터리에서 **rsync** 명령을 사용하는 스크립트로 이 작업을 수행할 수 있습니다.

Red Hat은 SAN에서 하드웨어 스냅샷을 작성하여 다른 시스템에 스냅샷을 제공하고 여기에서 지원하는 것이 좋습니다. 백업 시스템은 **-o lockproto=lock_nolock** 을 사용하여 스냅샷을 마운트해야 합니다.

3.4. GFS2 파일 시스템에서 작업 일시 중지

dmsetup suspend 명령을 사용하여 파일 시스템에 쓰기 활동을 일시 중단할 수 있습니다. 쓰기 활동을 일시 중지하면 하드웨어 기반 장치 스냅샷을 사용하여 파일 시스템을 일관된 상태로 캡처할 수 있습니다. **dmsetup resume** 명령은 중지를 종료합니다.

GFS2 파일 시스템의 활동을 일시 중단하는 명령 형식은 다음과 같습니다.

```
dmsetup suspend MountPoint
```

이 예제에서는 파일 시스템 **/mygfs2** 에 대한 쓰기를 일시 중지합니다.

```
# dmsetup suspend /mygfs2
```

GFS2 파일 시스템에서 작업 중단을 종료할 명령의 형식은 다음과 같습니다.

```
dmsetup resume MountPoint
```

-

이 예제는 파일 시스템 **/mygfs2** 에 대한 쓰기 중단을 종료합니다.

```
# dmsetup resume /mygfs2
```

3.5. GFS2 파일 시스템 확장

gfs2_grow 명령은 파일 시스템이 있는 장치가 확장된 후 GFS2 파일 시스템을 확장하는 데 사용됩니다. 기존 GFS2 파일 시스템에서 **gfs2_grow** 명령을 실행하면 파일 시스템의 현재 끝과 새로 초기화된 GFS2 파일 시스템 확장자로 장치 종료 사이에 모든 여유 공간이 채워집니다. 클러스터의 모든 노드는 추가된 추가 스토리지 공간을 사용할 수 있습니다.

참고

GFS2 파일 시스템의 크기를 줄일 수 없습니다.

마운트된 파일 시스템에서 **gfs2_grow** 명령을 실행해야 합니다. 다음 절차에서는 **/mnt/gfs2** 의 마운트 지점인 **shared_vg/shared_lv1** 에 마운트된 클러스터의 GFS2 파일 시스템의 크기를 늘립니다.

절차

1. 파일 시스템에서 데이터 백업을 수행합니다.
2. 파일 시스템에서 확장할 논리 볼륨을 모르는 경우 **df *mountpoint*** 명령을 실행하여 확인할 수 있습니다. 그러면 장치 이름이 다음 형식으로 표시됩니다.
   /dev/mapper/*vg-lv*

   예를 들어 장치 이름 **/dev/mapper/shared_vg-shared_lv1** 은 논리 볼륨이 **shared_vg/shared_lv1** 임을 나타냅니다.
3. 클러스터의 한 노드에서 **lvextend** 명령을 사용하여 기본 클러스터 볼륨을 확장합니다. RHEL 8.0을 실행 중인 경우 **--lockopt skiplv** 옵션을 사용하여 일반 논리 볼륨 잠금을 재정의합니다. RHEL 8.1 이상을 실행하는 시스템에서는 필요하지 않습니다.
   RHEL 8.1 이상에서는 다음 명령을 사용합니다.

   ```
   # 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 8.0의 경우 다음 명령을 사용합니다.

   ```
   # 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 8.0을 실행하는 경우 클러스터의 모든 추가 노드에서 논리 볼륨을 새로 고침하여 해당 노드의 활성 논리 볼륨을 업데이트합니다. 논리 볼륨이 확장될 때 단계가 자동화되므로 RHEL 8.1을 실행하는 시스템에서 이 단계가 필요하지 않습니다.

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

5. 클러스터 노드 1개로, GFS2 파일 시스템의 크기를 늘립니다. 논리 볼륨이 모든 노드에서 새로 고쳐지지 않은 경우 파일 시스템을 확장하지 마십시오. 그렇지 않으면 파일 시스템 데이터를 클러스터 전체에서 사용할 수 없게 됩니다.

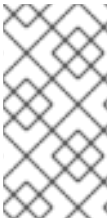
```
# gfs2_grow /mnt/gfs2
FS: Mount point:             /mnt/gfs2
FS: Device:                  /dev/mapper/shared_vg-shared_lv1
FS: Size:                    1310719 (0x13ffff)
DEV: Length:                 1572864 (0x180000)
The file system will grow by 1024MB.
gfs2_grow complete.
```

6. 모든 노드에서 **df** 명령을 실행하여 파일 시스템에서 새 공간을 사용할 수 있는지 확인합니다. 모든 노드의 **df** 명령이 동일한 파일 시스템 크기를 표시하는 데 최대 30초가 걸릴 수 있습니다.

```
# 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 파일 시스템에 저널 추가

GFS2에는 파일 시스템을 마운트해야 하는 클러스터의 각 노드에 대해 저널이 1개 있어야 합니다. 클러스터에 노드를 추가하는 경우 **gfs2_jadd** 명령을 사용하여 GFS2 파일 시스템에 저널을 추가할 수 있습니다. 기본 논리 볼륨을 확장하지 않고 언제든지 GFS2 파일 시스템에 저널을 동적으로 추가할 수 있습니다. **gfs2_jadd** 명령은 마운트된 파일 시스템에서 실행해야 하지만 클러스터의 하나의 노드에서만 실행해야 합니다. 다른 모든 노드는 확장이 발생했음을 인식합니다.



참고

GFS2 파일 시스템이 가득 차면 파일 시스템이 포함된 논리 볼륨이 확장되어 파일 시스템보다 큰 경우에도 **gfs2_jadd** 명령이 실패합니다. 이는 GFS2 파일 시스템에서 저널이 포함된 메타데이터가 아닌 일반 파일이므로 기본 논리 볼륨을 확장하면 저널 공간을 제공하지 않습니다.

GFS2 파일 시스템에 저널을 추가하기 전에 다음 예와 같이 현재 **gfs2_edit -p jindex** 명령을 사용하여 현재 GFS2 파일 시스템에 포함된 저널 수를 확인할 수 있습니다.

```
# 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 파일 시스템에 저널을 추가하는 기본 명령의 형식은 다음과 같습니다.

```
gfs2_jadd -j Number MountPoint
```

숫자

추가할 새 저널 수를 지정합니다.

MountPoint

GFS2 파일 시스템이 마운트된 디렉터리를 지정합니다.

이 예에서는 **/mygfs2** 디렉터리의 파일 시스템에 하나의 저널이 추가됩니다.

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

4장. GFS2 할당량 관리

파일 시스템 할당량은 사용자 또는 그룹이 사용할 수 있는 파일 시스템 공간의 양을 제한하는 데 사용됩니다. 사용자 또는 그룹은 설정할 때까지 할당량 제한이 없습니다. **quota=on** 또는 **quota=account** 옵션을 사용하여 GFS2 파일 시스템을 마운트하면, GFS2는 제한이 없는 경우에도 각 사용자 및 그룹에서 사용하는 공간을 추적합니다. GFS2는 트랜잭션 방식으로 할당량 정보를 업데이트하므로 시스템 충돌 시 할당량 사용을 재구성할 필요가 없습니다.

성능 저하를 방지하려면 GFS2 노드는 주기적으로 업데이트를 할당량 파일에 동기화합니다. 퍼지 할당량 회계를 통해 사용자 또는 그룹이 설정된 제한을 약간 초과할 수 있습니다. 이를 최소화하기 위해 GFS2는 하드 할당량 제한에 접근할 때 동기화 기간을 동적으로 줄입니다.

참고

GFS2에서는 표준 Linux 할당량 기능을 지원합니다. 이를 사용하려면 **quota** RPM을 설치해야 합니다. 이는 GFS2에서 할당량을 관리하는 데 선호되는 방식이며 할당량을 사용하여 GFS2의 모든 신규 배포에 사용해야 합니다. 이 섹션에서는 이러한 기능을 사용하여 GFS2 할당량 관리를 문서화합니다.

디스크 할당량에 대한 자세한 내용은 다음 명령의 **도움말** 페이지를 참조하십시오.

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

4.1. GFS2 디스크 할당량 구성

GFS2 파일 시스템에 대한 디스크 할당량을 구현하려면 수행해야 하는 세 가지 단계가 있습니다.

디스크 할당량을 구현하기 위한 단계는 다음과 같습니다.

1. 실행 또는 계정 모드에서 할당량을 설정합니다.
2. 현재 블록 사용량 정보로 할당량 데이터베이스 파일을 초기화합니다.
3. 할당량 정책을 할당합니다. (회계 모드에서는 이러한 정책이 적용되지 않습니다.)

이러한 각 단계에 대해서는 다음 섹션에서 자세히 설명합니다.

4.1.1. 적용 또는 계정 모드에서 할당량 설정

GFS2 파일 시스템에서 할당량은 기본적으로 비활성화되어 있습니다. 파일 시스템의 할당량을 활성화하려면 **quota=on** 옵션이 지정된 파일 시스템을 마운트합니다.

할당량이 활성화된 파일 시스템을 마운트하려면 클러스터에 GFS2 파일 시스템 리소스를 생성할 때 **options** 인수에 **quota=on** 을 지정합니다. 예를 들어 다음 명령은 생성되는 GFS2 **Filesystem** 리소스가 할당량을 활성화하여 마운트되도록 지정합니다.

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

제한 및 경고 값을 적용하지 않고 디스크 사용량을 추적하고 모든 사용자 및 그룹에 대한 할당량 계정을 유지 관리할 수 있습니다. 이렇게 하려면 **quota=account** 옵션이 지정된 파일 시스템을 마운트합니다.

할당량이 비활성화된 상태로 파일 시스템을 마운트하려면 클러스터에 GFS2 파일 시스템 리소스를 생성할 때 **options** 인수에 **quota=off** 를 지정합니다.

4.1.2. 할당량 데이터베이스 파일 생성

할당량이 활성화된 각 파일 시스템이 마운트되면 시스템에서 디스크 할당량을 사용할 수 있습니다. 그러나 파일 시스템 자체는 아직 할당량을 지원할 준비가 되지 않았습니다. 다음 단계는 **quotacheck** 명령을 실행하는 것입니다.

quotacheck 명령은 할당량 사용 파일 시스템을 검사하고 파일 시스템당 현재 디스크 사용량 테이블을 작성합니다. 그러면 테이블이 디스크 사용량의 운영 체제 복사본을 업데이트하는 데 사용됩니다. 또한 파일 시스템의 디스크 할당량 파일이 업데이트됩니다.

파일 시스템에 할당량 파일을 생성하려면 **-u** 및 **quotacheck** 명령의 **-g** 옵션을 사용합니다. 이러한 두 옵션은 모두 초기화하려면 사용자 및 그룹 할당량을 지정해야 합니다. 예를 들어 **/home** 파일 시스템에 할당량을 활성화하면 **/home** 디렉터리에 파일을 생성합니다.

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

4.1.3. 사용자당 할당량 할당

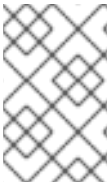
마지막 단계는 **edquota** 명령을 사용하여 디스크 할당량을 할당하는 것입니다. 계정 모드에서 파일 시스템을 마운트한 경우 **할당량** 을 적용하지 않습니다.

사용자의 할당량을 쉘 프롬프트에서 root로 구성하려면 다음 명령을 실행합니다.

```
# edquota username
```

할당량이 필요한 각 사용자에 대해 이 단계를 수행합니다. 예를 들어 아래 예제의 **/home** 파티션 (예:**/dev/VolGroup00/LogVol** 02)에 할당량이 활성화되어 있고 **edquota testuser** 명령이 실행되면 시스템의 기본값으로 구성된 편집기에 다음이 표시됩니다.

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



참고

EDITOR 환경 변수에서 정의한 텍스트 편집기는 **edquota** 에서 사용됩니다. 편집기를 변경하려면 **~/.bash_profile** 파일의 **EDITOR** 환경 변수를 선택한 편집기의 전체 경로로 설정합니다.

첫 번째 열은 할당량이 활성화된 파일 시스템의 이름입니다. 두 번째 열에는 현재 사용자가 사용 중인 블록 수가 표시됩니다. 다음 두 열은 파일 시스템에서 사용자에 대한 소프트 및 하드 블록 제한을 설정하는 데 사용됩니다.

소프트 블록 제한은 사용할 수 있는 최대 디스크 공간을 정의합니다.

하드 블록 제한은 사용자 또는 그룹이 사용할 수 있는 디스크 공간의 절대 최대 크기입니다. 이 제한에 도달하면 추가 디스크 공간을 사용할 수 없습니다.

GFS2 파일 시스템은 inode에 대한 할당량을 유지하지 않으므로 이러한 열은 GFS2 파일 시스템에는 적용되지 않으며 비어 있습니다.

값 중 하나를 0으로 설정하면 해당 제한이 설정되지 않습니다. 텍스트 편집기에서 제한을 변경합니다. 예를 들면 다음과 같습니다.

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

사용자 할당량이 설정되었는지 확인하려면 다음 명령을 사용하십시오.

```
# quota testuser
```

setquota 명령을 사용하여 명령줄에서 할당량을 설정할 수도 있습니다. **setquota** 명령에 대한 자세한 내용은 **setquota**(8) 매뉴얼 페이지를 참조하십시오.

4.1.4. 그룹당 할당량 할당

할당량은 그룹별로 할당할 수도 있습니다. 계정 모드에서 파일 시스템을 마운트한 경우 (**account=on** 옵션을 지정한 상태에서) 할당량이 적용되지 않습니다.

devel 그룹(그룹 할당량을 설정하기 전에 그룹이 존재해야 함)에 대한 그룹 할당량을 설정하려면 다음 명령을 사용합니다.

```
# edquota -g devel
```

이 명령은 텍스트 편집기에서 그룹의 기존 할당량을 표시합니다.

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

GFS2 파일 시스템은 inode에 대한 할당량을 유지하지 않으므로 이러한 열은 GFS2 파일 시스템에는 적용되지 않으며 비어 있습니다. 제한을 수정한 다음 파일을 저장합니다.

그룹 할당량이 설정되었는지 확인하려면 다음 명령을 사용합니다.

```
$ quota -g devel
```

4.2. GFS2 디스크 할당량 관리

할당량을 구현하는 경우, 주로 할당량을 초과하는지 확인하고 할당량이 올바른지 확인하기 위해 몇 가지 유지보수가 필요합니다.

