Red Hat Enterprise Linux 8

시스템 상태 및 성능 모니터링 및 관리

시스템 처리량, 대기 시간 및 전력 소비 최적화

Last Updated: 2022-08-08

Red Hat Enterprise Linux 8 시스템 상태 및 성능 모니터링 및 관리

시스템 처리량, 대기 시간 및 전력 소비 최적화

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 문서 컬렉션에서는 다양한 시나리오에서 Red Hat Enterprise Linux 8의 처리량, 대기 시간 및 전력 소비를 모니터링하고 최적화하는 방법에 대한 지침을 제공합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 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

문서 개선을 위한 의견을 보내 주십시오. Red Hat이 어떻게 이를 개선할 수 있는지 알려 주십시오.

- 특정 문구에 대한 간단한 주석은 다음과 같습니다.
  1. 문서가 *Multi-page HTML* 형식으로 표시되는지 확인합니다. 또한 문서 오른쪽 상단에 **Feedback** (피드백) 버튼이 있는지 확인합니다.
  2. 마우스 커서를 사용하여 주석 처리하려는 텍스트 부분을 강조 표시합니다.
  3. 강조 표시된 텍스트 아래에 표시되는 **피드백 추가** 팝업을 클릭합니다.
  4. 표시된 지침을 따릅니다.
- Bugzilla를 통해 피드백을 제출하려면 새 티켓을 생성하십시오.
  1. Bugzilla 웹 사이트로 이동하십시오.
  2. Component로 **Documentation**을 선택하십시오.
  3. **Description** 필드에 문서 개선을 위한 제안 사항을 기입하십시오. 관련된 문서의 해당 부분 링크를 알려주십시오.
  4. **Submit Bug**를 클릭하십시오.

1장. 성능 모니터링 옵션 개요

다음은 Red Hat Enterprise Linux 8에서 사용할 수 있는 몇 가지 성능 모니터링 및 구성 툴입니다.

- PCP(Performance Co-Pilot)는 시스템 수준의 성능 측정을 모니터링, 시각화, 저장 및 분석하는 데 사용됩니다. 실시간 데이터의 모니터링 및 관리, 기록 데이터의 로깅 및 검색을 가능하게 합니다.
- Red Hat Enterprise Linux 8은 명령줄에서 사용할 수 있는 여러 도구를 제공하여 시스템 외부 실행 수준 **5** 를 모니터링합니다. 다음은 기본 제공되는 명령줄 도구입니다.
  - **top** 은 **procps-ng** 패키지에서 제공합니다. 실행 중인 시스템에서 프로세스의 동적 보기를 제공합니다. 시스템 요약 및 Linux 커널에서 현재 관리 중인 작업 목록을 포함하여 다양한 정보를 표시합니다.
  - **PS** 는 **procps-ng** 패키지에서 제공합니다. 활성 프로세스로 이루어진 일부 그룹의 스냅샷을 캡처합니다. 기본적으로 검사된 그룹은 현재 사용자가 소유한 프로세스와 **ps** 명령이 실행되는 터미널과 연결된 프로세스로 제한됩니다.
  - 가상 메모리 통계(**vm**stat)는 **procps-ng** 패키지에서 제공합니다. 시스템 프로세스, 메모리, 페이징, 블록 입력/출력, 인터럽트 및 CPU 활동의 즉시 보고를 제공합니다.
  - 시스템 활동 리포터(sar)는 **sysstat** 패키지에서 제공합니다. 현재까지 발생한 시스템 활동에 대한 정보를 수집 및 보고합니다.
- **perf** 는 하드웨어 성능 카운터 및 커널 추적 지점을 사용하여 시스템에서 다른 명령 및 애플리케이션의 영향을 추적합니다.
- BCC **-tools** 는 BCC(BPF Compiler Collection)에 사용됩니다. 커널 활동을 모니터링하는 100개가 넘는 **eBPF** 스크립트를 제공합니다. 각 툴에 대한 자세한 내용은 사용 방법과 수행하는 기능을 설명하는 도움말 페이지를 참조하십시오.
- **turbostat** 는 **kernel-tools** 패키지에서 제공합니다. Intel 64 프로세서에서 프로세서 토폴로지, 빈도, 유휴 전원 상태 통계, 온도 및 전력 사용량에 대해 보고합니다.
- **iostat** 는 **sysstat** 패키지에서 제공합니다. 시스템 IO 장치 로드를 모니터링하고 보고하여 관리자가 물리적 디스크 간 IO 로드의 균형을 조정하는 방법을 결정할 수 있습니다.
- **irqbalance** 는 시스템 성능을 향상시키기 위해 프로세서 간에 하드웨어 인터럽트를 배포합니다.
- **SS** 는 소켓에 대한 통계 정보를 인쇄하여 관리자가 시간 경과에 따른 장치 성능을 평가할 수 있습니다. Red Hat Enterprise Linux 8에서는 **netstat** 를 통해 **ss** 를 사용하는 것이 좋습니다.
- **numastat** 는 **numactl** 패키지에서 제공합니다. 기본적으로 **numastat** 는 노드별 NUMA가 커널 메모리 할당기의 누락 시스템 통계를 표시합니다. 최적의 성능은 높은 **numa_hit** 값과 낮은 **numa_miss** 값으로 표시됩니다.
- **numad** 는 자동 NUMA 선호도 관리 데몬입니다. NUMA 리소스 할당, 관리 및 시스템 성능을 동적으로 개선하는 시스템 내의 NUMA 토폴로지 및 리소스 사용량을 모니터링합니다.
- **SystemTap** 은 운영 체제 활동, 특히 커널 활동을 모니터링 및 분석합니다.
- **Valgrind** 는 실행에 따른 기존 애플리케이션 코드를 계측하고 이를 실행하면서 온도 CPU에서 애플리케이션을 실행하여 애플리케이션을 분석합니다. 그런 다음 애플리케이션 실행과 관련된 각 프로세스를 사용자 지정 파일, 파일 설명자 또는 네트워크 소켓에 명확하게 식별하는 주석이 인쇄됩니다. 또한 메모리 누수를 찾는 데 유용합니다.

- **pqos** 는 **intel-cmt-cat** 패키지에서 제공합니다. 최신 Intel 프로세서에서 CPU 캐시 및 메모리 대역폭을 모니터링 및 제어합니다.

추가 리소스

- **PCP**,**top**,**ps**,**vmstat**,**sar**,**perf**,**iostat**,**irqbalance**,**ss**,**numastat**,**numad**,**valgrind**, and **pqos** 도움말 페이지
- **/usr/share/doc/** 디렉토리
- iostat에서 보고한 "await" 값의 의미는 정확히 무엇입니까? Red Hat Knowledgebase 문서
- Performance Co-Pilot을 통한 성능 모니터링

2장. TUNED 시작하기

시스템 관리자는 **TuneD** 애플리케이션을 사용하여 다양한 사용 사례에 맞게 시스템의 성능 프로필을 최적화할 수 있습니다.

2.1. TUNED의 목적

tuned는 시스템을 모니터링하고 특정 워크로드에서 성능을 최적화하는 서비스입니다. **TuneD** 의 핵심은 다양한 사용 사례에 맞게 시스템을 조정하는 *프로필* 입니다.

tuned는 다음과 같은 사용 사례에 대해 사전 정의된 여러 프로필과 함께 배포됩니다.

- 높은 처리량
- 짧은 대기 시간
- 절전

각 프로필에 대해 정의된 규칙을 수정하고 특정 장치를 조정하는 방법을 사용자 지정할 수 있습니다. 다른 프로파일로 전환하거나 **TuneD** 를 비활성화하면 이전 프로필의 시스템 설정에 대한 모든 변경 사항이 원래 상태로 되돌아갑니다.

장치 사용 변경 사항에 대응하고 설정을 조정하여 활성 장치의 성능을 개선하고 비활성 장치의 전력 소비를 줄이기 위해 **TuneD** 를 구성할 수도 있습니다.

2.2. TUNED 프로필

시스템의 상세한 분석은 시간이 매우 오래 걸릴 수 있습니다. **tuned는 일반적인 사용 사례에 대해 사전 정의된 여러 프로필을 제공합니다.** 또한 프로필을 생성, 수정 및 삭제할 수 있습니다.

TuneD 와 함께 제공되는 프로필은 다음 범주로 나뉩니다.

- 절전 프로필
- performance-boosting 프로필

performance-boosting 프로필에는 다음 측면에 중점을 둔 프로필이 포함됩니다.

- 스토리지 및 네트워크에 대한 짧은 대기 시간
- 스토리지 및 네트워크에 대한 높은 처리량
- 가상 머신 성능
- 가상화 호스트 성능

프로파일 구성 구문
tuned.conf 파일에는 **[main]** 섹션 1개와 플러그인 인스턴스를 구성하는 다른 섹션이 포함될 수 있습니다. 그러나 모든 섹션은 선택 사항입니다.

해시 기호(**#**)로 시작하는 행은 주석입니다.

추가 리소스

- **tuned.conf(5)** 도움말 페이지.

2.3. 기본 TUNED 프로필

설치하는 동안 시스템에 가장 적합한 프로필이 자동으로 선택됩니다. 현재 다음과 같은 사용자 지정 규칙에 따라 기본 프로필이 선택됩니다.

환경	기본 프로필	목적
계산 노드	**throughput-performance**	최상의 처리량 성능
가상 머신	**virtual-guest**	최고의 성능. 최상의 성능에 관심이 없는 경우 **balanced** 또는 **powersave** 프로필로 변경할 수 있습니다.
기타 케이스	**균형 조정**	균형 있는 성능 및 전력 소비

추가 리소스

- **tuned.conf(5)** 도움말 페이지.

2.4. 병합 TUNED 프로필

실험적 기능으로 더 많은 프로필을 한 번에 선택할 수 있습니다. **tuned는** 부하 중에 병합하려고 시도합니다.

충돌이 있는 경우 마지막 지정된 프로필의 설정이 우선합니다.

> **예 2.1. 가상 게스트의 전력 소비가 적습니다.**
>
> 다음 예제에서는 최상의 성능을 위해 가상 시스템에서 실행되도록 시스템을 최적화하고 낮은 전력 소비를 위해 동시에 튜닝하는 반면, 낮은 전력 소비는 우선 순위입니다.
>
> ```
> # tuned-adm profile virtual-guest powersave
> ```

주의

병합은 결과 매개 변수 조합이 적합한지 확인하지 않고 자동으로 수행됩니다. 결과적으로 기능은 반생산성일 수 있는 일부 매개 변수를 반대로 조정할 수 있습니다. 예를 들어 throughput- **performance 프로필을 사용하고 spindown-disk 프로필에서 디스크 회전을 낮은 값으로 동시에 설정하여 높은 처리량** 을 위해 디스크를 설정합니다.

추가 리소스

***tuned-adm** man 페이지. * **tuned.conf(5)** 도움말 페이지.

2.5. TUNED 프로필의 위치

tuned는 프로필을 다음 디렉터리에 저장합니다.

/usr/lib/tuned/

배포별 프로필은 디렉터리에 저장됩니다. 각 프로필에는 자체 디렉터리가 있습니다. 프로필은 **tuned.conf** 라는 기본 구성 파일과 선택적으로 다른 파일(예: 도우미 스크립트)으로 구성됩니다.

/etc/tuned/

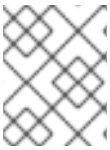
프로필을 사용자 지정해야 하는 경우 사용자 지정 프로필에 사용되는 프로필 디렉터리를 디렉터리로 복사합니다. 동일한 이름의 프로필이 두 개인 경우 **/etc/tuned/** 에 있는 사용자 지정 프로필이 사용됩니다.

추가 리소스

- **tuned.conf(5)** 도움말 페이지.

2.6. RHEL로 배포된 조정된 프로필

다음은 Red Hat Enterprise Linux에서 **TuneD** 를 사용하여 설치되는 프로필 목록입니다.



참고

사용 가능한 제품별 또는 타사 **TuneD** 프로필이 더 있을 수 있습니다. 이러한 프로필은 일반적으로 별도의 RPM 패키지에서 제공합니다.

균형 조정

기본 절전 프로필. 성능 및 전력 소비를 손상시키기 위한 것입니다. 가능한 경우 자동 확장 및 자동 튜닝을 사용합니다. 유일한 단점은 대기 시간이 늘어납니다. 현재 **TuneD** 릴리스에서는 CPU, 디스크, 오디오 및 비디오 플러그인을 활성화하고 **보수적인** CPU governor를 활성화합니다. **radeon_powersave** 옵션은 지원되는 경우 **dpm-balanced** 값을 사용합니다. 그렇지 않으면 **auto** 로 설정됩니다.
이는 **energy_performance_preference** 특성을 **일반** 에너지 설정으로 변경합니다. 또한 **scaling_governor** 정책 특성을 일반 **또는 절전** CPU governor로 변경합니다.

powersave

절전 성능을 최대화하는 프로필입니다. 실제 전력 소비를 최소화하기 위해 성능을 제한할 수 있습니다. 현재 **TuneD** 릴리스에서는 SATA 호스트 어댑터에 대해 USB 자동 일시 중지, TPM 절전, ALPM(Aggressive Link Power Management) 전력 절감을 지원합니다. 또한 기동률이 낮은 시스템에 대해 멀티코어 전력 절감을 예약하고 **온디맨드** governor를 활성화합니다. AC97 오디오 절전 또는 시스템에 따라 10초 시간 초과로 HDA-Intel 전원 절감을 활성화합니다. 시스템에 KMS가 활성화된 지원되는 Radeon 그래픽 카드가 포함된 경우 프로필에서 자동으로 절전되도록 구성합니다. ASUS Eee PC에서는 동적 수퍼 하이브리드 엔진이 활성화되어 있습니다.
이는 **energy_performance_preference** 특성을 **절전 또는 전원 에너지 설정으로 변경합니다. 또한 scaling_governor 정책 속성을 ondemand 또는 powersave CPU governor로 변경합니다.**

참고

특정 경우에는 **powersave** 프로필에 비해 **balanced** 프로필이 더 효율적입니다.

지정된 양의 작업을 수행해야 합니다(예: 트랜스코딩해야 하는 동영상 파일). 작업을 신속하게 완료하면 시스템이 유휴 상태가 되기 때문에 트랜스코딩이 전체 전원에서 수행되면 시스템이 유휴 상태가 되고 매우 효율적인 절전 모드로 자동으로 스테이징될 수 있습니다. 반면 스로틀드 시스템으로 파일을 트랜스코딩하는 경우 시스템은 트랜스코딩 중에 전력을 적게 소비하지만 프로세스가 더 오래 걸리고 전반적으로 소비되는 에너지는 더 높을 수 있습니다.

따라서 **balanced** 프로필은 일반적으로 더 나은 옵션일 수 있습니다.

throughput-performance

높은 처리량에 최적화된 서버 프로필. 절전 메커니즘을 비활성화하고 디스크 및 네트워크 IO의 처리량 성능을 개선하는 **sysctl** 설정을 활성화합니다. **CPU governor**가 **performance**로 설정되어 있습니다.

energy_performance_preference 및 **scaling_governor** 특성을 성능 프로필로 변경합니다.

accelerator-performance

accelerator-performance 프로필에는 **throughput -performance** 프로필과 동일한 튜닝이 포함되어 있습니다. 또한 대기 시간이 **100us** 미만이 되도록 **CPU**가 낮은 **C** 상태로 잠깁니다. 따라서 **GPU**와 같은 특정 액셀러레이터의 성능이 향상됩니다.

latency-performance

짧은 대기 시간에 최적화된 서버 프로필입니다. 절전 메커니즘을 비활성화하고 대기 시간을 개선하는 **sysctl** 설정을 활성화합니다. **CPU governor**는 **performance** 로 설정되고 **CPU**가 낮은 **C** 상태 **(PM QoS**에 의해)로 잠겨 있습니다.

energy_performance_preference 및 **scaling_governor** 특성을 성능 프로필로 변경합니다.

network-latency

짧은 대기 시간 네트워크 튜닝을 위한 프로필입니다. **latency-performance** 프로필을 기반으로 합니다. 또한 투명한 대규모 페이지와 **NUMA** 분산을 비활성화하고 다른 여러 네트워크 관련 **sysctl** 매개변수를 튜닝 합니다.

energy_performance_ preference 및 **scaling_governor** 특성을 성능 프로필로 변경하는

latency- performance 프로필을 상속받습니다.

hpc-compute

고성능 컴퓨팅에 최적화된 프로필입니다. **latency-performance** 프로필을 기반으로 합니다.

network-throughput

처리량 네트워크 튜닝을 위한 프로필입니다. **throughput-performance** 프로필을 기반으로 합니다. 커널 네트워크 버퍼가 추가로 증가합니다.

latency-performance 또는 **throughput-performance** 프로필을 상속하고 **energy_performance_preference** 및 **scaling_governor** 특성을 성능 프로필로 변경합니다.

virtual-guest

다른 작업 중에 가상 메모리 스왑을 줄이고 디스크 읽기 값을 늘리는 **throughput-performance** 프로필을 기반으로 **Red Hat Enterprise Linux 8** 가상 시스템 및 **VMWare** 게스트용으로 설계된 프로필입니다. 디스크 장벽을 비활성화하지 않습니다.

throughput-performance 프로필을 상속하고 **energy_performance_preference** 및 **scaling_governor** 특성을 성능 프로필로 변경합니다.

virtual-host

다른 작업에서 가상 메모리 스왑성을 줄이고 디스크 읽기 헤드 값을 높이며 더티 페이지 나중 쓰기 백 값을 활성화하는 **throughput-performance** 프로필을 기반으로 가상 호스트를 위해 설계된 프로필입니다.

throughput-performance 프로필을 상속하고 **energy_performance_preference** 및 **scaling_governor** 특성을 성능 프로필로 변경합니다.