사용자가 할당량을 반복적으로 초과하거나 소프트 제한에 일관되게 도달하는 경우 시스템 관리자는 사용자 유형 및 디스크 공간이 작업에 미치는 영향에 따라 몇 가지 선택을 할 수 있습니다. 관리자는 사용자가 디스크 공간을 적게 사용하는 방법을 결정하거나 사용자의 디스크 할당량을 늘리는 데 도움이 될 수 있습니다.

repquota 유틸리티를 실행하여 디스크 사용량 보고서를 생성할 수 있습니다. 예를 들어, **repquota /home** 명령은 다음 출력을 생성합니다.

```
*** 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
```

모든(옵션 **-a**) 할당량 지원 파일 시스템에 대한 디스크 사용량 보고서를 보려면 명령을 사용합니다.

```
# repquota -a
```

각 사용자가 블록 제한을 초과했는지 여부를 확인하는 빠른 방법입니다. **블록 소프트 제한을 초과하면 출력에 + 가 첫 번째 - 대신 표시됩니다. 두 번째 - 는 inode 제한을 나타내지만, GFS2 파일 시스템은 inode 제한을 지원하지 않으므로 문자가 - 로 유지됩니다. GFS2 파일 시스템은 유예 기간을 지원하지 않으므로 grace 열은 빈 상태로 유지됩니다.**

기본 파일 시스템에 관계없이 NFS에서는 repquota 명령이 지원되지 않습니다.

4.3. QUOTACHECK 명령을 사용하여 GFS2 디스크 할당량을 정확하게 유지합니다.

할당량을 비활성화한 상태로 실행 중인 경우 quotacheck 명령을 실행하여 할당량 파일을 생성, 확인 및 복구해야 합니다. 또한 시스템 충돌 후 파일 시스템을 완전히 마운트 해제하지 않을 수 있으므로 할당량 파일이 정확하지 않다고 생각되면 quotacheck 명령을 실행할 수 있습니다.

quotacheck 명령에 대한 자세한 내용은 quotacheck(8) 매뉴얼 페이지를 참조하십시오.

참고

디스크 활동이 계산된 할당량 값에 영향을 미칠 수 있으므로 모든 노드에서 파일 시스템이 상대적으로 유휴 상태인 경우 quotacheck 를 실행합니다.

4.4. QUOTASYNC 명령을 사용하여 할당량 동기화

GFS2는 모든 할당량 정보를 디스크의 자체 내부 파일에 저장합니다. GFS2 노드는 모든 파일 시스템 쓰기에 대해 이 할당량 파일을 업데이트하지 않습니다. 대신 기본적으로 60초마다 할당량 파일을 업데이트합니다. 이는 할당량 파일에 쓰는 노드 간 경합을 방지하기 위해 필요합니다. 이로 인해 성능이 저하됩니다.

사용자 또는 그룹이 할당량 제한에 접근하므로, GFS2는 할당량 파일 업데이트 간의 시간을 동적으로 줄여 제한이 초과되지 않도록 합니다. 할당량 동기화 사이의 정상적인 시간 기간은 조정 가능한 매개변수

인 quota_quantum 입니다. 마운트 옵션을 지정하는 GFS2 파일 시스템 마운트의 "GFS2-Specific Mount Options" 테이블에 설명된 대로 quota_quantum= 마운트 옵션을 사용하여 이 값을 60초의 기본 값에서 변경할 수 있습니다.

quota_quantum 매개 변수는 각 노드에 설정되어야 하며 파일 시스템이 마운트될 때마다 설정해야 합니다. quota_quantum 매개변수는 마운트 해제 시 지속되지 않습니다. mount -o reinstall 를 사용하여 quota_quantum 값을 업데이트할 수 있습니다.

quotasync 명령을 사용하여 노드의 할당량 정보를 GFS2에서 수행하는 자동 업데이트 간의 디스크상의 할당량 파일과 동기화할 수 있습니다. 사용량 동기화 할당량 정보

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

u

사용자 할당량 파일을 동기화합니다.

g

그룹 할당량 파일 동기화

a

현재 할당량을 활성화하고 지원하는 모든 파일 시스템을 동기화합니다. a가 없으면 파일 시스템 마운트 지점을 지정해야 합니다.

mountpoint

작업이 적용되는 GFS2 파일 시스템을 지정합니다.

quota-quantum 마운트 옵션을 지정하여 동기화 간 시간을 조정할 수 있습니다.

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

MountPoint

작업이 적용되는 GFS2 파일 시스템을 지정합니다.

보안 설정

GFS2의 일반 할당량 파일 동기화 간 새 기간을 지정합니다. 값이 작은 경우 경합이 증가하여 성능이 저하될 수 있습니다.

다음 예제에서는 파일 시스템 /mnt/mygfs2 의 노드에서 실행된 노드에서 캐시된 모든 더티 할당량을 동기화합니다.

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

다음 예제에서는 논리 볼륨 /dev/volgroup/logical_volume 에서 해당 파일 시스템을 다시 마운트할 때 파일 시스템 /mnt/mygfs2 의 경우 일반 quota-file이 1시간(3600초)로 업데이트합니다.

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

5장. GFS2 파일 시스템 복구

파일 시스템을 마운트하여 노드가 실패하면 파일 시스템 저널링을 사용하면 빠른 복구를 수행할 수 있습니다. 그러나 스토리지 장치의 전원이 손실되거나 물리적으로 연결이 끊어지면 파일 시스템 손상이 발생할 수 있습니다. (**journaling**은 스토리지 하위 시스템 실패에서 복구하는 데 사용할 수 없습니다.) 이러한 유형의 손상이 발생하면 **fsck.gfs2** 명령을 사용하여 **GFS2** 파일 시스템을 복구할 수 있습니다.

중요

fsck.gfs2 명령은 모든 노드에서 마운트 해제된 파일 시스템에서만 실행해야 합니다. 파일 시스템을 **Pacemaker** 클러스터 리소스로 관리 중인 경우 파일 시스템 리소스를 비활성화하여 파일 시스템을 마운트 해제할 수 있습니다. **fsck.gfs2** 명령을 실행한 후 파일 시스템 리소스를 다시 활성화합니다. **pcs resource disable** 의 **--wait** 옵션으로 지정된 *타임아웃* 값은 값을 초 단위로 나타냅니다.

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

부트시 **GFS2** 파일 시스템에서 **fsck.gfs2** 명령이 실행되지 않도록 클러스터에 **GFS2** 파일 시스템 리소스를 생성할 때 **options** 인수의 **run_fsck** 매개 변수를 설정할 수 있습니다. **"run_fsck=no"** 를 지정하면 **fsck** 명령을 실행하지 않아야 합니다.

5.1. FSCK.GFS2를 실행하는 데 필요한 메모리 확인

fsck.gfs2 명령을 실행하려면 운영 체제 및 커널에 사용된 메모리 이상으로 위의 시스템 메모리가 필요할 수 있습니다. 특히 대규모 파일 시스템에는 이 명령을 실행하기 위해 추가 메모리가 필요할 수 있습니다.

다음 표는 **1TB, 10TB, 100TB**의 블록 크기가 **4K**인 **GFS2** 파일 시스템에서 **fsck.gfs2** 파일 시스템을 실행하는 데 필요할 수 있는 대략적인 메모리 값을 보여줍니다.

GFS2 파일 시스템 크기	fsck.gfs2를 실행하는 데 필요한 대략적인 메모리
1TB	0.16GB
10TB	1.6GB
100TB	16GB

파일 시스템의 더 작은 블록 크기는 더 많은 양의 메모리가 필요합니다. 예를 들어 블록 크기가 1K인 GFS2 파일 시스템에는 이 표에 표시된 메모리 양이 4배가 필요합니다.

5.2. GFS2 파일 시스템 복구

GFS2 파일 시스템을 복구하기 위한 fsck.gfs2 명령의 형식은 다음과 같습니다.

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

-y

-y 플래그는 모든 질문에 yes 로 응답합니다. -y 플래그를 지정하면 fsck.gfs2 명령을 실행하면 변경하기 전에 답변을 입력하라는 메시지가 표시되지 않습니다.

BlockDevice

GFS2 파일 시스템이 상주하는 블록 장치를 지정합니다.

이 예에서는 블록 장치 /dev/testvg/testlv 에 상주하는 GFS2 파일 시스템이 복구됩니다. 복구에 대한 모든 쿼리는 자동으로 yes 로 답변됩니다.

```
# 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
```

6장. GFS2 성능 개선

GFS2 구성에는 파일 시스템 성능을 개선하기 위해 분석할 수 있는 많은 측면이 있습니다.

High Availability Add-On 및 Red Hat Global File System 2(GFS2)를 사용하여 Red Hat Enterprise Linux 클러스터 배포 및 업그레이드에 대한 일반적인 권장 사항은 Red Hat Customer Portal의 Red Hat Enterprise Linux Cluster, High Availability 및 GFS 배포 모범 사례를 참조하십시오.

6.1. GFS2 파일 시스템 조각 모음

Red Hat Enterprise Linux에서 GFS2에 대한 조각 모음 툴은 없지만, filefrag 툴과 식별하여 개별 파일을 조각 모음하고, 임시 파일로 복사한 다음, 임시 파일로 복사하고, 원본을 교체하도록 임시 파일의 이름을 바꾸면 개별 파일을 조각 모음할 수 있습니다.

6.2. GFS2 노드 잠금

GFS2 파일 시스템에서 최상의 성능을 얻으려면 작업의 기본 이론을 이해하는 것이 중요합니다. 단일 노드 파일 시스템은 캐시와 함께 구현되며, 자주 요청된 데이터를 사용할 때 디스크 액세스 대기 시간을 제거할 수 있습니다. Linux에서 페이지 캐시(및 이전에 버퍼 캐시)는 이 캐싱 기능을 제공합니다.

GFS2에서 각 노드에는 디스크의 일부 데이터가 포함될 수 있는 자체 페이지 캐시가 있습니다. GFS2에서는 *glocks* (pronounced gee-locks)라는 잠금 메커니즘을 사용하여 노드 간 캐시의 무결성을 유지합니다. glock 하위 시스템은 DLM(*Distributed Lock Manager*)을 기본 통신 계층으로 사용하여 구현된 캐시 관리 기능을 제공합니다.

glocks는 노드별로 캐시 보호를 제공하므로 캐싱 계층을 제어하는 데 사용되는 inode당 하나의 잠금이 있습니다. 공유 모드(DLM 잠금 모드)에서 해당 glock이 부여된 경우: 그런 다음 해당 glock 아래의 데이터는 동시에 하나 이상의 노드에 캐시되므로 모든 노드가 데이터에 대한 로컬 액세스 권한을 가질 수 있습니다.

glock이 전용 모드 (DLM 잠금 모드)로 부여 된 경우: 그런 다음 단일 노드만 해당 glock 아래에 데이터를 캐시할 수 있습니다. 이 모드는 데이터를 수정하는 모든 작업에서 사용합니다(예: 쓰기 시스템 호출).

다른 노드에서 즉시 부여할 수 없는 glock을 요청하면 DLM은 현재 새 요청을 차단하여 잠금을 삭제하도록 요청하는 노드 또는 노드에 메시지를 보냅니다. glock을 삭제하는 것은 (대부분의 파일 시스템 작업의 기준에 따라) 긴 프로세스가 될 수 있습니다. 공유 glock을 삭제하려면 캐시를 무효화해야 하며, 이는 상대적으로 빠르고 캐시된 데이터의 양에 비례합니다.

배타적인 glock을 삭제하려면 로그 플러시가 필요하고 변경된 데이터를 디스크에 다시 작성한 다음 공유 glock에 따라 무효화가 발생합니다.

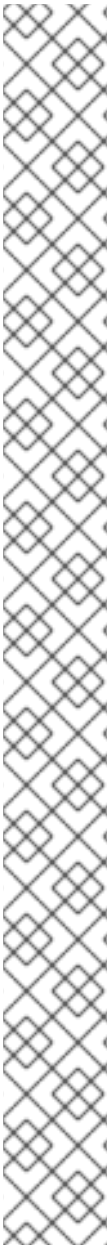
단일 노드 파일 시스템과 GFS2의 차이점은 단일 노드 파일 시스템에 단일 캐시가 있고 GFS2에는 각 노드에 별도의 캐시가 있다는 것입니다. 두 경우 모두 캐시된 데이터에 액세스하기 위한 대기 시간은 유사한 수준의 크기이지만 캐시되지 않은 데이터에 액세스하기 위한 대기 시간은 다른 노드가 이전에 동일한 데이터를 캐시한 경우 GFS2에서 훨씬 더 큽니다.

읽기(buffered), stat 및 read dir 과 같은 작업에는 공유 glock만 필요합니다. 쓰기 (buffered), EgressIP ,rmdir, unlink 와 같은 작업에는 배타적 glock이 필요합니다. 직접 I/O 읽기/쓰기 작업에서는 할당이 없는 경우 지연된 glock 또는 쓰기에 할당이 필요한 경우 독점 glock(즉, 파일 확장 또는 누락)이 필요합니다.

이 경우 다음과 같은 두 가지 주요 성능 고려 사항이 있습니다. 첫 번째 읽기 전용 작업은 모든 노드에서 독립적으로 실행될 수 있으므로 클러스터에서 매우 잘 병렬화됩니다. 두 번째, 배타적인 glock을 필요로 하는 작업에서는 동일한 inode에 대한 액세스를 위해 여러 노드가 제한되는 경우 성능을 줄일 수 있습니다. 따라서 각 노드에서 작업 집합에 대한 고려 사항은 GFS2 파일 시스템 백업에 설명된 대로 파일 시스템 백업을 수행하는 경우와 같이 GFS2 파일 시스템 성능에 중요한 요소입니다.

이로 인해 가능한 경우 애플리케이션이 이를 허용하는 noatime 또는 nodiratime 마운트 옵션을 최대한 사용하여 noatime 또는 nodiratime 마운트 옵션을 사용하는 것이 좋습니다. 이 경우 atime timestamp를 업데이트하기 위해 배타적 잠금이 필요하지 않습니다.

working set 또는 캐싱 효율성에 대한 우려가 있는 사용자를 위해 GFS2 파일 시스템의 성능을 모니터링할 수 있는 툴을 제공합니다. Performance Co-octets 및 GFS2 추적 지점.



참고

GFS2의 캐싱이 구현되는 방식으로 인해 다음 중 하나를 수행할 때 최상의 성능을 얻을 수 있습니다.

- inode는 모든 노드에서 읽기 전용 방식으로 사용됩니다.
- inode는 단일 노드에서만 작성되거나 수정됩니다.

파일 생성 및 삭제 중에 디렉터리에서 항목을 삽입 및 제거하는 것은 디렉터리 inode에 쓰는 것으로 계산됩니다.

이 규칙이 비교적 자주 손상되어 있음을 알리는 규칙이 중단될 수 있습니다. 이 규칙을 너무 자주 무시하면 심각한 성능 저하가 발생합니다.

읽기/쓰기 매핑만 있고 읽기/쓰기 매핑만 있는 GFS2의 파일을 mmap()하는 경우 이 파일은 읽기로만 계산됩니다.

noatime mount 매개변수를 설정하지 않으면 읽기에서도 파일 타임스탬프를 업데이트하기 위한 쓰기가 발생합니다. 특정 요구 사항이 없는 한 모든 GFS2 사용자가 no atime으로 마운트해야 합니다.