Oracle

Oracle 데이터베이스에 최적화된 프로필은 처리량-성능 프로필을 기반으로 로드됩니다. 또한 투명한 대규모 페이지를 비활성화하고 다른 성능 관련 커널 매개 변수를 수정합니다. 이 프로필은 **tuned-profiles-oracle** 패키지에서 제공합니다.

desktop

balanced 프로필을 기반으로 하여 데스크탑에 최적화된 프로필입니다. 또한 스케줄러 자동 그룹을 활성화하여 대화형 애플리케이션에 더 효율적으로 응답할 수 있습니다.

optimize-serial-console

printk 값을 줄임으로써 직렬 콘솔에 대한 **I/O** 활동을 조정하는 프로필입니다. 직렬 콘솔의 응답성이 높아집니다. 이 프로필은 다른 프로필에서 오버레이로 사용하기 위한 것입니다. 예를 들면 다음과 같습니다.

```
# tuned-adm profile throughput-performance optimize-serial-console
```

mssql

Microsoft SQL Server에 제공되는 프로필입니다. 이 프로필은 **thoguhput-performance** 프로필을 기반으로 합니다.

intel-sst

사용자 정의 **Intel Speed Select Technology** 구성이 있는 시스템에 최적화된 프로필입니다. 이 프로필은 다른 프로필에서 오버레이로 사용하기 위한 것입니다. 예를 들면 다음과 같습니다.

```
# tuned-adm profile cpu-partitioning intel-sst
```

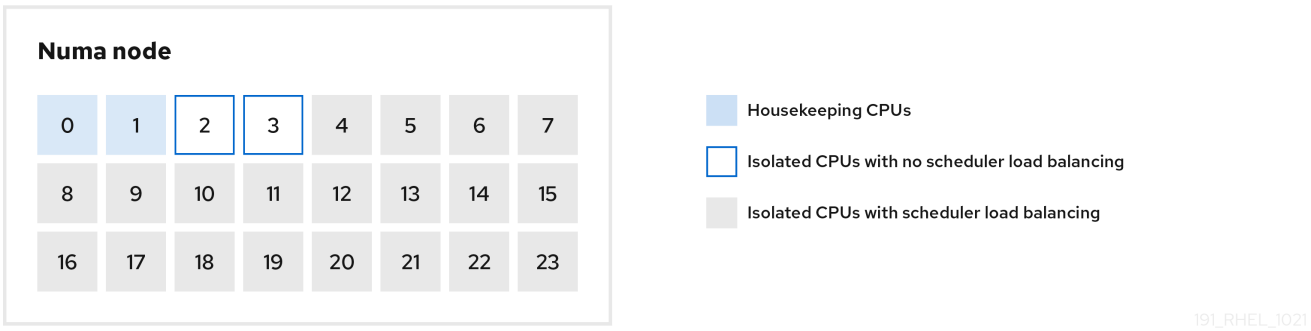
2.7. TUNED CPU-PARTITIONING 프로파일

대기 시간에 민감한 워크로드를 위해 **Red Hat Enterprise Linux 8**을 튜닝하려면 **cpu-partitioning TuneD** 프로필을 사용하는 것이 좋습니다.

Red Hat Enterprise Linux 8 이전에는 대기 시간이 짧은 **Red Hat** 설명서에서는 대기 시간이 짧은 튜닝을 달성하는 데 필요한 수많은 낮은 수준의 단계를 설명했습니다. **Red Hat Enterprise Linux 8**에서는 **cpu-partitioning TuneD** 프로필을 사용하여 대기 시간이 짧은 튜닝을 보다 효율적으로 수행할 수 있습니다. 이 프로필은 대기 시간이 짧은 개별 애플리케이션의 요구 사항에 따라 쉽게 사용자 지정할 수 있습니다.

다음 그림은 **cpu-partitioning** 프로필을 사용하는 방법을 보여주는 예입니다. 이 예에서는 **CPU** 및 노드 레이아웃을 사용합니다.

그림 2.1. 그림 cpu-partitioning



다음 설정 옵션을 사용하여 /etc/tuned/cpu-partitioning-variables.conf 파일에서 cpu-partitioning 프로필을 구성할 수 있습니다.

로드 밸런싱이 있는 분리된 CPU

cpu-partitioning 그림의 4에서 23으로 번호가 지정된 블록은 기본 분리된 CPU입니다. 커널 스케줄러의 프로세스 로드 밸런싱이 이러한 CPU에서 활성화됩니다. 커널 스케줄러 부하 분산이 필요한 여러 스레드가 있는 대기 시간이 짧은 프로세스를 위해 설계되었습니다.

isolated_cores=cpu-list 옵션을 사용하여 /etc/tuned/cpu-partitioning-variables.conf 파일에서 cpu- partitioning 프로필을 구성할 수 있습니다. 이 옵션은 커널 스케줄러 로드 밸런싱을 사용할 CPU를 격리합니다.

분리된 CPU 목록은 쉼표로 구분되거나 3-5 와 같은 대시를 사용하여 범위를 지정할 수 있습니다. 이 옵션은 필수입니다. 이 목록에서 누락된 CPU는 자동으로 하우스키핑 CPU로 간주됩니다.

로드 밸런싱이 없는 분리된 CPU

cpu-partitioning 그림의 경우 번호가 매겨진 2 및 3 블록은 추가 커널 스케줄러 프로세스 로드 밸런싱을 제공하지 않는 분리된 CPU입니다.

no_balance_cores=cpu-list 옵션을 사용하여 /etc/tuned/cpu-partitioning-variables.conf 파일에서 cpu- partitioning 프로필을 구성할 수 있습니다. 이 옵션은 커널 스케줄러 부하 분산을 사용하지 않는 CPU를 격리하도록 나열합니다.

no_balance_cores 옵션을 지정하는 것은 선택 사항이지만 이 목록의 모든 CPU는 isolated_cores 목록에 나열된 CPU의 서브 세트여야 합니다.

이러한 CPU를 사용하는 애플리케이션 스레드를 각 CPU에 개별적으로 고정해야 합니다.

하우스키핑 CPU

cpu-partitioning- Variabless.conf 파일에서 분리되지 않은 모든 CPU는 하우스키핑 CPU로 자동으로 간주됩니다. 하우스키핑 CPU에서는 모든 서비스, 데몬, 사용자 프로세스, 이동 가능한 커널 스레드, 인터럽트 핸들러 및 커널 타이머를 실행할 수 있습니다.

추가 리소스

- tuned-profiles-cpu-partitioning(7) 도움말 페이지

2.8. 대기 시간이 짧은 튜닝을 위해 TUNED CPU-PARTITIONING 프로필 사용

다음 절차에서는 TuneD의 cpu-partitioning 프로필을 사용하여 대기 시간이 짧은 시스템을 조정하는 방법을 설명합니다. cpu-partitioning 그림에 언급된 대로 cpu-partitioning 및 CPU 레이아웃을 사용할 수 있는 대기 시간이 짧은 애플리케이션의 예를 사용합니다.

이 경우 애플리케이션은 다음을 사용합니다.

- 네트워크에서 데이터를 읽는 전용 리더 스레드가 CPU 2에 고정됩니다.
- 이 네트워크 데이터를 처리하는 다수의 스레드가 CPU 4-23에 고정됩니다.
- 처리된 데이터를 네트워크에 쓰는 전용 작성기 스레드는 CPU 3에 고정됩니다.

사전 요구 사항

- yum install tuned-profiles -cpu-partitioning 명령을 root로 사용하여 cpu -partitioning TuneD 프로필을 설치했습니다.

절차

1. /etc/tuned/cpu-partitioning-variables.conf 파일을 편집하고 다음 정보를 추가합니다.

```
# Isolated CPUs with the kernel's scheduler load balancing:
isolated_cores=2-23
# Isolated CPUs without the kernel's scheduler load balancing:
no_balance_cores=2,3
```

2. **cpu-partitioning TuneD 프로필을 설정합니다.**

```
# tuned-adm profile cpu-partitioning
```

3. **재부팅**

재부팅 후 cpu-partitioning 그림의 격리에 따라 시스템이 대기 시간이 짧아지도록 조정됩니다. 이 애플리케이션은 taskset을 사용하여 reader 및 writer 스레드를 CPU 2 및 3에 고정하고 나머지 애플리케이션 스레드는 CPU 4-23에 고정할 수 있습니다.

추가 리소스

- **tuned-profiles-cpu-partitioning(7) 도움말 페이지**

2.9. CPU-PARTITIONING TUNED 프로필 사용자 정의

TuneD 프로필을 확장하여 추가 튜닝을 변경할 수 있습니다.

예를 들어 cpu-partitioning 프로필은 CPU를 cstate=1 으로 설정합니다. cpu-partitioning 프로필을 사용하지만 CPU cstate1을 cstate0으로 추가로 변경하기 위해 다음 절차에서는 cpu-partitioning 프로필을 상속한 다음 C state-0을 설정하는 *my_profile* 이라는 새 TuneD 프로필을 설명합니다.

절차

1. **/etc/tuned/my_profile 디렉토리를 생성합니다.**

```
# mkdir /etc/tuned/my_profile
```

2. **이 디렉터리에 tuned.conf 파일을 생성하고 다음 내용을 추가합니다.**

```
# vi /etc/tuned/my_profile/tuned.conf
[main]
summary=Customized tuning on top of cpu-partitioning
include=cpu-partitioning
[cpu]
force_latency=cstate.id:0|1
```

3. 새 프로필을 사용합니다.

```
# tuned-adm profile my_profile
```

참고

공유 예에서는 재부팅이 필요하지 않습니다. 그러나 ***my_profile*** 프로필의 변경 사항을 적용하려면 시스템을 재부팅해야 합니다.

추가 리소스

- **tuned-profiles-cpu-partitioning(7)** 도움말 페이지

2.10. RHEL을 사용하여 배포된 실시간 TUNED 프로필

실시간 프로필은 실시간 커널을 실행하는 시스템을 위한 것입니다. 특수 커널 빌드가 없으면 시스템을 실시간으로 구성하지 않습니다. **RHEL**의 프로필은 추가 리포지토리에서 사용할 수 있습니다.

다음과 같은 실시간 프로필을 사용할 수 있습니다.

realtime

베어메탈 실시간 시스템에서 사용.

RT 또는 **NFV** 리포지토리에서 사용할 수 있는 **tuned-profiles-realtime** 패키지에서 제공합니다.

realtime-virtual-host

실시간용으로 구성된 가상화 호스트에서 를 사용합니다.

NFV 리포지토리에서 사용할 수 있는 **tuned-profiles-nfv-host** 패키지에서 제공합니다.

realtime-virtual-guest

실시간용으로 구성된 가상화 게스트에서 사용.

NFV 리포지토리에서 사용할 수 있는 **tuned-profiles-nfv-guest** 패키지에서 제공합니다.

2.11. TUNED의 정적 및 동적 튜닝

이 섹션에서는 **TuneD** 가 적용되는 두 가지 시스템 튜닝 범주의 차이점(정적 및 *동적*) *에* 대해 설명합니다.

정적 튜닝

주로 사전 정의된 **sysctl** 및 **sysfs** 설정 적용 및 **ethtool** 과 같은 여러 구성 도구의 한 가지 스냅샷 활성화로 구성됩니다.

동적 튜닝

시스템 가동 시간 전체에서 다양한 시스템 구성 요소를 사용하는 방법을 살펴봅니다. **tuned**는 모니터링 정보를 기반으로 시스템 설정을 동적으로 조정합니다.

예를 들어, 하드 드라이브는 시작 및 로그인 중에 많이 사용되지만 사용자가 주로 웹 브라우저 또는 이메일 클라이언트와 같은 애플리케이션에서 주로 작업할 수 있는 경우에는 나중에 거의 사용되지 않습니다. 마찬가지로 **CPU** 및 네트워크 장치는 서로 다른 시간에 다르게 사용됩니다. **tuned**는 이러한 구성 요소의 활동을 모니터링하고 사용 중인 변경 사항에 대응합니다.

기본적으로 동적 튜닝이 비활성화됩니다. 이를 활성화하려면 **/etc/tuned/tuned-main.conf** 파일을 편집하고 **dynamic_tuning** 옵션을 **1** 로 변경합니다. 그런 다음 **tuned** 는 시스템 통계를 정기적으로 분석하고 이를 사용하여 시스템 튜닝 설정을 업데이트합니다. 이러한 업데이트 사이의 시간 간격(초)을 구성하려면 **update_interval** 옵션을 사용합니다.

현재 구현된 동적 튜닝 알고리즘은 성능과 절전의 균형을 맞추기 때문에 성능 프로필에서 비활성화됩니다. **TuneD** 프로필에서 개별 플러그인에 대한 동적 튜닝을 활성화하거나 비활성화할 수 있습니다.

예 2.2. 워크스테이션에서 정적 및 동적 튜닝

일반적인 사무실 워크스테이션에서 이더넷 네트워크 인터페이스는 대부분의 시간이 비활성 상태입니다. 몇 개의 이메일만 들어오거나 일부 웹 페이지를 로드할 수 있습니다.

이러한 유형의 부하의 경우 기본적으로처럼 네트워크 인터페이스를 항상 최대 속도로 실행할 필요가 없습니다. **tuned**에는 낮은 활동을 감지한 다음 해당 인터페이스의 속도를 자동으로 낮추므로 일반적으로 전력 사용량이 줄어들 수 있는 네트워크 장치에 대한 모니터링 및 튜닝 플러그인이 있습니다.

인터페이스의 활동이 더 오랫동안 증가하는 경우 예를 들어 **DVD** 이미지가 다운로드되거나 첨부 파일이 큰 이메일이 열리면 **TuneD** 는 이를 감지하고 활동 수준이 높은 동안 최고의 성능을 제공하기 위

해 인터페이스 속도를 최대로 설정합니다.

이 원칙은 CPU 및 디스크용 다른 플러그인에도 사용됩니다.

2.12. TUNED NO-DAEMON 모드

상주 메모리가 필요하지 않은 경우 TuneD 를 no-daemon 모드에서 실행할 수 있습니다. 이 모드에서 TuneD 는 설정을 적용하고 종료합니다.

기본적으로 no-daemon 모드는 다음을 포함하여 이 모드에서 많은 TuneD 기능이 누락되어 비활성화되어 있습니다.

- D-Bus 지원
- 핫플러그 지원
- 설정 롤백 지원

데몬 모드를 활성화하려면 /etc/tuned/tuned-main.conf 파일에 다음 행을 포함합니다.

```
daemon = 0
```

2.13. TUNED 설치 및 활성화

이 절차에서는 TuneD 애플리케이션을 설치 및 활성화하고, TuneD 프로필을 설치하며, 시스템의 기본 TuneD 프로필을 사전 설정합니다.

절차

1. tuned 패키지를 설치합니다.

```
# yum install tuned
```

2. **tuned 서비스를 활성화하고 시작합니다.**

```
# systemctl enable --now tuned
```

3. **선택적으로 실시간 시스템에 TuneD 프로필을 설치합니다.**

```
# yum install tuned-profiles-realtime tuned-profiles-nfv
```

4. **TuneD 프로필이 활성화되어 적용되었는지 확인합니다.**

```
$ tuned-adm active

Current active profile: balanced
```

```
$ tuned-adm verify

Verfication succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

2.14. 사용 가능한 TUNED 프로필 나열

이 절차에서는 현재 시스템에서 사용할 수 있는 모든 TuneD 프로필이 나열됩니다.

절차

- **시스템에서 사용 가능한 모든 TuneD 프로필을 나열하려면 다음을 사용합니다.**

```
$ tuned-adm list

Available profiles:
- accelerator-performance - Throughput performance based tuning with disabled higher latency STOP states
- balanced                - General non-specialized tuned profile
- desktop                 - Optimize for the desktop use-case
- latency-performance     - Optimize for deterministic performance at the cost of increased power consumption
- network-latency         - Optimize for deterministic performance at the cost of increased power consumption, focused on low latency network performance
- network-throughput      - Optimize for streaming network throughput, generally only necessary on older CPUs or 40G+ networks
- powersave               - Optimize for low power consumption
- throughput-performance  - Broadly applicable tuning that provides excellent performance across a variety of common server workloads
```

```
- virtual-guest          - Optimize for running inside a virtual guest
- virtual-host           - Optimize for running KVM guests
Current active profile: balanced
```

- 현재 활성화된 프로필만 표시하려면 다음을 사용합니다.

```
$ tuned-adm active

Current active profile: balanced
```

추가 리소스

- **tuned-adm(8)** 도움말 페이지.

2.15. TUNED 프로필 설정

이 절차에서는 시스템에서 선택한 **TuneD** 프로필을 활성화합니다.

사전 요구 사항

- **tuned** 서비스가 실행 중입니다. 자세한 내용은 **TuneD 설치 및 활성화** 를 참조하십시오.

절차

1. 선택적으로 **TuneD** 가 시스템에 가장 적합한 프로파일을 권장하도록 할 수 있습니다.

```
# tuned-adm recommend

balanced
```

2. 프로필을 활성화합니다.

```
# tuned-adm profile selected-profile
```

또는 다음과 같은 여러 프로필의 조합을 활성화할 수 있습니다.

```
# tuned-adm profile profile1 profile2
```

예 2.3. 낮은 전력 소비에 최적화된 가상 머신

다음 예제에서는 최상의 성능을 갖춘 가상 시스템에서 실행되도록 시스템을 최적화하고 낮은 전력 소비를 위해 동시에 튜닝하는 반면, 낮은 전력 소비는 우선 순위입니다.

```
# tuned-adm profile virtual-guest powersave
```

3. 시스템에서 현재 활성 **TuneD** 프로파일을 확인합니다.

```
# tuned-adm active

Current active profile: selected-profile
```

4. 시스템을 재부팅합니다.

```
# reboot
```

검증 단계

- **TuneD** 프로필이 활성화되어 적용되었는지 확인합니다.

```
$ tuned-adm verify

Verfication succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

추가 리소스

- **tuned-adm(8)** 도움말 페이지

2.16. TUNED 비활성화

이 절차에서는 **TuneD** 를 비활성화하고 영향을 받는 모든 시스템 설정을 **TuneD** 를 수정하기 전에 원래 상태로 재설정합니다.

절차

- 모든 튜닝을 일시적으로 비활성화하려면 다음을 수행합니다.

```
# tuned-adm off
```

tuned 서비스가 다시 시작된 후 튜닝이 다시 적용됩니다.

- 또는 **tuned** 서비스를 영구적으로 중지하고 비활성화하려면 다음을 수행합니다.

```
# systemctl disable --now tuned
```

추가 리소스

- **tuned-adm(8)** 도움말 페이지

3장. TUNED 프로필 사용자 정의

TuneD 프로필을 생성하거나 수정하여 의도한 사용 사례에 맞게 시스템 성능을 최적화할 수 있습니다.

사전 요구 사항

- 자세한 내용은 TuneD 설치 및 활성화에 설명된 대로 TuneD를 설치하고 활성화합니다.

3.1. TUNED 프로필

시스템의 상세한 분석은 시간이 매우 오래 걸릴 수 있습니다. tuned는 일반적인 사용 사례에 대해 사전 정의된 여러 프로필을 제공합니다. 또한 프로필을 생성, 수정 및 삭제할 수 있습니다.

TuneD 와 함께 제공되는 프로필은 다음 범주로 나뉩니다.