6.3. POSIX 잠금 문제

Posix locking를 사용할 때는 다음을 고려해야 합니다.

- Flock을 사용하면 Posix 잠금을 사용하는 것보다 더 빠르게 처리 할 수 있습니다.
- GFS2에서 Posix 잠금을 사용하는 프로그램은 클러스터형 환경에서 GETLK 함수를 사용하지 않아야 하며, 클러스터의 다른 노드에 프로세스 ID가 있을 수 있습니다.

6.4. GFS2로 성능 튜닝

일반적으로 어려운 애플리케이션이 상당한 성능 이점을 얻기 위해 데이터를 저장하는 방법을 변경할

수 있습니다.

어려운 애플리케이션의 일반적인 예는 이메일 서버입니다. 각 사용자(**mbox**) 또는 각 사용자에 대한 파일이 포함된 **spool** 디렉터리에 각 메시지에 대한 파일이 포함된 디렉터리(**maildir**)와 함께 배치되는 경우가 많습니다. **commands**를 통해 요청이 도착하면 각 사용자에게 특정 노드에 선호도를 부여하기 위한 최적의 배열이 있습니다. 이렇게 하면 이메일 메시지를 보고 삭제하라는 요청이 하나의 노드의 캐시에서 제공되는 경향이 있습니다. 해당 노드가 실패하면 다른 노드에서 세션을 다시 시작할 수 있습니다.

메일이 **SMTP**를 통해 도착하면 기본적으로 특정 사용자의 이메일을 특정 노드에 전달할 수 있도록 개별 노드를 다시 설정할 수 있습니다. 기본 노드가 작동하지 않으면 수신 노드에서 사용자의 메일 스풀에 메시지를 직접 저장할 수 있습니다. 이 설계는 일반적인 사례에서 하나의 노드에 캐시된 특정 파일 집합을 유지하기 위한 것으로, 노드 장애 발생 시 직접 액세스가 허용됩니다.

이 설정을 사용하면 **GFS2**의 페이지 캐시를 최대한 활용할 수 있으며 **redfish** 또는 **ProfileBundle** 에도 관계없이 애플리케이션에 대한 실패를 투명하게 수 행할 수 있습니다.

백업은 종종 또 다른 어려운 영역입니다. 다시 말해, 가능한 경우 특정 **inode** 집합을 캐싱하는 노드에서 직접 각 노드의 작업 세트를 백업하는 것이 좋습니다. 일반 시점에 실행되는 백업 스크립트가 있고, 이 경우 **GFS2**에서 실행되는 애플리케이션의 응답 시간에 급증하는 것처럼 보이는 백업 스크립트가 있는 경우 클러스터가 페이지 캐시를 가장 효율적으로 사용하지 못할 가능성이 높습니다.

분명히 백업을 수행하기 위해 응용 프로그램을 중지 할 수있는 위치에있는 경우 이것은 문제가되지 않습니다. 반면 백업이 한 노드에서만 실행되는 경우 파일 시스템의 많은 부분을 완료한 후 다른 노드의 후속 액세스로 성능 저하가 발생합니다. 다음 명령을 사용하여 백업이 완료된 후 백업 노드에서 **vGPU** 페이지 캐시를 삭제하여 일정 범위까지 완화할 수 있습니다.

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

그러나 이 방법은 각 노드에서 작업 집합이 주로 클러스터 전체에서 읽기 전용인지 또는 단일 노드에서 대부분 액세스할 수 있도록 주의를 기울이는 것이 좋지 않습니다.

6.5. GFS2 잠금 덤프를 사용하여 GFS2 성능 문제 해결

GFS2 캐싱을 비효율적으로 사용하기 때문에 클러스터 성능이 저하되는 경우 대규모 **I/O** 대기 시간이 증가할 수 있습니다. **GFS2**의 잠금 덤프 정보를 사용하여 문제의 원인을 확인할 수 있습니다.

debugfs 가 **/sys/kernel/debug/** 에 마운트되었다고 가정하면 **debug2** 잠금 덤프 정보를 **debugfs** 파일에서 수집할 수 있습니다.

/sys/kernel/debug/gfs2/*fsname*/glocks

파일의 내용은 일련의 행입니다. **G:**로 시작하는 각 줄은 하나의 **glock**을 나타내며, 다음 줄은 파일에서 바로 **glock**과 관련된 정보 항목을 하나의 공백으로 나타냅니다.

debugfs 파일을 사용하는 가장 좋은 방법은 **cat** 명령을 사용하여 파일의 전체 콘텐츠 복사본을 사용하는 것입니다(애플리케이션에 많은 **RAM**과 캐시된 **inode**가 있는 경우 시간이 오래 걸릴 수 있음) 애플리케이션에서 문제가 발생하는 동안 나중에 결과 데이터를 확인하는 것입니다.

참고

debugfs 파일의 두 사본을 몇 초 또는 **1**분 또는 **2**분 후에 생성하는 것이 유용할 수 있습니다. 동일한 **glock** 번호와 관련된 두 가지 추적의 보유자 정보를 비교하면 워크로드가 진행 중인지(중지 느림)인지를 알 수 있습니다(항상 버그이므로 **Red Hat** 지원에 즉시 보고해야 함).

H: (holders)로 시작하는 **debugfs** 파일의 행은 부여되거나 부여 대기 중인 잠금 요청을 나타냅니다. **holders line f:**의 **flags** 필드는 다음을 보여줍니다. **'W'** 플래그는 대기 요청을 나타냅니다. **'H'** 플래그는 부여된 요청을 나타냅니다. 많은 수의 대기 요청이 있는 **glock**은 특정 경합이 발생하는 것일 수 있습니다.

다음 표에서는 **glock** 플래그와 **glock** 홀더 플래그의 의미를 보여줍니다.

표 6.1. Glock 플래그

플래그	이름	meaning
b	차단	잠긴 플래그가 설정된 경우 유효하고 DLM에서 요청된 작업이 차단될 수 있음을 나타냅니다. 이 플래그는 demotion 작업 및 "try" 잠금을 위해 지워집니다. 이 플래그를 사용하면 다른 노드에서 demote 잠금으로 걸리는 시간과 관계없이 DLM 응답 시간 통계를 수집할 수 있습니다.
d	보류 중인 데모	Deploying (remote) demote 요청
D	데모	데모 요청(로컬 또는 원격)
f	로그 플러시	이 glock을 해제하기 전에 로그를 커밋해야합니다.

플래그	이름	meaning
F	설정되지 않았습니다.	원격 노드의 응답 무시 - 복구가 진행 중입니다. 이 플래그는 다른 메커니즘을 사용하는 파일 시스템 정지와 관련이 없지만 복구에서만 사용됩니다.
i	진행 중 검증	이 glock 아래의 페이지를 무효화하는 과정에서
l	초기	DLM 잠금이 이 glock과 연결될 때 설정
l	잠긴	glock이 상태를 변경하는 중입니다.
L	LRU	LRU 목록에 glock이 있을 때 설정
o	개체	glock이 오브젝트와 연결될 때 설정 (즉, 유형 2 glock의 inode, 유형 3 glock의 리소스 그룹)
p	진행 중인 데모	glock은 데모 요청에 응답하는 중입니다.
q	Queue	홀더가 glock에 대기 중일 때 설정된 후 glock이 유지될 때 지워지지만 남아 있는 홀더는 없습니다. 알고리즘의 일부로 사용되는 경우 glock의 최소 대기 시간을 계산합니다.
r	응답 보류	원격 노드에서 수신한 응답 처리 대기
y	더티	이 잠금을 해제하기 전에 데이터가 디스크로 플러시해야 합니다.

표 6.2. Glock holder 플래그

플래그	이름	meaning
a	async	glock 결과를 기다리지 마십시오 (나중에 결과에 대해 폴링할 수 있음)
A	Any	모든 호환 가능 잠금 모드가 허용됩니다.

플래그	이름	meaning
c	캐시 없음	잠금 해제 시 DLM이 즉시 잠길 수 있습니다.
e	만료 없음	후속 잠금 취소 요청 무시
E	정확한	정확한 잠금 모드가 있어야 합니다.
F	첫 번째	이 잠금에 대해 부여해야 할 첫 번째 소유자 설정
H	홀더	요청된 잠금이 허용됨을 나타냅니다.
p	우선 순위	대기열 헤드에 있는 대기열 보유자
t	try	"try" 잠금
T	1CB 시도	콜백을 전송하는 "try" 잠금
W	wait	완료 요청 대기 중 설정

문제를 일으키는 glock을 확인한 다음, 다음 단계는 관련 inode를 찾는 것입니다. glock 번호 (n: 라인)는 이를 나타냅니다. 이는 *형식/번호* 이며 *유형이* 2인 경우 glock은 inode glock이고 *숫자는* inode 번호입니다. inode를 추적하려면 find -inum *번호를* 실행합니다. 여기서 *number* 는 glocks 파일의 16진수 형식에서 변환된 inode 번호입니다.

주의

잠금 경합이 발생할 때 파일 시스템에서 find 명령을 실행하면 문제가 더 나빠질 수 있습니다. contended inodes를 찾을 때 find 명령을 실행하기 전에 애플리케이션을 중지하는 것이 좋습니다.

다음 표에서는 다른 glock 유형의 의미를 보여줍니다.

표 6.3. Glock 유형

유형 번호	잠금 유형	사용
1	trans	트랜잭션 잠금
2	inode	inode 메타데이터 및 데이터
3	Rgrp	리소스 그룹 메타데이터
4	meta	수퍼 블록
5	Iopen	inode 마지막 더 가까운 탐지
6	flock	**flock**(2) syscall
8	Quota	할당량 작업
9	journal	비트랜스커뮤니스

확인된 **glock**이 다른 유형인 경우 **3:**(리소스 그룹) 유형일 가능성이 높습니다. 일반 부하에서 다른 유형의 **glock** 대기 중인 상당한 프로세스 수가 표시되면 이를 **Red Hat** 지원에 보고합니다.

리소스 그룹 잠금에 대기 중인 대기 요청이 여러 개 표시되면 이에 대한 여러 가지 이유가 있을 수 있습니다. 하나는 파일 시스템의 리소스 그룹 수와 비교하여 많은 수의 노드가 있다는 것입니다. 또 다른 하나는 파일 시스템이 매우 완전할 수 있다는 것입니다 (중대적으로, 자유 블록을 더 긴 검색 등). 두 경우의 상황은 더 많은 스토리지를 추가하고 **gfs2_grow** 명령을 사용하여 파일 시스템을 확장하여 개선할 수 있습니다.

6.6. 데이터 저널링 활성화

일반적으로 **GFS2**는 저널에 대한 메타데이터만 씁니다. 이후 파일 콘텐츠는 파일 시스템 버퍼를 플러시하는 커널의 주기적 동기화에 의해 디스크에 기록됩니다. **fsync()** 호출을 사용하면 파일의 데이터가 디스크에 즉시 기록됩니다. 호출은 디스크가 모든 데이터가 안전하게 기록된다고 보고할 때 반환됩니다.

데이터 저널링은 파일 데이터가 메타데이터 외에도 저널에 기록되기 때문에 매우 작은 파일에 대해 **fsync()** 시간을 줄 수 있습니다. 이렇게 하면 파일 크기가 증가함에 따라 빠르게 줄어듭니다. 중간 및 큰 파일에 쓰기가 훨씬 느려지고 데이터 저널링이 켜집니다.

fsync() 를 사용하여 파일 데이터를 동기화하는 애플리케이션은 데이터 저널링을 사용하여 성능이 향상될 수 있습니다. 플래그 지정된 디렉토리(및 해당 하위 디렉토리)에서 생성된 모든 **GFS2** 파일에 대해 데이터 저널링을 자동으로 활성화할 수 있습니다. 길이가 **0**인 기존 파일은 데이터 저널링을 켜거나 해제할 수도 있습니다.

디렉토리에서 데이터 저널링을 활성화하면 디렉토리를 **"herit jdata"**로 설정합니다. 그러면 해당 디렉토리에서 생성되는 모든 파일과 디렉터리가 저널되어 있습니다. **chattr** 명령을 사용하여 파일에서 데이터 저널링을 활성화하고 비활성화할 수 있습니다.

다음 명령은 **/mnt/gfs2/gfs2_dir/newfile** 파일에서 데이터 저널링을 활성화한 다음 플래그가 올바르게 설정되었는지 확인합니다.

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

다음 명령은 **/mnt/gfs2/gfs2_dir/newfile** 파일에서 저널링 데이터를 비활성화한 다음 플래그가 올바르게 설정되었는지 확인합니다.

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

또한 **chattr** 명령을 사용하여 디렉토리에 **j** 플래그를 설정할 수도 있습니다. 디렉토리에 이 플래그를 설정하면 해당 디렉토리에서 이후에 생성되는 모든 파일과 디렉터리가 저널됩니다. 다음 명령 세트는 **gfs2_dir** 디렉터리에 **j** 플래그를 설정한 다음 플래그가 올바르게 설정되었는지 확인합니다. 이 후 명령은 **/mnt/gfs2/gfs2_dir** 디렉터리에 **newfile** 이라는 새 파일을 생성한 다음 **j** 플래그가 파일에 설정되어 있는지 확인합니다. **j** 플래그가 디렉터리에 설정된 경우 **newfile** 에도 저널링이 활성화되어야 합니다.

```
# 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
```

7장. GFS2 파일 시스템의 문제 진단 및 수정

다음 절차에서는 일반적인 일부 **GFS2** 문제를 설명하고 해결 방법에 대한 정보를 제공합니다.

7.1. 노드에서 GFS2 파일 시스템을 사용할 수 없음(GFS2 인출 기능)

GFS2 *인출* 기능은 보안 취약점이 있는 하드웨어 또는 커널 소프트웨어로 인한 파일 시스템의 파일 시스템 손상을 방지할 수 있는 **GFS2** 파일 시스템의 데이터 무결성 기능입니다. 지정된 클러스터 노드에서 **GFS2** 파일 시스템을 사용하는 동안 **GFS2** 커널 모듈이 불일치를 감지하면 파일 시스템에서 인출하여 마운트 해제 및 다시 마운트할 때까지 해당 노드에서 사용할 수 없게 됩니다(또는 문제가 재부팅됨을 감지하는 시스템이 재부팅됨). 마운트된 다른 모든 **GFS2** 파일 시스템은 해당 노드에서 완전히 작동하는 상태로 유지됩니다. (**GFS2** 인출 기능은 커널 패닉보다 덜 심각하여 노드를 펜싱할 수 있습니다.)

GFS2 인출을 유발할 수 있는 불일치의 주요 범주는 다음과 같습니다.