- 절전 프로필
- performance-boosting 프로필

performance-boosting 프로필에는 다음 측면에 중점을 둔 프로필이 포함됩니다.

- 스토리지 및 네트워크에 대한 짧은 대기 시간
- 스토리지 및 네트워크에 대한 높은 처리량
- 가상 머신 성능
- 가상화 호스트 성능

프로파일 구성 구문

tuned.conf 파일에는 [main] 섹션 1개와 플러그인 인스턴스를 구성하는 다른 섹션이 포함될 수 있습니다. 그러나 모든 섹션은 선택 사항입니다.

해시 기호(#)로 시작하는 행은 주석입니다.

추가 리소스

- **tuned.conf(5)** 도움말 페이지.

3.2. 기본 TUNED 프로필

설치하는 동안 시스템에 가장 적합한 프로필이 자동으로 선택됩니다. 현재 다음과 같은 사용자 지정 규칙에 따라 기본 프로필이 선택됩니다.

환경	기본 프로필	목적
계산 노드	**throughput-performance**	최상의 처리량 성능
가상 머신	**virtual-guest**	최고의 성능. 최상의 성능에 관심이 없는 경우 **balanced** 또는 **powersave** 프로필로 변경할 수 있습니다.
기타 케이스	**균형 조정**	균형 있는 성능 및 전력 소비

추가 리소스

- **tuned.conf(5)** 도움말 페이지.

3.3. 병합 TUNED 프로필

실험적 기능으로 더 많은 프로필을 한 번에 선택할 수 있습니다. **tuned**는 부하 중에 병합하려고 시도합니다.

충돌이 있는 경우 마지막 지정된 프로필의 설정이 우선합니다.

> **예 3.1.** 가상 게스트의 전력 소비가 적습니다.
>
> 다음 예제에서는 최상의 성능을 위해 가상 시스템에서 실행되도록 시스템을 최적화하고 낮은 전력 소비를 위해 동시에 튜닝하는 반면, 낮은 전력 소비는 우선 순위입니다.

```
# tuned-adm profile virtual-guest powersave
```

주의

병합은 결과 매개 변수 조합이 적합한지 확인하지 않고 자동으로 수행됩니다. 결과적으로 기능은 반생산성일 수 있는 일부 매개 변수를 반대로 조정할 수 있습니다. 예를 들어 **throughput- performance** 프로필을 사용하고 **spindown-disk** 프로필에서 디스크 회전을 낮은 값으로 동시에 설정하여 높은 처리량 을 위해 디스크를 설정합니다.

추가 리소스

*tuned-adm man 페이지. * tuned.conf(5) 도움말 페이지.

3.4. TUNED 프로필의 위치

tuned는 프로필을 다음 디렉터리에 저장합니다.

/usr/lib/tuned/

배포별 프로필은 디렉터리에 저장됩니다. 각 프로필에는 자체 디렉터리가 있습니다. 프로필은 tuned.conf 라는 기본 구성 파일과 선택적으로 다른 파일(예: 도우미 스크립트)으로 구성됩니다.

/etc/tuned/

프로필을 사용자 지정해야 하는 경우 사용자 지정 프로필에 사용되는 프로필 디렉터리를 디렉터리로 복사합니다. 동일한 이름의 프로필이 두 개인 경우 /etc/tuned/ 에 있는 사용자 지정 프로필이 사용됩니다.

추가 리소스

- tuned.conf(5) 도움말 페이지.

3.5. TUNED 프로필 간 상속

tuned 프로필 은 다른 프로필을 기반으로 하고 상위 프로필의 특정 측면만 수정할 수 있습니다.

TuneD 프로필의 **[main]** 섹션은 **include** 옵션을 인식합니다.

```
[main]
include=parent
```

상위 프로필의 모든 설정이 이 *하위* 프로필에 로드됩니다. 다음 섹션에서 *하위* 프로필은 *상위 프로필에서 상속된 특정 설정을 재정의하거나 상위* 프로필에 없는 새 설정을 추가할 수 있습니다 .

일부 매개 변수만 조정하면 / **usr/lib/tuned/**의 사전 설치된 프로필을 기반으로 **/etc /tuned/** 디렉토리에 고유한 *하위* 프로필을 만들 수 있습니다.

TuneD 업그레이드 후와 같이 *상위* 프로필이 업데이트되면 변경 사항이 *하위* 프로필에 반영됩니다.

예 3.2. 균형에 따른 절전 프로파일

다음은 균형이 조정되는 프로필을 확장하고 모든 장치에 대한 **ALPM(Aggressive Link Power Management)**을 최대 절전으로 설정하는 사용자 지정 프로필의 예입니다.

```
[main]
include=balanced

[scsi_host]
alpm=min_power
```

추가 리소스

- **tuned.conf(5)** 도움말 페이지

3.6. TUNED의 정적 및 동적 튜닝

이 섹션에서는 **TuneD** 가 적용되는 두 가지 시스템 튜닝 범주의 차이점(정적 및 *동적*) *에* 대해 설명합니다.

정적 튜닝

주로 사전 정의된 **sysctl** 및 **sysfs** 설정 적용 및 **ethtool** 과 같은 여러 구성 도구의 한 가지 스냅샷 활성화로 구성됩니다.

동적 튜닝

시스템 가동 시간 전체에서 다양한 시스템 구성 요소를 사용하는 방법을 살펴봅니다. **tuned**는 모니터링 정보를 기반으로 시스템 설정을 동적으로 조정합니다.

예를 들어, 하드 드라이브는 시작 및 로그인 중에 많이 사용되지만 사용자가 주로 웹 브라우저 또는 이메일 클라이언트와 같은 애플리케이션에서 주로 작업할 수 있는 경우에는 나중에 거의 사용되지 않습니다. 마찬가지로 **CPU** 및 네트워크 장치는 서로 다른 시간에 다르게 사용됩니다. **tuned**는 이러한 구성 요소의 활동을 모니터링하고 사용 중인 변경 사항에 대응합니다.

기본적으로 동적 튜닝이 비활성화됩니다. 이를 활성화하려면 **/etc/tuned/tuned-main.conf** 파일을 편집하고 **dynamic_tuning** 옵션을 **1** 로 변경합니다. 그런 다음 **tuned** 는 시스템 통계를 정기적으로 분석하고 이를 사용하여 시스템 튜닝 설정을 업데이트합니다. 이러한 업데이트 사이의 시간 간격(초)을 구성하려면 **update_interval** 옵션을 사용합니다.

현재 구현된 동적 튜닝 알고리즘은 성능과 절전의 균형을 맞추기 때문에 성능 프로필에서 비활성화됩니다. **TuneD** 프로필에서 개별 플러그인에 대한 동적 튜닝을 활성화하거나 비활성화할 수 있습니다.

예 3.3. 워크스테이션에서 정적 및 동적 튜닝

일반적인 사무실 워크스테이션에서 이더넷 네트워크 인터페이스는 대부분의 시간이 비활성 상태입니다. 몇 개의 이메일만 들어오거나 일부 웹 페이지를 로드할 수 있습니다.

이러한 유형의 부하의 경우 기본적으로처럼 네트워크 인터페이스를 항상 최대 속도로 실행할 필요가 없습니다. **tuned**에는 낮은 활동을 감지한 다음 해당 인터페이스의 속도를 자동으로 낮추므로 일반적으로 전력 사용량이 줄어들 수 있는 네트워크 장치에 대한 모니터링 및 튜닝 플러그인이 있습니다.

인터페이스의 활동이 더 오랫동안 증가하는 경우 예를 들어 **DVD** 이미지가 다운로드되거나 첨부 파일이 큰 이메일이 열리면 **TuneD** 는 이를 감지하고 활동 수준이 높은 동안 최고의 성능을 제공하기 위해 인터페이스 속도를 최대로 설정합니다.

이 원칙은 **CPU** 및 디스크용 다른 플러그인에도 사용됩니다.

3.7. TUNED 플러그인

플러그인은 **TuneD** 가 시스템에서 다양한 장치를 모니터링하거나 최적화하는 데 사용하는 **TuneD** 프로필의 모듈입니다.

tuned는 다음 두 가지 유형의 플러그인을 사용합니다.

모니터링 플러그인

모니터링 플러그인은 실행 중인 시스템에서 정보를 가져오는 데 사용됩니다. 모니터링 플러그인의 출력은 동적 튜닝을 위한 튜닝 플러그인에서 사용할 수 있습니다.

모니터링 플러그인은 활성화된 튜닝 플러그인에서 지표가 필요할 때마다 자동으로 인스턴스화됩니다. 두 개의 튜닝 플러그인에 동일한 데이터가 필요한 경우 모니터링 플러그인의 인스턴스 하나만 생성되고 데이터를 공유합니다.

튜닝 플러그인

각 튜닝 플러그인은 개별 하위 시스템을 튜닝하고 **tuned** 프로필에서 채워지는 여러 매개 변수를 사용합니다. 각 하위 시스템에는 튜닝 플러그인의 개별 인스턴스에서 처리하는 여러 개의 **CPU** 또는 네트워크 카드와 같은 장치가 있을 수 있습니다. 개별 장치에 대한 특정 설정도 지원됩니다.

TuneD 프로필의 플러그인 구문

플러그인 인스턴스를 설명하는 섹션은 다음과 같은 방식으로 포맷됩니다.

```
[NAME]
type=TYPE
devices=DEVICES
```

이름

로그에 사용되는 플러그인 인스턴스의 이름입니다. 임의의 문자열일 수 있습니다.

유형

튜닝 플러그인의 유형입니다.

장치

이 플러그인 인스턴스에서 처리하는 장치 목록입니다.

devices 행에는 목록, 와일드카드(*) 및 부정(!)이 포함될 수 있습니다. **device** 행이 없으면 ***TYPE*** 의 시스템에 있거나 나중에 연결된 모든 장치는 플러그인 인스턴스에서 처리합니다. **devices=*** 옵션을 사용하는 것과 동일합니다.

예 3.4. 플러그인과 블록 장치 일치

다음 예는 **as sda** 또는 **sdb** 와 같이 **sd** 로 시작하는 모든 블록 장치와 일치하며 해당 장치에서 장벽을 비활성화하지 않습니다.

```
[data_disk]
type=disk
devices=sd*
disable_barriers=false
```

다음 예제는 모든 블록 장치와 일치합니다. 단,**da 1** 및 **sda2:**

```
[data_disk]
type=disk
devices=!sda1, !sda2
disable_barriers=false
```

플러그인 인스턴스를 지정하지 않으면 플러그인이 활성화되지 않습니다.

플러그인에서 더 많은 옵션을 지원하는 경우 플러그인 섹션에서도 지정할 수 있습니다. 옵션을 지정하지 않고 포함된 플러그인에 이전에 지정하지 않은 경우 기본값이 사용됩니다.

짧은 플러그인 구문

플러그인 인스턴스에 대한 사용자 지정 이름이 필요하지 않고 구성 파일에 인스턴스 정의가 하나뿐인 경우 **TuneD** 는 다음과 같은 짧은 구문을 지원합니다.

```
[TYPE]
devices=DEVICES
```

이 경우 유형 행을 생략할 수 있습니다. 그런 다음 인스턴스를 유형과 같은 이름으로 참조합니다. 그런 다음 이전 예제를 다음과 같이 다시 작성할 수 있습니다.

예 3.5. 짧은 구문을 사용하여 블록 장치 일치

```
[disk]
devices=sdb*
disable_barriers=false
```

프로필의 플러그인 정의 충돌

include 옵션을 사용하여 동일한 섹션이 두 번 이상 지정되면 설정이 병합됩니다. 충돌로 인해 병합할 수 없는 경우 마지막 충돌 정의는 이전 설정을 재정의합니다. 이전에 정의된 내용을 모르는 경우 **replace** 부울 옵션을 사용하여 **true** 로 설정할 수 있습니다. 이로 인해 동일한 이름의 이전 정의를 모두 덮어쓰고 병합이 발생하지 않습니다.

enabled=false 옵션을 지정하여 플러그인을 비활성화할 수도 있습니다. 이 동작은 인스턴스를 정의하지 않은 경우와 동일합니다. 플러그인 비활성화는 **include** 옵션에서 이전 정의를 재정의하고 사용자 지정 프로필에서 플러그인이 활성화되지 않도록 하려는 경우 유용합니다.

참고

tuned에는 튜닝 프로필 활성화 또는 비활성화의 일부로 모든 쉘 명령을 실행할 수 있는 기능이 포함되어 있습니다. 이를 통해 **TuneD** 에 아직 통합되지 않은 기능을 사용하여 **TuneD** 프로필을 확장할 수 있습니다.

script 플러그인을 사용하여 임의의 쉘 명령을 지정할 수 있습니다.

추가 리소스

- **tuned.conf(5)** 도움말 페이지

3.8. 사용 가능한 TUNED 플러그인

이 섹션에는 현재 **TuneD** 에서 사용할 수 있는 모든 모니터링 및 튜닝 플러그인이 나열되어 있습니다.

모니터링 플러그인

현재 다음과 같은 모니터링 플러그인이 구현됩니다.

disk

장치 및 측정 간격당 디스크 부하(IO 작업 수) 확보.

net

네트워크 카드 및 측정 간격별로 네트워크 부하(전송된 패킷 수) 가져오기.

load

CPU당 CPU 부하 확보 및 측정 간격.

튜닝 플러그인

현재 다음 튜닝 플러그인이 구현되어 있습니다. 이러한 플러그인 중 일부만 동적 튜닝을 구현합니다. 플러그인에서 지원하는 옵션도 나열됩니다.

cpu

CPU governor를 governor 옵션에 지정된 값으로 설정하고 CPU 부하에 따라 PM QoS(Power Management Quality of Service) CPU Direct Memory Access(DMA) 대기 시간을 동적으로 변경합니다.

CPU 로드가 load_threshold 옵션에서 지정한 값보다 낮은 경우 대기 시간이 latency_high 옵션에서 지정한 값으로 설정되고, 그렇지 않으면 latency_low 에서 지정한 값으로 설정됩니다.

대기 시간을 특정 값으로 강제 적용하고 동적으로 변경되지 않도록 할 수도 있습니다. 이렇게 하려면 force_latency 옵션을 필요한 대기 시간 값으로 설정합니다.

eeepc_she

CPU 부하에 따라 FSB(전면 버스) 속도를 동적으로 설정합니다.

이 기능은 일부 넷북에서 확인할 수 있으며 ASUS Super Hybrid Engine (SHE)이라고도 합니다.

CPU 로드가 load_threshold_powersave 옵션에서 지정한 값과 더 낮거나 같은 경우 플러그인은 FSB 속도를 she_powersave 옵션에 지정된 값으로 설정합니다. CPU 부하가 load_threshold_normal 옵션에서 지정한 값보다 크거나 같은 경우 FSB 속도를 she_normal 옵션에 지정된 값으로 설정합니다.

정적 튜닝은 지원되지 않으며 TuneD 가 이 기능에 대한 하드웨어 지원을 탐지하지 않으면 플러그인을 투명하게 비활성화합니다.

net

Wake-on-LAN 기능을ake _on_lan 옵션에서 지정한 값으로 구성합니다. ethtool 유틸리티와 동일한 구문을 사용합니다. 인터페이스 사용률에 따라 인터페이스 속도를 동적으로 변경합니다.

sysctl

플러그인 옵션으로 지정된 다양한 **sysctl** 설정을 설정합니다.

구문은 ***name=값입니다***. 여기서 ***name*** 은 **sysctl** 유틸리티에서 제공하는 이름과 동일합니다.

TuneD 에서 사용 가능한 다른 플러그인에서 다루지 않는 시스템 설정을 변경해야 하는 경우 **sysctl** 플러그인을 사용합니다. 일부 특정 플러그인에서 설정을 적용하는 경우 다음 플러그인을 선호합니다.

usb

USB 장치의 자동 일시 중지 타임아웃을 **autosuspend** 매개 변수에서 지정한 값으로 설정합니다.

값 **0** 은 **autosuspend**가 비활성화되었음을 의미합니다.

vm

transparent _hugepages 옵션의 값에 따라 투명한 대규모 페이지를 활성화하거나 비활성화합니다.

transparent_hugepages 옵션의 유효한 값은 다음과 같습니다.

- "항상"
- "안함"
- "madvise"

audio

오디오 코덱의 자동 일시 중지 타임아웃을 **timeout** 옵션에 지정된 값으로 설정합니다.

현재 **snd_hda_intel** 및 **snd_ac97_codec codec**가 지원됩니다. 값 **0** 은 **autosuspend**가 비활성화되었음을 의미합니다. 부울 옵션 **reset _controller**를 **true** 로 설정하여 컨트롤러 재설정 을 적용할 수도 있습니다.

disk

디스크 프레임을 확대 옵션에서 지정한 값으로 설정합니다.

또한 다음과 같은 설정도 설정됩니다.

- **apm** 옵션에 지정된 값으로의 **APM**
- **scheduler _quantum** 옵션으로 지정한 값에 대한 스케줄러 쿼터
- 스핀 다운 옵션으로 지정된 값에 대한 디스크 회전 시간 초과
- **disk readahead** 매개변수에 지정된 값의 **readahead**
- 현재 디스크 읽기 묶음에서 **readahead_multiply** 옵션에 지정된 상수를 곱한 값

또한 이 플러그인은 현재 드라이브 사용률에 따라 드라이브에 대한 고급 전원 관리 및 회전 시간 제한 설정을 동적으로 변경합니다. 동적 튜닝은 부울 옵션 동적으로 제어할 수 있으며 기본적으로 활성화됩니다.

scsi_host

SCSI 호스트의 옵션을 조정합니다.

ALPM(Aggressive Link Power Management)을 **ALPM** 옵션으로 지정한 값으로 설정합니다.

mounts

disable _barriers 옵션의 부울 값에 따라 마운트에 대한 장벽을 활성화하거나 비활성화합니다.

script

프로필을 로드하거나 언로드할 때 외부 스크립트 또는 바이너리를 실행합니다. 임의의 실행 파일을 선택할 수 있습니다.

중요

script 플러그인은 주로 이전 릴리스와의 호환성을 위해 제공됩니다. 필요한 기능을 다루는 경우 다른 **TuneD** 플러그인을 사용하는 것이 좋습니다.

tuned는 다음 인수 중 하나를 사용하여 실행 파일을 호출합니다.

- 프로파일을 로드할 때 시작
- 프로파일을 언로드 할 때 중지

실행 파일에 중지 작업을 올바르게 구현하고 시작 작업 중에 변경된 모든 설정을 되돌립니다. 그러지 않으면 **TuneD** 프로필을 변경한 후 롤백 단계가 작동하지 않습니다.