- **inode** 일관성 오류
- 리소스 그룹 일관성 오류
- **journal consistency** 오류
- 매직 수 메타데이터 일관성 오류
- 메타데이터 유형 일관성 오류

GFS2 인출으로 인한 불일치의 예로는 파일의 **inode**에 대한 잘못된 블록 수가 있습니다. **GFS2**가 파일을 삭제하면 해당 파일에서 참조하는 모든 데이터 및 메타데이터 블록이 체계적으로 제거됩니다. 완료되면 **inode**의 블록 수를 확인합니다. **inode** 블록 수가 파일에 사용된 실제 블록과 일치하지 않기 때문에 블록 수가 **1**이 아닌 경우(즉, 파일 시스템 불일치를 나타내는) 파일 시스템 불일치를 나타내는 경우 블록 수가 파일에 사용된 실제 블록과 일치하지 않기 때문입니다.

대부분의 경우 이 문제는 결함이 있는 하드웨어(주요 메모리, 마더보드, **HBA**, 디스크 드라이브, 케이블 등)로 인해 발생할 수 있습니다. 커널 버그로 인한 것일 수도 있습니다(다른 커널 모듈에서 실수로 **GFS2** 메모리를 덮어쓰거나, 실제 파일 시스템 손상(**GFS2** 버그에 의해 사용됨)이 발생할 수 있습니다.

대부분의 경우 인출된 **GFS2** 파일 시스템에서 복구하는 가장 좋은 방법은 노드를 재부팅하거나 펜싱하는 것입니다. 제거된 **GFS2** 파일 시스템으로 서비스를 클러스터의 다른 노드로 재배치할 수 있습니다. 서비스가 재배치된 후 노드를 재부팅하거나 이 명령을 사용하여 펜싱을 강제 수행할 수 있습니다.

```
pcs stonith fence node
```

주의

umount 및 **mount** 명령을 사용하여 파일 시스템을 수동으로 마운트 해제하고 다시 마운트하지 마십시오. **pcs** 명령을 사용해야 합니다. 그러지 않으면 **Pacemaker**에서 파일 시스템 서비스가 사라졌고 노드를 펜싱합니다.

인출으로 인한 일관성 문제로 인해 시스템이 중단될 수 있으므로 파일 시스템 서비스를 중지하지 못할 수 있습니다.

다시 마운트한 후에도 문제가 지속되면 파일 시스템 서비스를 중지하여 클러스터의 모든 노드에서 파일 시스템을 마운트 해제한 다음 다음 절차를 통해 서비스를 다시 시작하기 전에 **fsck.gfs2** 명령을 사용하여 파일 시스템 검사를 수행해야 합니다.

1. 영향을 받는 노드를 재부팅합니다.

2. **Pacemaker**에서 복제되지 않은 파일 시스템 서비스를 비활성화하여 클러스터의 모든 노드에서 파일 시스템을 마운트 해제합니다.

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

3. 클러스터 한 노드에서 **fsck.gfs2** 명령을 파일 시스템 장치에서 실행하여 파일 시스템 손상을 확인하고 복구합니다.

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

4. 파일 시스템 서비스를 다시 활성화하여 모든 노드에서 **GFS2** 파일 시스템을 다시 마운트합니다.

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

파일 시스템 서비스에 지정된 **-o errors=panic** 옵션으로 파일 시스템을 마운트하여 **GFS2** 인출 기능을 재정의할 수 있습니다.

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

이 옵션을 지정하면 일반적으로 오류가 발생하면 시스템이 대신 커널 패닉을 강제로 제거하도록 합니다. 이렇게 하면 노드 통신이 중지되어 노드가 펜싱됩니다. 이는 모니터링 또는 개입없이 오랜 기간 동안 자동으로 남아 있는 클러스터에 특히 유용합니다.

내부적으로, 모든 추가 파일 시스템 작업에서 **I/O** 오류가 발생하도록 잠금 프로토콜을 분리하여 **GFS2** 인출 기능이 작동합니다. 결과적으로 인출이 발생하면 시스템 로그에 보고된 장치 매퍼 장치의 여러 **I/O** 오류를 확인하는 것이 일반적입니다.

7.2. GFS2 파일 시스템이 중단되고 하나의 노드를 재부팅해야 합니다.

GFS2 파일 시스템이 중단되고 명령을 반환하지 않지만 특정 노드를 재부팅하면 시스템이 정상적으로 반환되면 잠금 문제 또는 버그가 손상될 수 있습니다. 이러한 문제가 발생하는 경우 문제 해결을 위해 **GFS2** 데이터를 수집하고 **Red Hat** 지원팀에서 지원 티켓을 열고 문제 해결을 위해 **GFS2** 데이터 수집에 설명되어 있습니다.

7.3. GFS2 파일 시스템이 중단되고 모든 노드를 재부팅해야 합니다.

GFS2 파일 시스템이 중단되고 명령을 실행하지 않는 경우 사용하기 전에 클러스터의 모든 노드를 재부팅해야 다음 문제가 있는지 확인하십시오.

- 실패한 펜스가 있을 수 있습니다. **GFS2** 파일 시스템은 펜싱 실패 시 데이터 무결성을 보장하기 위해 동결됩니다. 메시지 로그를 확인하여 중단 시 실패한 펜스가 있는지 확인합니다. 펜싱이 올바르게 구성되었는지 확인합니다.

- **GFS2** 파일 시스템이 제거되었을 수 있습니다. 메시지 로그를 통해 인출 이라는 단어가 있는지 확인하고, 파일 시스템이 제거되었음을 나타내는 **GFS2**의 메시지 및 호출 추적을 확인합니다. 인출은 파일 시스템 손상, 스토리지 오류 또는 버그를 나타냅니다. 파일 시스템을 마운트 해제하는 것이 편리합니다. 최대한 빨리 다음 절차를 수행해야 합니다.

  a. 인출이 발생한 노드를 재부팅합니다.

```
# /sbin/reboot
```

b. 파일 시스템 리소스를 중지하여 모든 노드에서 **GFS2** 파일 시스템을 마운트 해제합니다.

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

c. **gfs2_edit savemeta...**로 메타데이터를 캡처합니다. 명령. 파일에 충분한 공간이 있는지 확인해야 합니다. 경우에 따라 크기가 클 수 있습니다. 이 예에서 메타데이터는 **/root** 디렉터리의 파일에 저장됩니다.

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

d. **gfs2-utils** 패키지를 업데이트합니다.

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

e. 한 노드에서 **fsck.gfs2** 명령을 파일 시스템에서 실행하여 파일 시스템의 무결성을 확인하고 손상을 복구합니다.

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

f. **fsck.gfs2** 명령이 완료되면 파일 시스템 리소스를 다시 활성화하여 서비스로 반환합니다.

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

g. **Red Hat** 지원팀에서 지원 티켓을 엽니다. **sosreports** 및 **gfs2_edit savemeta** 명령에 의해 생성된 디버깅 정보 및 로그를 보고했습니다.

일부 **GFS2**의 경우 파일 시스템 또는 해당 블록 장치에 액세스하려는 명령이 중단될 수 있습니다. 이 경우 클러스터를 재부팅하려면 하드 재부팅이 필요합니다.

GFS2 인출 기능에 대한 정보는 노드에서 사용할 수 없는 **GFS2** 파일 시스템(**GFS2** 인출 기능) 을 참조하십시오.

- 이 오류는 잠금 문제 또는 버그를 나타내는 것일 수 있습니다. 이러한 경우 중 하나를 통해 데이터를 수집하고 문제 해결을 위해 **GFS2** 데이터 수집에 설명된 대로 **Red Hat** 지원팀 에서 지원

티켓을 엽니다.

7.4. 새로 추가된 클러스터 노드에 GFS2 파일 시스템이 마운트되지 않음

클러스터에 새 노드를 추가하고 해당 노드에 GFS2 파일 시스템을 마운트할 수 없는 경우, GFS2 파일 시스템에 액세스하려는 노드보다 GFS2 파일 시스템에 저널이 적을 수 있습니다. 파일 시스템을 마운트하려는 GFS2 호스트당 저널이 있어야 합니다(스펙터 마운트 옵션으로 마운트된 GFS2 파일 시스템을 제외하고 저널이 필요하지 않으므로). 저널 추가에 설명된 대로 gfs2_jadd 명령을 사용하여 GFS2 파일 시스템에 저널을 추가할 수 있습니다.

7.5. 빈 파일 시스템에 사용된 공간

GFS2 파일 시스템이 비어 있는 경우 df 명령은 사용 중인 공간이 있음을 표시합니다. 이는 GFS2 파일 시스템 저널이 디스크의 공간(작업 수 * 저널 크기)을 소비하기 때문입니다. 다수의 저널이 있거나 많은 저널 크기를 지정하여 GFS2 파일 시스템을 생성한 경우 df 명령을 실행할 때 이미 사용 중인 (작업자 * 저널 크기)가 표시됩니다. 많은 수의 저널 또는 대규모 저널을 지정하지 않은 경우에도 소규모 GFS2 파일 시스템(1GB 이하의 경우)은 기본 GFS2 저널 크기와 함께 사용되는 것처럼 많은 공간을 표시합니다.

7.6. 문제 해결을 위한 GFS2 데이터 수집

GFS2 파일 시스템이 중단되고 이에 대해 실행된 명령을 반환하지 않고 Red Hat 지원을 통해 티켓을 열어야 한다는 것을 발견하면 먼저 다음 데이터를 수집해야 합니다.

- 각 노드의 파일 시스템의 GFS2 잠금 덤프

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

- 각 노드의 파일 시스템의 DLM 잠금 덤프 이 정보는 dlm_tool:로 얻을 수 있습니다.

```
dlm_tool lockdebug -sv lsname
```

이 명령에서 *lsname* 은 DLM에서 해당 파일 시스템에 사용하는 잠금 공간 이름입니다. group_tool 명령의 출력에서 이 값을 찾을 수 있습니다.

- sysrq -t 명령의 출력입니다.

- **/var/log/messages 파일의 내용입니다.**

해당 데이터를 수집한 후 Red Hat 지원팀으로 티켓을 열고 수집한 데이터를 제공할 수 있습니다.

8장. 클러스터의://<2 파일 시스템

이 섹션에서는 Red Hat 고가용성 클러스터에서 kubeconfig2 파일 시스템을 구성하기 위한 관리 절차를 설명합니다.

이 섹션에는 다음이 포함됩니다.

- FlexVolume2 파일 시스템 파일 시스템을 포함하는 Pacemaker 클러스터 설정 절차
- 암호화된 FlexVolume2 파일 시스템을 사용하여 Pacemaker 클러스터 설정 절차
- FlexVolume2 파일 시스템이 포함된 RHEL 7 논리 볼륨을 RHEL 8 클러스터로 마이그레이션하는 절차

8.1. 클러스터에서 KUBECONFIG2 파일 시스템 구성

이 절차에서는 FlexVolume2 파일 시스템을 포함하는 Pacemaker 클러스터를 설정하는 데 필요한 단계를 간략하게 설명합니다. 이 예제에서는 세 개의 논리 볼륨에 3개의 ESXi2 파일 시스템을 생성합니다.

전제 조건

- 모든 노드에서 클러스터 소프트웨어를 설치하고 시작하고 기본 2-노드 클러스터를 생성합니다.
- 클러스터의 펜싱을 구성합니다.

Pacemaker 클러스터 생성 및 클러스터의 펜싱 구성에 대한 자세한 내용은 Pacemaker를 사용하여 Red Hat High-Availability 클러스터 생성 을 참조하십시오.

절차

1. 클러스터의 두 노드 모두에서 시스템 아키텍처에 해당하는 탄력적 스토리지 리포지토리를 활성화합니다. 예를 들어 x86_64 시스템의 Resilient Storage 리포지토리를 활성화하려면 다음 subscription-manager 명령을 입력합니다.

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

Resilient Storage 리포지토리는 **High Availability** 리포지토리의 상위 세트입니다. 탄력적 스토리지 리포지토리를 활성화하는 경우 고가용성 리포지토리도 활성화할 필요가 없습니다.

2. 클러스터의 두 노드에서 **lvm2-lockd,gfs2-utils, dlm** 패키지를 설치합니다. 이러한 패키지를 지원하려면 **AppStream** 채널 및 탄력적 스토리지 채널에 가입해야합니다.

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

3. 클러스터의 두 노드 모두에서 **/etc/lvmvm/lvm.conf** 파일에서 **use_lvm lockd** 설정 옵션을 **use_lvmlockd=1** 로 설정합니다.

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

4. 글로벌 **Pacemaker** 매개변수 **no-quorum-policy** 를 **freeze** 로 설정합니다.

참고

기본적으로 **no-quorum-policy** 값은 **stop** 으로 설정되어 쿼럼이 유실되면 나머지 파티션의 모든 리소스가 즉시 중지됨을 나타냅니다. 일반적으로 이 기본값은 가장 안전하고 최적의 옵션이지만 대부분의 리소스와 달리 **FlexVolume2**에는 쿼럼이 있어야 작동합니다. 쿼럼이 모두 손실되면 **kubeconfig2** 마운트를 사용하는 애플리케이션 모두 유실되고 **IRQ2** 마운트 자체를 올바르게 중지할 수 없습니다. 쿼럼 없이 이러한 리소스를 중지하려고 하면 실패하므로 궁극적으로 쿼럼이 손실될 때마다 전체 클러스터가 펜싱됩니다.

이 상황을 해결하려면 **FlexVolume2**를 사용 중인 경우 **no-quorum-policy** 를 중지하도록 설정합니다. 즉, 쿼럼이 손실되면 나머지 파티션은 쿼럼이 될 때까지 아무 작업도 수행하지 않습니다.

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

5. **dlm** 리소스를 설정합니다. 이는 클러스터에서 **FlexVolume2** 파일 시스템을 구성하는 데 필요한 종속 항목입니다. 이 예제에서는 **locking** 이라는 리소스 그룹의 일부로 **dlm** 리소스를 생성합니다.

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

6. 클러스터의 두 노드에서 리소스 그룹을 활성화할 수 있도록 **locking** 리소스 그룹을 복제합니다.

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

7. 그룹 **locking** 의 일부로 **lvmlockd** 리소스를 설정합니다.

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

8. 클러스터의 상태를 확인하여 클러스터의 두 노드에서 **locking** 리소스 그룹이 시작되었는지 확인합니다.

```
[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. 클러스터의 한 노드에서 두 개의 공유 볼륨 그룹을 생성합니다. 하나의 볼륨 그룹에는 두 개의://<2 파일 시스템이 포함되며, 다른 볼륨 그룹에는 하나의 IRQ2 파일 시스템이 포함됩니다.

참고

LVM 볼륨 그룹에 원격 블록 스토리지(예: iSCSI 대상)에 있는 하나 이상의 물리 볼륨이 포함된 경우 Pacemaker가 시작되기 전에 서비스가 시작되도록 하는 것이 좋습니다. Pacemaker 클러스터에서 사용하는 원격 물리 볼륨의 시작 순서를 구성하는 방법에 대한 자세한 내용은 Pacemaker에서 관리하지 않는 리소스 종속 항목에 대한 시작 순서 구성 을 참조하십시오.

다음 명령은 **/dev/vdb** 에 공유 볼륨 그룹 **shared_vg1** 을 생성합니다.

```
[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...
```

다음 명령은 **/dev/ECDHE**에 공유 볼륨 그룹 **shared_vg2** 를 생성합니다.

```
[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.

클러스터의 두 번째 노드에서 다음 명령을 실행하여 공유 장치를 해당 노드의 장치 파일에 추가하고 각 공유 볼륨 그룹의 잠금 관리자를 시작합니다.

```
[root@z2 ~]# lvmdevices --adddev /dev/vdb
[root@z2 ~]# vgchange --lock-start shared_vg1
  VG shared_vg1 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...

[root@z2 ~]# lvmdevices --adddev /dev/vdc
[root@z2 ~]# vgchange --lock-start shared_vg2
  VG shared_vg2 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

11.

클러스터의 한 노드에서 공유 논리 볼륨을 생성하고 **FlexVolume2** 파일 시스템으로 볼륨을 포맷합니다. 파일 시스템을 마운트하는 각 노드에 저널 **1**개가 필요합니다. 클러스터의 각 노드에 대해 충분한 저널을 생성해야 합니다. 잠금 테이블 이름의 형식은 ***ClusterName:FSName*** 입니다. 여기서 ***ClusterName*** 은 **FlexVolume2** 파일 시스템이 생성되고 ***FSName*** 은 클러스터의 모든 **lock_dlm** 파일 시스템에 고유해야 하는 파일 시스템 이름입니다.

```
[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. 각 논리 볼륨에 대해 **LVM-**활성화 리소스를 생성하여 모든 노드에서 해당 논리 볼륨을 자동으로 활성화합니다.

a. 볼륨 그룹 **shared_vg1** 에서 논리 볼륨 **shared_lv1** 에 대해 **sharedlv1** 이라는 **LVM-**활성화 리소스를 만듭니다. 이 명령은 리소스를 포함하는 **resource group shared_vg1** 도 생성합니다. 이 예에서 리소스 그룹은 논리 볼륨을 포함하는 공유 볼륨 그룹과 이름이 동일합니다.

```
[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. 볼륨 그룹 **shared_vg1** 에서 논리 볼륨 **shared_lv2** 에 대해 **sharedlv2** 라는 **LVM-**활성화 리소스를 만듭니다. 이 리소스는 또한 리소스 그룹 **shared_vg1** 의 일부가 됩니다.

```
[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. 볼륨 그룹 **shared_vg2** 에서 논리 볼륨 **shared_lv1** 에 대해 **sharedlv3** 이라는 **LVM-**활성화 리소스를 생성합니다. 이 명령은 리소스를 포함하는 **resource group shared_vg2** 도 생성합니다.

```
[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. 두 개의 새 리소스 그룹을 복제합니다.

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

14. **dlm** 및 **lvmlockd** 리소스가 포함된 **locking** 리소스 그룹이 먼저 시작되도록 순서 지정 제약 조건을 구성합니다.

```
[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. 1 및ECDHE 2 리소스 그룹이 locking 리소스 그룹과 동일한 노드에서 시작되도록 공동 배치 제약 조건을 구성합니다.

```
[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. 클러스터의 두 노드 모두에서 논리 볼륨이 활성 상태인지 확인합니다. 몇 초가 지연될 수 있습니다.

```
[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. 파일 시스템 리소스를 생성하여 각 FlexVolume2 파일 시스템을 모든 노드에 자동으로 마운트합니다.

Pacemaker 클러스터 리소스로 관리되므로 파일 시스템을 /etc/fstab 파일에 추가해서는 안 됩니다. options = 옵션을 사용하여 리소스 구성의 일부로 마운트 옵션을 지정할 수*있습니다*. 전체 구성 옵션에 대해 pcs resource describe Filesystem 명령을 실행합니다.

다음 명령은 파일 시스템 리소스를 생성합니다. 이러한 명령은 해당 파일 시스템의 논리 볼륨 리소스를 포함하는 리소스 그룹에 각 리소스를 추가합니다.

```
[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
```

검증 단계

1. IRQ2 파일 시스템이 클러스터의 두 노드에 마운트되어 있는지 확인합니다.

```
[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. 클러스터 상태를 확인합니다.

```
[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 ]

...
```

추가 리소스

- FlexVolume2 파일 시스템 구성
- Microsoft Azure에서 Red Hat High Availability 클러스터 구성
- AWS에서 Red Hat High Availability 클러스터 구성
- Google Cloud Platform에서 Red Hat High Availability Cluster 구성
- ECDHE Cloud에서 Red Hat High Availability 클러스터에 대한 공유 블록 스토리지 구성

8.2. 클러스터에서 암호화된 FLEXVOLUME2 파일 시스템 구성

(RHEL 8.4 이상) 이 절차에서는 LUKS 암호화된 FlexVolume2 파일 시스템을 포함하는 Pacemaker 클러스터를 생성합니다. 이 예제에서는 논리 볼륨에 하나의 FlexVolume2 파일 시스템을 생성하고 파일 시스템을 암호화합니다. 암호화된 FlexVolume2 파일 시스템은 LUKS 암호화를 지원하는 crypt 리소스 에이전트를 사용하여 지원됩니다.

이 절차에는 세 가지 부분이 있습니다.

- Pacemaker 클러스터에서 공유 논리 볼륨 구성
- 논리 볼륨 암호화 및 암호 리소스 생성
- environments2 파일 시스템으로 암호화된 논리 볼륨 포맷 및 클러스터용 파일 시스템 리소스 생성

8.2.1. Pacemaker 클러스터에서 공유 논리 볼륨 구성

전제 조건

- 모든 노드에서 클러스터 소프트웨어를 설치하고 시작하고 기본 2-노드 클러스터를 생성합니다.
- 클러스터의 펜싱을 구성합니다.

Pacemaker 클러스터 생성 및 클러스터의 펜싱 구성에 대한 자세한 내용은 Pacemaker를 사용하여 Red Hat High-Availability 클러스터 생성 을 참조하십시오.

절차

1. 클러스터의 두 노드 모두에서 시스템 아키텍처에 해당하는 탄력적 스토리지 리포지토리를 활성화합니다. 예를 들어 x86_64 시스템의 Resilient Storage 리포지토리를 활성화하려면 다음 subscription-manager 명령을 입력합니다.

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

Resilient Storage 리포지토리는 High Availability 리포지토리의 상위 세트입니다. 탄력적 스토리지 리포지토리를 활성화하는 경우 고가용성 리포지토리도 활성화할 필요가 없습니다.

2. 클러스터의 두 노드에서 lvm2-lockd,gfs2-utils, dlm 패키지를 설치합니다. 이러한 패키지를 지원하려면 AppStream 채널 및 탄력적 스토리지 채널에 가입해야합니다.

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

3. 클러스터의 두 노드 모두에서 /etc/lvmvm/lvm.conf 파일에서 use_lvm lockd 설정 옵션을 use_lvmlockd=1 로 설정합니다.

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

4. 글로벌 Pacemaker 매개변수 no-quorum-policy 를 freeze 로 설정합니다.

참고

기본적으로 **no-quorum-policy** 값은 **stop** 으로 설정되어 쿼럼이 유실되면 나머지 파티션의 모든 리소스가 즉시 중지됨을 나타냅니다. 일반적으로 이 기본값은 가장 안전하고 최적의 옵션이지만 대부분의 리소스와 달리 **FlexVolume2**에는 쿼럼이 있어야 작동합니다. 쿼럼이 모두 손실되면 **kubeconfig2** 마운트를 사용하는 애플리케이션 모두 유실되고 **IRQ2** 마운트 자체를 올바르게 중지할 수 없습니다. 쿼럼 없이 이러한 리소스를 중지하려고 하면 실패하므로 궁극적으로 쿼럼이 손실될 때마다 전체 클러스터가 펜싱됩니다.

이 상황을 해결하려면 **FlexVolume2**를 사용 중인 경우 **no-quorum-policy** 를 중지하도록 설정합니다. 즉, 쿼럼이 손실되면 나머지 파티션은 쿼럼이 될 때까지 아무 작업도 수행하지 않습니다.

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

5. **dlm** 리소스를 설정합니다. 이는 클러스터에서 **FlexVolume2** 파일 시스템을 구성하는 데 필요한 종속 항목입니다. 이 예제에서는 **locking** 이라는 리소스 그룹의 일부로 **dlm** 리소스를 생성합니다.

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

6. 클러스터의 두 노드에서 리소스 그룹을 활성화할 수 있도록 **locking** 리소스 그룹을 복제합니다.

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

7. 그룹 **locking** 의 일부로 **lvmlockd** 리소스를 설정합니다.

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

8. 클러스터의 상태를 확인하여 클러스터의 두 노드에서 **locking** 리소스 그룹이 시작되었는지 확인합니다.

```
[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. 클러스터의 한 노드에서 공유 볼륨 그룹을 생성합니다.

참고

LVM 볼륨 그룹에 원격 블록 스토리지(예: iSCSI 대상)에 있는 하나 이상의 물리 볼륨이 포함된 경우 Pacemaker가 시작되기 전에 서비스가 시작되도록 하는 것이 좋습니다. Pacemaker 클러스터에서 사용하는 원격 물리 볼륨의 시작 순서를 구성하는 방법에 대한 자세한 내용은 Pacemaker에서 관리하지 않는 리소스 종속 항목에 대한 시작 순서 구성 을 참조하십시오.

다음 명령은 /dev/sda1 에 공유 볼륨 그룹 shared_vg1 을 생성합니다.

```
[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. 클러스터의 두 번째 노드에서 다음 명령을 실행하여 공유 장치를 해당 노드의 장치 파일에 추가하고 공유 볼륨 그룹의 잠금 관리자를 시작합니다.

```
[root@z2 ~]# lvmdevices --adddev /dev/sda1
[root@z2 ~]# vgchange --lock-start shared_vg1
  VG shared_vg1 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

11. 클러스터의 한 노드에서 공유 논리 볼륨을 생성합니다.

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

12.

논리 볼륨에 **LVM-**활성화 리소스를 생성하여 모든 노드에서 논리 볼륨을 자동으로 활성화합니다.

다음 명령은 볼륨 그룹 **shared_vg1** 에서 논리 볼륨 **shared_lv1** 에 대해 **sharedlv1** 이라는 **LVM-**활성화 리소스를 생성합니다. 이 명령은 리소스를 포함하는 **resource group shared_vg1** 도 생성합니다. 이 예에서 리소스 그룹은 논리 볼륨을 포함하는 공유 볼륨 그룹과 이름이 동일합니다.

```
[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. 새 리소스 그룹을 복제합니다.

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

14. **dlm** 및 **lvmlockd** 리소스가 포함된 **locking** 리소스 그룹이 먼저 시작되도록 순서 제한 조건을 구성합니다.

```
[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. **1** 및**ECDHE 2** 리소스 그룹이 **locking** 리소스 그룹과 동일한 노드에서 시작되도록 공동 배치 제약 조건을 구성합니다.

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

검증 단계

클러스터의 두 노드 모두에서 논리 볼륨이 활성 상태인지 확인합니다. 몇 초가 지연될 수 있습니다.

```
[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. 논리 볼륨을 암호화하고 crypt 리소스 생성

사전 요구 사항

- Pacemaker 클러스터에서 공유 논리 볼륨을 구성했습니다.

절차

1. 클러스터의 한 노드에서 crypt 키를 포함할 새 파일을 만들고 root에서만 읽을 수 있도록 파일에 대한 권한을 설정합니다.

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

2. crypt 키를 만듭니다.

```
[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. 설정한 권한을 유지하기 위해 -p 매개변수를 사용하여 클러스터의 다른 노드에 crypt 키 파일을 배포합니다.

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

4. 암호화된 IRQ2 파일 시스템을 구성할 LVM 볼륨에 암호화된 장치를 생성합니다.

```
[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 볼륨 그룹의 일부로 crypt 리소스를 생성합니다.

```
[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"
```

검증 단계

crypt 리소스가 **crypt** 장치를 생성했는지 확인합니다. 이 장치는 **/dev/mapper/luks_lv1** 입니다.

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

8.2.3. Encrypted2 파일 시스템으로 암호화된 논리 볼륨을 포맷하고 클러스터에 대한 파일 시스템 리소스를 생성합니다.

사전 요구 사항

- 논리 볼륨을 암호화하고 **crypt** 리소스를 생성했습니다.

절차

1. 클러스터의 한 노드에서 **IRQ2** 파일 시스템으로 볼륨을 포맷합니다. 파일 시스템을 마운트하는 각 노드에 저널 **1**개가 필요합니다. 클러스터의 각 노드에 대해 충분한 저널을 생성해야 합니다. 잠금 테이블 이름의 형식은 ***ClusterName:FSName*** 입니다. 여기서 ***ClusterName*** 은 **FlexVolume2** 파일 시스템이 생성되고 ***FSName*** 은 클러스터의 모든 **lock_dlm** 파일 시스템에 고유해야 하는 파일 시스템 이름입니다.

```
[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. 모든 노드에 **FlexVolume2** 파일 시스템을 자동으로 마운트하는 파일 시스템 리소스를 생성합니다.

Pacemaker 클러스터 리소스로 관리되므로 파일 시스템을 **/etc/fstab** 파일에 추가하지 마십

시오. **options =** 옵션을 사용하여 리소스 구성의 일부로 마운트 옵션을 지정할 수*있습니다*. 전체 구성 옵션에 대해 **pcs resource describe Filesystem** 명령을 실행합니다.

다음 명령은 파일 시스템 리소스를 생성합니다. 이 명령은 해당 파일 시스템의 논리 볼륨 리소스를 포함하는 리소스 그룹에 리소스를 추가합니다.

```
[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
```

검증 단계

1. **IRQ2** 파일 시스템이 클러스터의 두 노드에 마운트되어 있는지 확인합니다.

```
[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. 클러스터 상태를 확인합니다.