Bash 스크립트는 **/usr/lib/tuned/functions Bash** 라이브러리를 가져와서 여기에 정의된 함수를 사용할 수 있습니다. 이 함수는 기본적으로 **TuneD** 에서 제공하지 않는 기능에만 사용합니다. 함수 이름이 밑줄(예: **_wifi_set_power_level**)으로 시작하는 경우 **private** 함수를 고려하고 나중에 변경될 수 있으므로 스크립트에서 사용하지 마십시오.

플러그인 구성에서 **script** 매개 변수를 사용하여 실행 파일의 경로를 지정합니다.

예 3.6. 프로필에서 Bash 스크립트 실행

프로필 디렉터리에 있는 **script.sh** 라는 **Bash** 스크립트를 실행하려면 다음을 사용합니다.

```
[script]
script=${i:PROFILE_DIR}/script.sh
```

sysfs

플러그인 옵션으로 지정된 다양한 **sysfs** 설정을 설정합니다.

구문은 ***name=value*** 입니다. 여기서 ***name*** 은 사용할 **sysfs** 경로입니다.

다른 플러그인에서 다루지 않는 일부 설정을 변경해야 하는 경우 이 플러그인을 사용합니다. 필요한 설정을 다루는 경우 특정 플러그인을 선호합니다.

video

비디오 카드에 다양한 절전 수준을 설정합니다. 현재는 **Radeon** 카드만 지원됩니다.

절전 수준은 **radeon_powersave** 옵션을 사용하여 지정할 수 있습니다. 지원되는 값은 다음과 같습니다.

- **default**
- **auto**
- **low**
- **mid**
- 높음
- **dynpm**
- **dpm-battery**
- **dpm-balanced**
- **dpm-perfomance**

자세한 내용은 **www.x.org** 의 내용을 참조하십시오. 이 플러그인은 실험적이며 향후 릴리스에서 옵션이 변경될 수 있습니다.

bootloader

커널 명령줄에 옵션을 추가합니다. 이 플러그인은 **GRUB 2** 부트 로더만 지원합니다.

사용자 지정 **GRUB 2** 구성 파일의 비표준 위치는 **grub2_cfg_file** 옵션으로 지정할 수 있습니다.

커널 옵션이 현재 **GRUB** 설정과 해당 템플릿에 추가됩니다. 커널 옵션을 적용하려면 시스템을 다시 부팅해야 합니다.

다른 프로필로 전환하거나 **tuned** 서비스를 수동으로 중지하면 추가 옵션이 제거됩니다. 시스템을 종료하거나 재부팅하면 커널 옵션이 **grub.cfg** 파일에 유지됩니다.

커널 옵션은 다음 구문으로 지정할 수 있습니다.

```
cmdline=arg1 arg2 ... argN
```

예 3.7. 커널 명령줄 수정

예를 들어, **quiet** 커널 옵션을 **TuneD** 프로필에 추가하려면 **tuned.conf** 파일에 다음 행을 포함합니다.

```
[bootloader]
cmdline=quiet
```

다음은 커널 명령줄에 **isolcpus=2** 옵션을 추가하는 사용자 지정 프로필의 예입니다.

```
[bootloader]
cmdline=isolcpus=2
```

3.9. TUNED 프로필의 변수

TuneD 프로필이 활성화되면 변수는 런타임 시 확장됩니다.

TuneD 변수를 사용하면 **TuneD** 프로필에서 필요한 입력 양이 줄어듭니다.

TuneD 프로필에는 사전 정의된 변수가 없습니다. 프로필에 **[variables]** 섹션을 생성하고 다음 구문을

사용하여 자체 변수를 정의할 수 있습니다.

```
[variables]

variable_name=value
```

프로필의 변수 값을 확장하려면 다음 구문을 사용합니다.

```
${variable_name}
```

예 3.8. 변수를 사용하여 CPU 코어 격리

다음 예에서 **${isolated_cores}** 변수는 **1,2** 로 확장되므로 커널 은 **isolcpus=1,2** 옵션으로 부팅됩니다.

```
[variables]
isolated_cores=1,2

[bootloader]
cmdline=isolcpus=${isolated_cores}
```

변수는 별도의 파일에 지정할 수 있습니다. 예를 들어 **tuned.conf**에 다음 행을 추가할 수 있습니다.

```
[variables]
include=/etc/tuned/my-variables.conf

[bootloader]
cmdline=isolcpus=${isolated_cores}
```

isolated_cores=1,2 옵션을 **/etc/tuned/my-variables.conf** 파일에 추가하면 커널이 **isolcpus=1,2** 옵션으로 부팅됩니다.

추가 리소스

- **tuned.conf(5)** 도움말 페이지

3.10. TUNED 프로필의 기본 제공 함수

TuneD 프로필이 활성화되면 기본 제공 함수가 런타임에 확장됩니다.

다음을 수행할 수 있습니다.

- **TuneD** 변수와 함께 다양한 기본 제공 함수 사용
- **Python**에서 사용자 지정 함수를 만들고 플러그인 형식의 **TuneD** 에 추가합니다.

함수를 호출하려면 다음 구문을 사용하십시오.

```
${f:function_name:argument_1:argument_2}
```

프로필과 **tuned.conf** 파일이 있는 디렉터리 경로를 확장하려면 특수 구문이 필요한 **PROFILE_DIR** 함수를 사용합니다.

```
${i:PROFILE_DIR}
```

예 3.9. 변수 및 기본 제공 함수를 사용하여 CPU 코어 격리

다음 예에서 **${non_isolated_cores}** 변수는 **0,3-5** 로 확장되고 **cpulist_invert** 내장 함수가 **0,3-5** 인수를 사용하여 호출됩니다.

```
[variables]
non_isolated_cores=0,3-5

[bootloader]
cmdline=isolcpus=${f:cpulist_invert:${non_isolated_cores}}
```

cpulist_invert 함수는 **CPU** 목록을 되돌립니다. **6CPU** 시스템의 경우 **inversion**은 **1,2** 이고 커널은 **isolcpus=1,2** 명령줄 옵션으로 부팅됩니다.

추가 리소스

- **tuned.conf(5)** 도움말 페이지

3.11. TUNED 프로필에서 사용할 수 있는 기본 제공 함수

다음 기본 제공 함수는 모든 **TuneD** 프로필에서 사용할 수 있습니다.

PROFILE_DIR

프로필과 **tuned.conf** 파일이 있는 디렉터리 경로를 반환합니다.

exec

프로세스를 실행하고 해당 출력을 반환합니다.

assertion

는 두 인수를 비교합니다. *일치하지* 않으면 함수가 첫 번째 인수의 텍스트를 로그하고 프로파일 로드를 중단합니다.

assertion_non_equal

는 두 인수를 비교합니다. *일치하는* 경우 함수에서 첫 번째 인수의 텍스트를 로그하고 프로파일 로드를 중단합니다.

kb2s

는 킬로바이트를 디스크 섹터로 변환합니다.

s2kb

디스크 섹터를 킬로바이트로 변환합니다.

strip

전달된 모든 인수에서 문자열을 만들고 선행 및 후행 공백을 둘 다 삭제합니다.

virt_check

TuneD 가 **VM**(가상 머신) 내에서 실행 중인지 또는 베어 메탈에서 실행 중인지 확인합니다.

- **VM** 내에서 함수는 첫 번째 인수를 반환합니다.
- 베어 메탈에서 함수는 오류가 발생하더라도 두 번째 인수를 반환합니다.

cpulist_invert

는 **CPU** 목록을 변환하여 보완되도록 합니다. 예를 들어 **CPU**가 **4**개인 시스템에서 **0**에서 **3** 사이의 번호가 지정된 시스템에서 목록 **0,2,3** 의 버전이 **1** 입니다.

cpulist2hex

CPU 목록을 **16**진수 **CPU** 마스크로 변환합니다.

cpulist2hex_invert

CPU 목록을 **16**진수 **CPU** 마스크로 변환하고 역순으로 표시합니다.

hex2cpulist

16진수 **CPU** 마스크를 **CPU** 목록으로 변환합니다.

cpulist_online

목록의 **CPU**가 온라인 상태인지 확인합니다. 온라인 **CPU**만 포함하는 목록을 반환합니다.

cpulist_present

목록의 **CPU**가 있는지 확인합니다. 현재 **CPU**만 포함하는 목록을 반환합니다.

cpulist_unpack

1-3,4 에서 **1,2,3,4** 형식의 **CPU** 목록의 압축을 풉니다.

cpulist_pack

는 **1,2,3,5** 의 형식으로 **CPU** 목록을 **1-3,5** 형식으로 팩합니다.

3.12. 새 TUNED 프로필 생성

이 절차에서는 사용자 지정 성능 규칙을 사용하여 새 **TuneD** 프로필을 생성합니다.

사전 요구 사항

- **tuned** 서비스가 실행 중입니다. 자세한 내용은 **TuneD** 설치 및 활성화 를 참조하십시오.

절차

1. **/etc/tuned/** 디렉토리에서 생성할 프로필과 동일한 라는 새 디렉터리를 생성합니다.

```
# mkdir /etc/tuned/my-profile
```

2. 새 디렉터리에서 **tuned.conf** 라는 파일을 생성합니다. 요구 사항에 따라 **[main]** 섹션과 플러그인 정의를 추가합니다.