```
[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 ]
...
```

-

추가 리소스

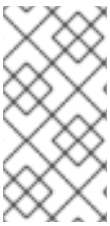
- FlexVolume2 파일 시스템 구성

8.3. RHEL7에서 RHEL8로 TRIGGERBINDING2 파일 시스템 마이그레이션

다음 절차에서는 **kubeconfig2** 파일 시스템을 포함하는 **RHEL 8** 클러스터를 구성할 때 기존 **Red Hat Enterprise 7** 논리 볼륨을 사용할 수 있습니다.

Red Hat Enterprise Linux 8에서 **LVM**은 활성/활성 클러스터에서 공유 스토리지 장치를 관리하기 위해 **clvmd** 대신 **LVM** 잠금 데몬 **lvmlockd** 를 사용합니다. 이를 위해서는 활성/활성 클러스터에 필요한 논리 볼륨을 공유 논리 볼륨으로 구성해야 합니다. 또한 **LVM-**활성화 리소스를 사용하여 **LVM** 볼륨을 관리하고 **lvmlockd** 리소스 에이전트를 사용하여 **lvmlockd** 데몬을 관리해야 합니다. 공유 논리 볼륨을 사용하여 **kube 2** 파일 시스템을 포함하는 **Pacemaker** 클러스터를 구성하기 위한 전체 프로시저는 클러스터에서 **kube2** 파일 시스템 구성 을 참조하십시오.

FlexVolume2 파일 시스템을 포함하는 **RHEL8** 클러스터를 구성할 때 기존 **Red Hat Enterprise Linux 7** 논리 볼륨을 사용하려면 **RHEL8** 클러스터에서 다음 절차를 수행합니다. 이 예에서 클러스터형 **RHEL 7** 논리 볼륨은 볼륨 그룹 **upgrade_gfs_vg** 의 일부입니다.



참고

RHEL8 클러스터의 이름은 기존 파일 시스템을 유효하게 하려면 **FlexVolume2** 파일 시스템을 포함하는 **RHEL7** 클러스터의 이름이 같아야 합니다.

절차

1. **IRQ2** 파일 시스템을 포함하는 논리 볼륨이 현재 비활성화되었는지 확인합니다. 이 절차는 모든 노드가 볼륨 그룹 사용을 중지한 경우에만 안전합니다.

2. 클러스터의 한 노드에서 볼륨 그룹을 강제로 로컬로 변경합니다.

```
[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. 클러스터의 한 노드에서 로컬 볼륨 그룹을 공유 볼륨 그룹으로 변경합니다.

-

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

4. 클러스터의 각 노드에서 볼륨 그룹의 잠금을 시작합니다.

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

이 절차를 수행한 후 각 논리 볼륨에 대해 **LVM** 활성화 리소스를 만들 수 있습니다.

9장. GFS2 추적 지점 및 GLOCK DEBUGFS 인터페이스

이 설명서는 GFS2 추적 지점과 glock debugfs 인터페이스 모두에 대한 문서는 파일 시스템 내부에 익숙한 고급 사용자용이며, GFS2 설계 및 GFS2 관련 문제를 디버그하는 방법에 대해 자세히 알아보고자 합니다.

다음 섹션에서는 GFS2 추적 지점 및 GFS2 glocks 파일에 대해 설명합니다.

9.1. GFS2 추적 지점 유형

현재 세 가지 유형의 GFS2 추적 지점이 있습니다. *glock* (prounced "gee-lock") tracepoints, *bmap* tracepoints 및 *log* tracepoints. 이는 실행 중인 GFS2 파일 시스템을 모니터링하고 이전 Red Hat Enterprise Linux 릴리스에서 지원되는 디버깅 옵션을 통해 얻을 수 있는 추가 정보를 제공하는 데 사용할 수 있습니다. 추적 지점은 중단 또는 성능 문제와 같은 문제를 재현할 수 있으므로 문제가 있는 작업 중에 tracepoint 출력을 얻을 수 있는 경우 특히 유용합니다. GFS2에서 glock은 기본 캐시 제어 메커니즘이며, GFS2 코어의 성능을 이해하는 데 핵심입니다. bmap (block map) 추적 포인트는 발생할 때 블록 할당 및 블록 매핑 (on-disk 메타데이터 트리에 이미 할당된 블록의 조회)을 모니터링하고 액세스의 지역성에 관한 문제를 확인하는 데 사용할 수 있습니다. 로그 추적 지점은 저널에 기록되고 릴리스된 데이터를 추적하여 추적하면 해당 부분에 대한 유용한 정보를 제공할 수 있습니다.

추적 지점은 가능한 한 일반적이 되도록 설계되었습니다. 이는 Red Hat Enterprise Linux 8 과정에서 API를 변경할 필요가 없음을 의미합니다. 반면 이 인터페이스의 사용자는 일반적인 Red Hat Enterprise Linux 8 API 세트의 일부가 아닌 디버깅 인터페이스라는 사실을 알고 있어야 합니다. 따라서 Red Hat은 GFS2 추적 포인트 인터페이스의 변경이 발생하지 않을 것이라는 보장은 없습니다.

추적 지점은 Red Hat Enterprise Linux의 일반적인 기능이며, 해당 범위는 GFS2를 벗어납니다. 특히 blktrace 인프라를 구현하는 데 사용되며 blktrace 추적 포인트를 사용하면 시스템 성능에 대한 전체 정보를 얻을 수 있습니다. 추적 포인트가 작동하는 수준으로 인해 매우 짧은 기간 내에 대량의 데이터를 생성할 수 있습니다. 활성화될 때 시스템에 최소 부하를 배치하도록 설계되었지만 효과가 있다는 것은 불가피합니다. 다양한 방법으로 이벤트를 필터링하면 데이터 양을 줄이고 특정 상황을 이해하는 데 유용한 정보만 얻는 데 집중할 수 있습니다.

9.2. TRACEPOINTS

tracepoints는 /sys/kernel/debug/debug 디렉토리의 표준 위치에 debugfs 가 마운트되었다고 가정하고 /sys/kernel/debug / 디렉토리에서 찾을 수 있습니다. events 하위 디렉터리에는 지정할 수 있는 모든 추적 이벤트가 포함되어 있으며 gfs2 모듈이 로드되면 각 GFS2 이벤트에 대해 gfs2 하위 디렉터리가 포함된 gfs2 하위 디렉터리가 있습니다. /sys/kernel/debug/tracing/gfs2 디렉터리의 내용은 다음과 유사합니다.

```
[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
```

모든 GFS2 추적 지점을 활성화하려면 다음 명령을 입력합니다.

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

특정 추적 지점을 활성화하기 위해 개별 이벤트 하위 디렉터리에는 enable 파일이 있습니다. 동일한 필터 파일에서 각 이벤트 또는 이벤트 세트에 대한 이벤트 필터를 설정하는 데 사용할 수 있습니다. 개별 이벤트의 의미는 아래에서 자세히 설명합니다.

추적 지점의 출력은 ASCII 또는 바이너리 형식으로 사용할 수 있습니다. 이 부록은 현재 바이너리 인터페이스를 다루지 않습니다. ASCII 인터페이스는 두 가지 방법으로 사용할 수 있습니다. 링 버퍼의 현재 콘텐츠를 나열하려면 다음 명령을 입력합니다.

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

이 인터페이스는 일정 기간 동안 장기 실행 프로세스를 사용하고 일부 이벤트 후 버퍼에서 최신 캡처된 정보를 다시 확인하려는 경우에 유용합니다. 대체 인터페이스 /sys/kernel/debug/tracing/trace_pipe 는 모든 출력이 필요할 때 사용할 수 있습니다. 이벤트가 발생하면 이 파일에서 읽습니다. 이 인터페이스를 통해 사용 가능한 기록 정보는 없습니다. 출력 형식은 두 인터페이스에서 모두 동일하며 이 부록의 이후 섹션에 있는 각 GFS2 이벤트에 대해 설명되어 있습니다.

trace-cmd 라는 유틸리티는 tracepoint 데이터를 읽는 데 사용할 수 있습니다. 이 유틸리티에 대한 자세한 내용은 http://lwn.net/Articles/341902/ 을 참조하십시오. trace-cmd 유틸리티를 사용하면 strace 유틸리티와 유사한 방식으로 사용할 수 있습니다. 예를 들어 다양한 소스에서 추적 데이터를 수집하는 동안 명령을 실행할 수 있습니다.

9.3. GLOCKS

GFS2를 이해하기 위한 가장 중요한 개념과 다른 파일 시스템과 별도로 설정하는 개념은 glocks 개념입니다. 소스 코드 측면에서 glock은 DLM을 결합하고 단일 상태 시스템으로 캐싱하는 데이터 구조입니다. 각 glock은 단일 DLM 잠금과 1:1 관계가 있으며 해당 잠금 상태에 대한 캐싱을 제공하므로 파일 시스템의 단일 노드에서 수행되는 반복적인 작업이 DLM을 반복적으로 호출할 필요가 없으므로 불필요한 네트워크 트래픽을 방지할 수 있습니다. 두 가지 광범위한 범주의 glock이 있습니다. 캐시 메타데이터와 그렇지 않은 항목은 무엇입니까. inode glocks 및 리소스 그룹 모두 캐시 메타데이터, 기타 유형의 glocks는 캐시 메타데이터를 캐시하지 않습니다. inode glock은 메타데이터 외에 데이터 캐싱에도 포함되어 있으며 모든 glocks의 가장 복잡한 논리가 있습니다.

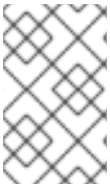
표 9.1. Glock 모드 및 DLM 잠금 모드

Glock 모드	DLM 잠금 모드	참고
UN	IV/NL	잠금 해제 (I 플래그에 따라 glock 또는 NL 잠금과 연결된 DLM 잠금 없음)
SH	PR	공유 (보호된 읽기) 잠금
EX	EX	배타적 잠금
DF	CW	Direct I/O 및 파일 시스템 정지에 사용되는 유한 (concurrent write)

glock은 잠금 해제될 때까지(다른 노드의 요청 또는 **VM** 요청 시) 로컬 사용자가 없을 때까지 메모리에 유지됩니다. 그 시점에서 그들은 **glock** 해시 테이블에서 제거되고 해제됩니다. **glock**이 생성되면 **DLM** 잠금과 즉시 **glock**과 연결되지 않습니다. **DLM** 잠금은 **DLM**에 대한 첫 번째 요청 시 **glock**과 연결되며 이 요청이 성공하면 **glock**에 **'I'(initial)** 플래그가 설정됩니다. **glock debugfs** 인터페이스의 **"Glock flags"** 테이블은 다른 **glock** 플래그의 의미를 보여줍니다. **DLM**이 **glock**과 연결되면 **glock**을 해제할 때까지 **DLM** 잠금은 항상 **NL (ECDHE)** 잠금 모드를 유지합니다. 잠금 해제로 **DLM** 잠금을 해제하는 것은 항상 **glock**의 수명 중 마지막 작업입니다.

각 **glock**은 그에 연결된 여러 "보유자"를 가질 수 있으며, 각각은 상위 계층에서 하나의 잠금 요청을 나타냅니다. 코드의 중요한 섹션을 보호하기 위해 **GFS2** 큐와 관련된 시스템 호출 및 **glock** 소유자와 관련된 시스템 호출.

glock 상태 시스템은 작업 큐를 기반으로 합니다. 성능상의 이유로 **tasklet**이 더 바람직할 수 있지만, 현재 구현에서는 **I/O**를 해당 컨텍스트에서 제출하여 사용을 금지해야 합니다.



참고

Workqueue에는 **GFS2** 추적 지점과 함께 사용할 수 있는 자체 추적 지점이 있습니다.

다음 표는 각 **glock** 모드에서 캐시될 수 있는 상태와 해당 캐시된 상태가 더러워질 수 있는지 여부를 보여줍니다. 이는 리소스 그룹 잠금에 대한 데이터 구성 요소가 없지만 메타데이터만 있지만 **inode** 및 리소스 그룹 잠금 모두에 적용됩니다.

표 9.2. Glock 모드 및 데이터 유형

Glock 모드	캐시 데이터	캐시 메타데이터	더티 데이터	더티 메타데이터
UN	없음	없음	없음	없음

Glock 모드	캐시 데이터	캐시 메타데이터	더티 데이터	더티 메타데이터
SH	있음	있음	없음	없음
DF	없음	있음	없음	없음
EX	있음	있음	있음	있음

9.4. GLOCK DEBUGFS 인터페이스

glock debugfs 인터페이스를 사용하면 **glock**과 홀더의 내부 상태를 시각화할 수 있으며 경우에 따라 잠긴 객체에 대한 몇 가지 요약 세부 정보도 포함되어 있습니다. 파일의 각 줄은 들여쓰기 없이 **G:**를 사용하거나**(Glock** 자체 참조**)** 또는 단일 공백으로 들여쓰기하여 다른 문자로 시작하고**,** 파일의 바로 위 **glock**과 관련된 구조를 나타냅니다**(H:**는 홀더**, I: R: a resource group) .** 다음은 이 파일의 내용은 다음과 같습니다**.**

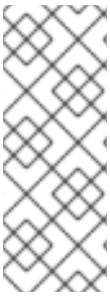
```
G:  s:SH n:5/75320 f:I t:SH d:EX/0 a:0 r:3
 H: s:SH f:EH e:0 p:4466 [postmark] gfs2_inode_lookup+0x14e/0x260 [gfs2]
G:  s:EX n:3/258028 f:yI t:EX d:EX/0 a:3 r:4
 H: s:EX f:tH e:0 p:4466 [postmark] gfs2_inplace_reserve_i+0x177/0x780 [gfs2]
 R: n:258028 f:05 b:22256/22256 i:16800
G:  s:EX n:2/219916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

위의 예는 단일 노드 **GFS2** 파일 시스템에서 **postmark** 벤치마크를 실행하는 동안 **cat /sys/kernel/gfs2/unity:myfs/glocks >my.lock** 명령으로 생성된 일련의 발췌 **(18MB** 파일에서**)**입니다. 그림의 **glocks**는 **glock dumps**의 더 흥미로운 기능을 표시하기 위해 선택되었습니다.

glock 상태는 **EX (exclusive), DF (deferred), SH (shared)** 또는 **UN (unlocked)**입니다. 이러한 상태는 **DLM null** 잠금 상태를 나타내는 **UN**을 제외하고 **DLM** 잠금 모드와 직접 대응하거나**, GFS2**가 **DLM** 잠금을 유지하지 않습니다(위에 설명된 대로 **I** 플래그에 따라 다름**). glock**의 **s:** 필드는 잠금의 현재 상태를 나타내고 홀더의 동일한 필드는 요청된 모드를 나타냅니다. 잠금이 부여된 경우 홀더는 **H** 비트가 플래그 **(f:** 필드**)**에 설정됩니다. 그렇지 않으면 **W wait** 비트가 설정됩니다.

n: 필드**(number)**는 각 항목과 관련된 번호를 나타냅니다**. glocks**의 경우**,** 즉 유형 번호이며 위의 예에

서 첫 번째 glock은 n:5/7532mtls이며, inode 75320과 관련된 iopen glock을 나타냅니다. inode와 iopen glocks의 경우 glock 번호는 항상 inode의 디스크 블록 번호와 동일합니다.



참고

debugfs glocks 파일의 glock 번호(n: 필드)는 16진수에 있는 반면 tracepoints 출력은 10진수로 나열됩니다. 이것은 과거의 이유 때문입니다. glock 번호는 항상 16진수로 작성되었지만 추적점에 대해 10진수가 선택되었으므로 다른 추적 포인트 출력(예: blktrace부터)과 stat(1)의 출력과 비교될 수 있습니다.

소유자와 glock 모두에 대한 모든 플래그의 전체 목록은 아래 "Glock flags" 테이블에 있으며, 아래 Glock Holder FeatureGates의 "Glock Holder flags" 테이블에 설정됩니다. 잠금 값 블록의 내용은 현재 glock debugfs 인터페이스를 통해 사용할 수 없습니다.

다음 표에서는 다른 glock 유형의 의미를 보여줍니다.

표 9.3. Glock 유형

유형 번호	잠금 유형	사용
1	trans	트랜잭션 잠금
2	inode	inode 메타데이터 및 데이터
3	rgrp	리소스 그룹 메타데이터
4	meta	수퍼 블록
5	iopen	inode 마지막 더 가까운 탐지
6	flock	**flock**(2) syscall
8	quota	할당량 작업
9	journal	비트랜스커뮤니스

가장 중요한 glock 플래그 중 하나는 l (locked) 플래그입니다. 이는 상태 변경을 수행해야 할 때 glock 상태에 대한 액세스를 중재하는 데 사용되는 비트 잠금입니다. 상태 시스템이 DLM을 통해 원격 잠금 요청을 보내려는 경우 설정되고 전체 작업이 수행된 경우에만 삭제됩니다. 경우에 따라 여러 번 발생하는 다양한 무효화로 두 개 이상의 잠금 요청이 전송됨을 의미할 수 있습니다.

다음 표는 다른 glock 플래그의 의미를 보여줍니다.

표 9.4. Glock flags

플래그	이름	meaning
d	보류 중인 데모	Deploying (remote) demote 요청
D	데모	데모 요청(로컬 또는 원격)
f	로그 플러시	이 glock을 해제하기 전에 로그를 커밋해야합니다.
F	설정되지 않았습니다.	원격 노드의 응답 무시 - 복구가 진행 중입니다.
i	진행 중 검증	이 glock 아래의 페이지를 무효화하는 과정에서
I	초기	DLM 잠금이 이 glock과 연결될 때 설정
l	잠긴	glock이 상태를 변경하는 중입니다.
L	LRU	LRU 목록에 glock이 있을 때 설정
o	개체	glock이 오브젝트와 연결될 때 설정 (즉, 유형 2 glock의 inode, 유형 3 glock의 리소스 그룹)
p	진행 중인 데모	glock은 데모 요청에 응답하는 중입니다.
q	Queue	홀더가 glock에 대기 중일 때 설정된 후 glock이 유지될 때 지워지지만 남아 있는 홀더는 없습니다. 알고리즘의 일부로 사용되는 경우 glock의 최소 대기 시간을 계산합니다.
r	응답 보류	원격 노드에서 수신한 응답 처리 대기
y	더티	이 잠금을 해제하기 전에 데이터가 디스크로 플러시해야 합니다.

로컬 노드에서 보유 중인 것과 충돌하는 모드에서 잠금을 가져오려는 노드에서 원격 콜백을 수신하면 두 플래그 D(demote) 또는 d(demote pending) 중 하나 또는 기타와 충돌합니다. 특정 잠금에 경합이 있

을 때 배지 상태를 방지하기 위해 각 잠금에는 최소 대기 시간이 할당됩니다. 최소 보류 시간에 대한 잠금이 없는 노드는 간격이 만료될 때까지 해당 잠금을 유지할 수 있습니다.

시간 간격이 만료된 경우 **D(demote)** 플래그가 설정되고 필요한 상태가 기록됩니다. 이 경우 다음에 홀더 큐에 잠금이 부여되지 않으면 잠금이 차단됩니다. 시간 간격이 만료되지 않은 경우 **d(demote pending)** 플래그가 대신 설정됩니다. 또한 최소 대기 시간이 만료되면 **d(**데류 중 보류 중**)**를 지우도록 상태 시스템을 예약하고 **D(demote)**를 설정합니다.

glock에 **DLM** 잠금이 할당되면 **I(initial)** 플래그가 설정됩니다. 이는 **glock**이 처음 사용되고, **I** 플래그는 **glock**이 최종적으로 해제될 때까지(**DLM** 잠금을 잠금 해제) 때까지 설정된 상태로 유지됩니다.

9.5. GLOCK 보유자

다음 표는 다른 **glock** 홀더 플래그의 의미를 보여줍니다.

표 9.5. Glock Holder 플래그

플래그	이름	meaning
a	async	glock 결과를 기다리지 마십시오 (나중에 결과에 대해 폴링할 수 있음)
A	Any	모든 호환 가능 잠금 모드가 허용됩니다.
c	캐시 없음	잠금 해제 시 DLM이 즉시 잠길 수 있습니다.
e	만료 없음	후속 잠금 취소 요청 무시
E	정확한	정확한 잠금 모드가 있어야 합니다.
F	첫 번째	이 잠금에 대해 부여해야 할 첫 번째 소유자 설정
H	홀더	요청된 잠금이 허용됨을 나타냅니다.
p	우선 순위	대기열 헤드에 있는 대기열 보유자
t	try	"try" 잠금
T	1CB 시도	콜백을 전송하는 "try" 잠금

플래그	이름	meaning
W	wait	완료 요청 대기 중 설정

앞서 언급한 대로 가장 중요한 홀더 플래그는 **H(소유자)**와 **W(wait)**입니다. 이는 각각 부여된 잠금 요청 및 대기 중인 잠금 요청을 설정되었기 때문입니다. 목록에 있는 홀더의 순서가 중요합니다. 부여된 소유자가 있는 경우, 해당 소유자는 항상 큐의 헤드에 있고 그 후에는 대기 중인 소유자입니다.

지정된 홀더가 없는 경우 목록의 첫 번째 소유자는 다음 상태 변경을 트리거하는 대상이 됩니다. 데모 요청은 파일 시스템의 요청보다 항상 우선 순위가 높기 때문에 항상 요청된 상태 변경을 유발하지 못할 수 있습니다.

glock 하위 시스템은 두 가지 종류의 **"try"** 잠금을 지원합니다. 이는 정상적인 순서(적용 백오프 및 재시도)를 생략하는 것을 허용하고 다른 노드에서 리소스를 사용하지 않도록 하는 데 사용할 수 있기 때문에 둘 다 유용합니다. 일반적인 **T (try)** 잠금은 이름이 나타내는 것입니다; 특별한 작업을 수행하지 않는 **"try"** 잠금입니다. 반면, **T (try 1CB)** 잠금은 **DLM**이 현재 호환되지 않는 잠금 홀더에 단일 콜백을 보내는 것을 제외하고는 **t** 잠금과 동일합니다. **T (try 1CB)** 잠금의 한 가지 사용에는 **iopen** 잠금이 있으며, **inode**의 **i_nlink** 수가 **0**일 때 노드 간에 중재되는 데 사용되는 **iopen** 잠금이 있으며, **inode**를 재배치할 노드를 결정합니다. **iopen glock**은 일반적으로 공유 상태에 있지만 **i_nlink** 수가 **0**이 되고 →**evict_inode()**가 호출되면 **T (try 1CB**수행**)**가 설정된 배타적 잠금을 요청합니다. 잠금이 부여된 경우 **inode** 할당을 계속 해제합니다. 잠금이 부여되지 않으면 노드 **(s)**가 **D (demote)** 플래그를 사용하여 **glock**을 표시하는 것을 방지했습니다.이 플래그 **(demote)** 플래그. →**drop_inode()** 시간에 체크되어 거래 위치를 잊혀지지 않도록 합니다.

즉, 링크 수가 **0**개 있지만 열려 있는 **inode**는 최종 닫기()가 발생하는 노드에서 할당 해제됩니다. 또한 **inode**의 링크 수와 동시에 **inode**가 **0**으로 감소하고, **inode**는 제로 링크 수가 없는 특수 상태에 있지만 리소스 그룹 **rootfs**에서 여전히 사용 중인 것으로 표시됩니다. 이 함수는 **ext3** 파일 **system3**의 고립 목록과 같은 함수를 통해 비트맵의 이후 리더는 회수될 가능성이 있는 공간이 있음을 알 수 있고 회수를 시도합니다.

9.6. GLOCK TRACEPOINTS

또한 추적 지점은 **blktrace** 출력과 디스크상의 레이아웃에 대한 정보와 결합하여 캐시 컨트롤의 정확성을 확인할 수 있도록 설계되었습니다. 그런 다음 주어진 **I/O**가 올바른 잠금에 따라 발행 및 완료되었는지, 그리고 아무런 경쟁이 없는지 확인할 수 있습니다.

gfs2_glock_state_change tracepoint는 이해해야 할 가장 중요한 요소입니다. 이 명령은 **gfs2_glock_put** 로 끝나는 최종 데모를 통해 초기 생성 시 **glock**의 모든 상태 변경을 추적하고 최종 **NL**으로 전환하여 전환을 잠금 해제할 수 있습니다. **l (locked)** 플래그는 항상 상태 변경이 발생하기 전에 설정되며 완료된 후 지워지지 않습니다. 상태 변경 중에 부여 된 소유자 **(H glock holder** 플래그)는 없습니

다. 대기 중인 홀더가 있는 경우 항상 **W (waiting)** 상태가 됩니다. 상태 변경이 완료되면 **l glock** 플래그가 지워지기 전에 최종 작업인 홀더가 부여될 수 있습니다.

gfs2_demote_rq tracepoint는 로컬 및 원격 둘 다 데모 요청을 추적합니다. 노드에 충분한 메모리가 있다고 가정하면 로컬 데모 요청이 거의 표시되지 않으며 대부분의 경우 **umount** 또는 메모리 회수에 의해 생성됩니다. 원격 데모 요청 수는 특정 **inode** 또는 리소스 그룹의 노드 간 경합 측정입니다.

gfs2_glock_lock_time tracepoint는 **DLM**에 대한 요청에서 가져온 시간에 대한 정보를 제공합니다. 차단 **(b)** 플래그는이 추적 포인트와 함께 사용하기 위해 특별히 **glock**에 도입되었습니다.

홀더에 잠금이 부여되면 **gfs2_promote** 가 호출되면 상태 변경의 최종 단계 또는 **glock** 상태에 이미 적합한 모드의 잠금을 캐싱하여 즉시 부여할 수 있는 잠금이 요청될 수 있습니다. 소유자가 이 **glock**에 대해 부여해야 하는 첫 번째 플래그인 경우 해당 홀더에 **f(first)** 플래그가 설정됩니다. 이는 현재 리소스 그룹에서만 사용됩니다.

9.7. BMAP TRACEPOINTS

블록 매핑은 모든 파일 시스템에 중심적인 작업입니다. **GFS2**는 블록당 두 비트가 있는 기존의 비트 기반 시스템을 사용합니다. 이 하위 시스템의 추적 지점의 주요 목적은 블록을 할당 및 매핑하는 데 걸리는 시간을 모니터링할 수 있도록 하는 것입니다.

gfs2_bmap tracepoint는 각 **bmap** 작업에 대해 두 번 호출됩니다. **bmap** 요청을 표시하기 시작할 때 한 번 결과를 표시합니다. 이렇게 하면 요청과 결과를 함께 일치시키고 파일 시스템의 다양한 부분에 있는 블록을 매핑하는 데 걸리는 시간을 측정하거나 다른 파일 오프셋 또는 다른 파일도 측정할 수 있습니다. 또한 반환되는 평균 범위 크기가 요청되는 항목과 비교하여 확인할 수 있습니다.

gfs2_rs 추적 추적 블록 예약은 블록 할당기에서 생성, 사용 및 삭제됩니다.

할당된 블록을 추적하기 위해 **gfs2_block_alloc** 은 할당 시뿐만 아니라 블록을 해제하는 경우에도 호출됩니다. 할당은 모두 블록이 의도한 **inode**에 따라 참조되므로, 라이브 파일 시스템에서 어떤 물리 블록이 속하는지 추적할 수 있습니다. 이는 **blktrace** 와 결합할 때 특히 유용하며, 이는 이 추적점에서 얻은 매핑을 사용하여 관련 **inode**로 다시 참조할 수 있는 문제가 있는 **I/O** 패턴을 표시합니다.

직접 **I/O(iomap)**는 디스크와 사용자의 버퍼 간에 직접 파일 데이터 전송을 수행할 수 있는 대체 캐시 정책입니다. 이로 인해 캐시 적중률이 낮을 경우 이점이 있습니다. **gfs2_iomap_start** 및 **gfs2_iomap_end tracepoints**는 이러한 작업을 추적하며, 작업 유형과 함께 **Direct I/O** 파일 시스템인 **Direct I/O**를 사용하여 매핑을 추적할 수 있습니다.

9.8. 로그 추적 지점

이 하위 시스템의 **tracepoints**는 저널(**gfs2_pin**)에 추가되거나 제거되는 블록을 추적하고, 트랜잭션을 로그로 커밋하는 데 걸린 시간(**gfs2_log_flush**)을 추적합니다. 이 기능은 저널된 성능 문제를 디버깅하려고 할 때 매우 유용할 수 있습니다.

gfs2_log_blocks tracepoint는 로그에서 예약된 블록을 추적하여 워크로드에 대한 로그가 너무 작은지 여부를 표시하는 데 도움이 됩니다.

gfs2_ail_flush tracepoint는 **gfs2_log_flush** 추적 지점에서 **AIL** 목록의 시작 및 종료를 추적할 수 있습니다. **AIL** 목록에는 로그를 통과했지만 아직 다시 작성되지 않은 버퍼가 포함되어 있으며, 파일 시스템에서 사용하기 위해 더 많은 로그 공간을 해제하거나 프로세스에서 동기화 또는 **fsync** 를 요청할 때 정기적으로 플러시됩니다.

9.9. GLOCK 통계

GFS2는 파일 시스템 내에서 발생하는 상황을 추적하는 데 도움이 되는 통계를 유지 관리합니다. 이를 통해 성능 문제를 해결할 수 있습니다.

GFS2는 다음 두 카운터를 유지 관리합니다.

- **dcount:** 요청된 **DLM** 작업 수를 계산합니다. 이는 평균/변수 계산에 얼마나 많은 데이터가 있는지 보여줍니다.
- **qcount:** 요청된 **syscall** 수준 작업 수를 계산합니다. 일반적으로 **qcount** 는 **dcount** 보다 크거나 같습니다.

또한 **GFS2**는 세 개의 평균/변수 쌍을 유지합니다. **mean/variance** 쌍은 기하급수적인 추정치이며 사용된 알고리즘은 네트워크 코드에서 왕복 시간을 계산하는 데 사용됩니다.

GFS2에서 유지 관리되는 평균 및 분산 쌍은 확장되지 않지만 정수 나노초 단위로 구성되어 있습니다.

- **srtt/srttvar:** 비 차단 작업을 위한 원활한 왕복 시간

- **srttb/srttvarb:** 차단 작업을 위한 원활한 라운드 트립 시간

- **irtt/irttvar:** 요청 시간(예: **DLM** 요청 간의 시간)

차단되지 않은 요청은 **DLM** 잠금의 상태가 무엇이든 즉시 완료되는 요청입니다. 현재 **(a)** 현재 잠금 상태가 배타적인 경우**(b)** 요청된 상태가 **null**이거나 잠금 해제되거나 **(c) "try lock"** 플래그가 설정됨을 의미합니다. 차단 요청은 다른 모든 잠금 요청을 처리합니다.

recordsTT에 더 많은 시간이 더 나은 반면, **RTT**에 더 작은 시간은 더 좋습니다.

통계는 두 개의 **sysfs** 파일에 유지됩니다.

- **write stats** 파일입니다. 이 파일은 행당 하나의 **glock**이 포함된 통계를 포함하는 경우를 제외하고 **glocks** 파일과 유사합니다. 데이터는 **glock**이 생성되는 해당 **glock** 유형의 **"per cpu"** 데이터에서 초기화됩니다(가 **0**인 카운터에서 제외). 이 파일은 매우 클 수 있습니다.

- **lkstats** 파일 여기에는 각 **glock** 유형에 대한 **"CPU당"** 통계가 포함되어 있습니다. 행당 하나의 통계가 포함되어 있으며 각 열은 **cpu** 코어입니다. **glock** 유형당 **8**개의 라인이 있으며, 유형은 서로 다음과 같습니다.

9.10. 참고 자료

tracepoints 및 **GFS2 glocks** 파일에 대한 자세한 내용은 다음 리소스를 참조하십시오.

- 내부 잠금 규칙에 대한 자세한 내용은 https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystems/gfs2-glocks.rst 을 참조하십시오.

- 이벤트 추적에 대한 자세한 내용은 https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/trace/events.rst 을 참조하십시오.

- **trace-cmd** 유틸리티에 대한 자세한 내용은 http://lwn.net/Articles/341902/ 을 참조하십시오.

10장. PERFORMANCE CO-1.8.0(PCP)을 사용하여 GFS2 파일 시스템 모니터링 및 분석

PCP(Performance Co-1.8.0)는 GFS2 파일 시스템을 모니터링 및 분석하는 데 도움이 될 수 있습니다. PCP의 GFS2 파일 시스템 모니터링은 pcp-pmda-gfs2 패키지를 통해 제공되는 Red Hat Enterprise Linux의 GFS2 PMDA 모듈을 통해 제공됩니다.

GFS2 PMDA는 debugfs 하위 시스템에서 제공하는 GFS2 통계에서 제공하는 여러 메트릭을 제공합니다. 설치가 완료되면 PMDA는 glocks,glstats, sbstats 파일에 제공된 값을 노출합니다. 이러한 보고서 세트는 마운트된 각 GFS2 파일 시스템에 대한 통계 세트입니다. PMDA는 또한 Kernel Function Tracer (ftrace)에서 노출되는 GFS2 커널 추적 포인트를 사용합니다.

10.1. GFS2 PMDA 설치

올바르게 작동하기 위해 GFS2 PMDA를 사용하려면 debugfs 파일 시스템이 마운트되어야 합니다. debugfs 파일 시스템이 마운트되지 않은 경우 GFS2 PMDA를 설치하기 전에 다음 명령을 실행합니다.

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

GFS2 PMDA는 기본 설치의 일부로 활성화되어 있지 않습니다. PCP를 통한 GFS2 메트릭 모니터링을 사용하려면 설치 후 활성화해야 합니다.

다음 명령을 실행하여 PCP를 설치하고 GFS2 PMDA를 활성화합니다. PMDA 설치 스크립트는 root로 실행해야 합니다.

```
# 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 툴을 사용하여 사용 가능한 성능 지표에 대한 정보 표시

pminfo 툴은 사용 가능한 성능 지표에 대한 정보를 표시합니다. 다음 예제에서는 이 툴을 사용하여 표시할 수 있는 다양한 GFS2 지표를 보여줍니다.

10.2.1. 파일 시스템별로 현재 존재하는 glock 구조 수 검사

GFS2 glock 메트릭은 마운트된 각 GFS2 파일 시스템 및 해당 잠금 상태에 대해 현재 코어 수에 대한 glock 구조 수에 대한 통찰력을 제공합니다. GFS2에서 glock은 DLM을 결합하고 단일 상태 시스템으로 캐싱하는 데이터 구조입니다. 각 glock은 단일 DLM 잠금을 사용하는 1:1 매핑을 제공하며, 잠금 상태에 대한 캐싱을 제공하므로 단일 노드에서 수행되는 반복적인 작업이 DLM을 반복적으로 호출할 필요가 없으므로 불필요한 네트워크 트래픽을 줄일 수 있습니다.

다음 pminfo 명령은 마운트된 GFS2 파일 시스템당 glock 수 목록을 잠금 모드로 표시합니다.

```
# 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. 유형별로 파일 시스템당 존재하는 glock 구조 수 검사

GFS2 glstats 메트릭은 각 파일 ystem에 대해 존재하는 각 glock의 수를 제공하며, 이러한 지표는 일반적으로 inode(inode 및 metadata) 또는 리소스 그룹(resource group metadata) 유형 중 하나입니다.

다음 pminfo 명령은 마운트된 GFS2 파일 시스템당 각 Glock 유형 목록을 표시합니다.

```
# 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. 대기 상태에 있는 glock 구조의 수 확인

가장 중요한 홀더 플래그는 **H (holder:** 요청된 잠금이 부여됨) 및 **W (wait:** 요청이 완료될 때까지 기다리는 동안 설정)입니다. 이러한 플래그는 부여된 잠금 요청 및 대기 중인 잠금 요청에 각각 설정됩니다.

다음 **pminfo** 명령은 마운트된 각 **GFS2** 파일 시스템의 **Wait (W) holder** 플래그와 함께 **glock** 수 목록을 표시합니다.

```
# 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
```

리소스 그룹 잠금에 대기 중인 대기 요청이 여러 개 표시되면 이에 대한 여러 가지 이유가 있을 수 있습니다. 하나는 파일 시스템의 리소스 그룹 수와 비교하여 많은 수의 노드가 있다는 것입니다. 또 다른 하나는 파일 시스템이 매우 완전할 수 있다는 것입니다 (중대적으로, 자유 블록을 더 긴 검색 등). 두 경우의 상황은 더 많은 스토리지를 추가하고 **gfs2_grow** 명령을 사용하여 파일 시스템을 확장하여 개선할 수 있습니다.

10.2.4. 커널 추적 포인트 기반 메트릭을 사용하여 파일 시스템 작업 대기 시간 확인

GFS2 PMDA는 GFS2 커널 추적 지점에서 지표 수집을 지원합니다. 기본적으로 이러한 메트릭의 읽기는 비활성화되어 있습니다. 이러한 메트릭을 활성화하면 지표 값을 채우기 위해 지표가 수집될 때 GFS2 커널 추적 포인트를 켭니다. 이 기능은 이러한 커널 추적 메트릭이 활성화된 경우 성능 처리량에 약간의 영향을 미칠 수 있습니다.

PCP는 pmstore 툴을 제공하여 메트릭 값에 따라 PMDA 설정을 수정할 수 있습니다. gfs2.control.* 메트릭을 사용하면 GFS2 커널 추적 지점을 편집할 수 있습니다. 다음 예제에서는 pmstore 명령을 사용하여 모든 GFS2 커널 추적 포인트를 활성화합니다.

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

이 명령이 실행되면 PMDA는 debugfs 파일 시스템의 모든 GFS2 추적점에서 전환합니다. PCP의 GFS2에 대한 "Complete Metric List" 표는 각 제어 추적 지점 및 사용량을 설명합니다. 각 제어 추적 지점의 영향에 대한 설명과 사용 가능한 옵션에 대한 설명은 pminfo 의 도움말 전환을 통해 사용할 수 있습니다.

GFS2는 메트릭을 파일 시스템에 대한 승격 요청 수를 계산합니다. 이러한 요청은 첫 번째 시도에서 발생한 요청 수와 초기 승격 요청 후 부여된 "기타" 요청으로 구분됩니다. "기타" 승격이 증가하여 처음으로 승격되는 경우 파일 경합에 문제가 있음을 나타낼 수 있습니다.

승격 요청 지표와 같은 GFS2 데모 요청 메트릭은 파일 시스템에서 발생하는 데모 요청 수를 계산합니다. 그러나 이는 현재 노드에서 제공된 요청과 시스템의 다른 노드에서 가져온 요청 간에도 분할됩니다. 원격 노드의 많은 데모 요청은 지정된 리소스 그룹에 대한 두 노드 간의 경합을 나타낼 수 있습니다.

pminfo 툴은 사용 가능한 성능 지표에 대한 정보를 표시합니다. 다음 절차에서는 마운트된 각 GFS2 파일 시스템의 Wait (W) holder 플래그와 함께 glock 수 목록을 표시합니다. 다음 pminfo 명령은 마운트된 각 GFS2 파일 시스템의 Wait (W) holder 플래그와 함께 glock 수 목록을 표시합니다.

```
# 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
```

워크로드가 실행될 때 모니터링되는 일반적인 값을 결정하는 것이 좋습니다. 문제 없이 이러한 값이 정상 범위와 다를 때 성능 변경 사항을 확인할 수 있습니다.

예를 들어 첫 번째 시도 시 완료되는 대신 대기 중인 승격 요청 수의 변경으로 인해 다음 명령의 출력이 결정할 수 있습니다.

```
# 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
```

다음 명령의 출력에서는 원격 데모 요청(특히 다른 클러스터 노드에서 발생하는 경우)의 증가를 결정할 수 있습니다.

```
# 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
```

다음 명령의 출력은 설명되지 않은 로그 플러시 횟수를 나타낼 수 있습니다.

```
# 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. PCP에서 GFS2에 사용 가능한 지표 전체 목록

다음 표는 **GFS2** 파일 시스템의 **pcp-pmda-gfs2** 패키지에서 제공하는 전체 성능 지표 목록을 설명합니다.

표 10.1. 전체 메트릭 목록

메트릭 이름	설명
gfs2.glocks.*	glock 통계 파일(**g**lock)에서 수집된 정보와 관련된 지표는 시스템에 현재 마운트된 각 GFS2 파일 시스템에 대해 현재 존재하는 각 상태의 glock 수를 계산합니다.
gfs2.glocks.flags.*	지정된 glocks 플래그와 함께 존재하는 glocks 수를 계산하는 메트릭 범위
gfs2.holders.*	glock 통계 파일(glock)에서 수집된 정보(glock)에서 수집된 지표는 시스템에 현재 마운트된 각 GFS2 파일 시스템에 대해 현재 존재하는 각 잠금 상태에 있는 소유자와**glocks**의 수를 계산합니다.
gfs2.holders.flags.*	지정된 홀더 플래그가 있는 glocks 보유자 수를 계산하는 메트릭 범위
gfs2.sbstats.*	시스템에 현재 마운트된 각 GFS2 파일 시스템의 수퍼 블록 통계 파일(**sbstats**)에서 수집한 정보에 대한 타이밍 메트릭입니다.
gfs2.glstats.*	glock 통계 파일(보통 통계 파일)에서 수집된 정보와 관련된 지표는 현재 시스템에 마운트된 각 GFS2 파일 시스템에 대해 현재 존재하는 각 유형의 glock의 수를 계산합니다.
gfs2.latency.grant.*	**gfs2_glock_queue** 및 **gfs2_glock_state_change** tracepoints의 데이터를 사용하여 마운트된 각 파일 시스템에 대해 glock 부여 요청을 완료할 평균 대기 시간을 계산합니다. 이 메트릭은 부여 대기 시간이 증가할 때 파일 시스템에서 잠재적인 속도 저하를 검색하는 데 유용합니다.
gfs2.latency.demote.*	마운트된 각 파일 시스템에 대해 **gfs2_glock_state_change** 및 **gfs2_demote_rq** tracepoints의 데이터를 사용하여 microseconds의 평균 대기 시간을 계산합니다. 이 메트릭은 데모 대기 시간이 증가할 때 파일 시스템에서 잠재적인 속도 저하를 검색하는 데 유용합니다.
gfs2.latency.queue.*	**gfs2_glock_queue** 추적 지점의 데이터를 사용하여 마운트된 각 파일 시스템에 대해 glock 큐 요청을 완료하는 데 평균 대기 시간을 계산합니다.
gfs2.worst_glock.*	**gfs2_glock_lock_time** tracepoint의 데이터를 사용하여 마운트된 각 파일 시스템에 대해 "현재 가장 어려운 glock"을 계산하기 위해 파생된 메트릭입니다. 이 메트릭은 동일한 잠금이 여러 번 제안된 경우 잠재적인 잠금 경합 및 파일 시스템의 속도 저하를 검색하는 데 유용합니다.

메트릭 이름	설명
gfs2.tracepoints.*	시스템에 현재 마운트된 각 파일 시스템의 GFS2 **debugfs** 추적 지점의 출력과 관련된 지표입니다. 이러한 지표의 각 하위 유형(각 GFS2 추적 지점 중)은 제어 메트릭을 사용하여 온/오프할지 여부를 개별적으로 제어할 수 있습니다.
gfs2.control.*	PMDA에서 메트릭 레코딩을 켜거나 끄는 데 사용되는 구성 메트릭입니다. conrol metricsare는 **pmstore** 툴을 통해 전환합니다.

10.4. 파일 시스템 데이터 수집을 위한 최소 PCP 설정 수행

다음 절차에서는 Red Hat Enterprise Linux에서 통계를 수집하기 위해 최소 PCP 설정을 설치하는 방법에 대해 간단히 설명합니다. 이 설정에는 추가 분석을 위해 데이터를 수집하는 데 필요한 프로덕션 시스템에 최소 패키지 수를 추가해야 합니다.

pmlogger 출력의 tar.gz 아카이브는 추가 PCP 툴을 사용하여 분석할 수 있으며 다른 성능 정보 소스와 비교할 수 있습니다.

절차

1. **필수 PCP 패키지를 설치합니다.**

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

2. **PCP에 대한 GFS2 모듈을 활성화합니다.**

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

3. **pmcd 및 pmlogger 서비스를 둘 다 시작합니다.**

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

4. **GFS2 파일 시스템에서 작업을 수행합니다.**

5. **pmcd 및 pmlogger 서비스를 둘 다 중지합니다.**

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

6. 출력을 수집하고 호스트 이름과 현재 날짜 및 시간을 기반으로 이름이 지정된 **tar.gz** 파일에 저장합니다.

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

10.5. 추가 리소스

- **GFS2** 추적 지점 및 **glock debugfs** 인터페이스.
- **Performance Co-1.8.0**을 사용하여 성능 모니터링
- **Performance Co-dpdk(PCP)** 문서, 솔루션, 튜토리얼 및 화이트 문서