   예를 들어, **balanced** 프로필의 구성을 참조하십시오.

```
[main]
summary=General non-specialized tuned profile

[cpu]
governor=conservative
energy_perf_bias=normal

[audio]
timeout=10

[video]
radeon_powersave=dpm-balanced, auto

[scsi_host]
alpm=medium_power
```

3. 프로필을 활성화하려면 다음을 사용합니다.

```
# tuned-adm profile my-profile
```

4. **TuneD** 프로필이 활성 상태이고 시스템 설정이 적용되었는지 확인합니다.

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verfication succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

추가 리소스

- **tuned.conf(5)** 도움말 페이지

3.13. 기존 TUNED 프로필 수정

이 절차에서는 기존 **TuneD** 프로필을 기반으로 수정된 하위 프로필을 생성합니다.

사전 요구 사항

- **tuned** 서비스가 실행 중입니다. 자세한 내용은 **TuneD 설치 및 활성화** 를 참조하십시오.

절차

1. **/etc/tuned/** 디렉토리에서 생성할 프로필과 동일한 라는 새 디렉터리를 생성합니다.

```
# mkdir /etc/tuned/modified-profile
```

2. 새 디렉터리에서 **tuned.conf** 라는 파일을 생성하고 다음과 같이 **[main]** 섹션을 설정합니다.

```
[main]
include=parent-profile
```

parent-profile 을 수정할 프로필 이름으로 교체합니다.

3. 프로파일 수정 사항을 포함합니다.

예 3.10. throughput-performance 프로필에서 스왑성 감소

throughput-performance 프로필의 설정을 사용하고 기본 **10** 대신 **vm.swappiness** 값을 **5**로 변경하려면 다음을 사용합니다.

```
[main]
include=throughput-performance

[sysctl]
vm.swappiness=5
```

4. 프로필을 활성화하려면 다음을 사용합니다.

```
# tuned-adm profile modified-profile
```

5. **TuneD** 프로필이 활성 상태이고 시스템 설정이 적용되었는지 확인합니다.

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verfication succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

추가 리소스

- **tuned.conf(5)** 도움말 페이지

3.14. TUNED를 사용하여 디스크 스케줄러 설정

이 절차에서는 선택한 블록 장치에 지정된 디스크 스케줄러를 설정하는 **TuneD** 프로필을 생성하고 활성화합니다. 이 설정은 시스템을 재부팅해도 유지됩니다.

다음 명령 및 구성에서 교체합니다.

- 블록 *장치의* 이름이 있는 장치(예: **sdf**)
- 장치에 설정할 디스크 스케줄러가 있는 **select** ***-scheduler*** (예: **bfq**)

사전 요구 사항

- **tuned** 서비스가 설치 및 활성화되었습니다. 자세한 내용은 TuneD 설치 및 사용을 참조하십시오.

절차

1. 선택 사항: 프로필을 기반으로 할 기존 **TuneD** 프로필을 선택합니다. 사용 가능한 프로필 목록은 RHEL로 배포된 TuneD 프로필을 참조하십시오.

   현재 활성화된 프로필을 확인하려면 다음을 사용합니다.

```
$ tuned-adm active
```

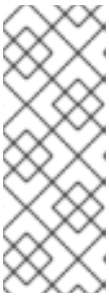
2. **TuneD** 프로필을 유지할 새 디렉토리를 만듭니다.

```
# mkdir /etc/tuned/my-profile
```

3. 선택한 블록 장치의 시스템 고유 식별자를 찾습니다.

```
$ udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'

ID_WWN=0x5002538d00000000_
ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
ID_SERIAL_SHORT=20120501030900000
```



참고

이 예제의 명령은 지정된 블록 장치와 연결된 **WWN(World Wide Name)** 또는 일련 번호로 식별된 모든 값을 반환합니다. **WWN**을 사용하는 것이 바람직하지만 **WWN**은 지정된 장치에 항상 사용 가능한 것은 아니며 예제 명령에서 반환된 값은 *장치 시스템 고유* ***ID*** 로 사용할 수 있습니다.

4. **/etc/tuned/*my-profile*/tuned.conf** 구성 파일을 만듭니다. 파일에서 다음 옵션을 설정합니다.

a. 선택 사항: 기존 프로필을 포함합니다.

```
[main]
include=existing-profile
```

b. **WWN** 식별자와 일치하는 장치에 대해 선택한 디스크 스케줄러를 설정합니다.

```
[disk]
devices_udev_regex=IDNAME=device system unique id
elevator=selected-scheduler
```

여기:

- ***IDNAME*** 을 사용 중인 식별자의 이름으로 바꿉니다(예: **ID_WWN**).

- *장치 시스템 고유 **ID*** 를 선택한 식별자 값으로 바꿉니다(예: **0x5002538d0000**).

  devices_udev_regex 옵션의 여러 장치를 일치시키려면 식별자를 괄호로 묶고 세로 막대로 구분합니다.

```
devices_udev_regex=(ID_WWN=0x5002538d00000000)|
(ID_WWN=0x1234567800000000)
```

5. 프로필을 활성화합니다.

```
# tuned-adm profile my-profile
```

검증 단계

1. **TuneD** 프로필이 활성화되어 적용되었는지 확인합니다.

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

2. **/sys/block/***장치***/queue/scheduler** 파일의 내용을 읽습니다.

```
# cat /sys/block/device/queue/scheduler

[mq-deadline] kyber bfq none
```

파일 이름에서 ***device*** 를 **exampledc**의 블록 장치 이름으로 바꿉니다.

활성 스케줄러는 대괄호(**[]**)로 나열됩니다.

추가 리소스

- **TuneD** 프로필 사용자 지정.

4장. TUNA 인터페이스를 사용하여 시스템 검토

tuna 툴을 사용하여 스케줄러 튜닝 가능 항목을 조정하고, 스레드 우선 순위, IRQ 핸들러를 조정하고, CPU 코어와 소켓을 격리합니다. tuna는 튜닝 작업 수행의 복잡성을 줄입니다.

tuna 툴은 다음 작업을 수행합니다.

- 시스템의 CPU 나열
- 시스템에서 현재 실행중인 인터럽트 요청 (IRQ) 나열
- 스레드의 정책 및 우선 순위 정보 변경
- 시스템의 현재 정책 및 우선 순위 표시

4.1. TUNA 툴 설치

tuna 도구는 실행 중인 시스템에서 사용하도록 설계되었습니다. 이를 통해 애플리케이션별 측정 툴은 변경이 완료된 직후 시스템 성능을 보고 분석할 수 있습니다.

다음 절차에서는 tuna 툴을 설치하는 방법을 설명합니다.

절차

- tuna 툴을 설치합니다.

```
# yum install tuna
```

검증 단계

- 사용 가능한 tuna CLI 옵션을 확인합니다.

```
# tuna -h
```

추가 리소스

- **tuna(8)** 도움말 페이지

4.2. TUNA 툴을 사용하여 시스템 상태 보기

이 절차에서는 **tuna CLI(** 명령줄 인터페이스) 툴을 사용하여 시스템 상태를 확인하는 방법을 설명합니다.

사전 요구 사항

- **tuna** 툴이 설치되어 있습니다. 자세한 내용은 **Installing tuna tool** 에서 참조하십시오.

절차

- 현재 정책 및 우선 순위를 보려면 다음을 수행합니다.

```
# tuna --show_threads
            thread
pid   SCHED_ rtpri affinity            cmd
1     OTHER     0      0,1           init
2      FIFO    99        0     migration/0
3     OTHER     0        0     ksoftirqd/0
4      FIFO    99        0      watchdog/0
```

- **PID**에 해당하거나 명령 이름과 일치하는 특정 스레드를 보려면 다음을 수행합니다.

```
# tuna --threads=pid_or_cmd_list --show_threads
```

pid_or_cmd_list 인수는 쉼표로 구분된 **PID** 또는 명령 이름 패턴 목록입니다.

- **tuna CLI** 를 사용하여 **CPU**를 조정하려면 **tuna** 툴을 사용하여 **CPU** 튜닝 을 참조하십시오.

- **tuna** 툴을 사용하여 **IRQ**를 조정하려면 **tuna** 도구를 사용하여 **IRQ** 튜닝을 참조하십시오.

- 변경된 구성을 저장하려면 다음을 수행합니다.

```
# tuna --save=filename
```

이 명령은 현재 실행 중인 커널 스레드만 저장합니다. 실행 중이 아닌 프로세스는 저장되지 않습니다.

추가 리소스

- tuna(8) 도움말 페이지

4.3. TUNA 툴을 사용하여 CPU 튜닝

tuna 툴 명령은 개별 CPU를 대상으로 할 수 있습니다.

tuna 도구를 사용하여 다음을 수행할 수 있습니다.

CPU 격리

지정된 CPU에서 실행되는 모든 작업은 사용 가능한 다음 CPU로 이동합니다. CPU를 격리하면 모든 스레드의 선호도 마스크에서 CPU를 제거하여 사용할 수 없습니다.

CPU 포함

지정된 CPU에서 작업을 실행할 수 있음

CPU 복원

는 지정된 CPU를 이전 구성으로 복원합니다.

다음 절차에서는 tuna CLI를 사용하여 CPU를 튜닝하는 방법을 설명합니다.

사전 요구 사항

- tuna 툴이 설치되어 있습니다. 자세한 내용은 Installing tuna tool 에서 참조하십시오.

절차

- 명령의 영향을 받을 CPU 목록을 지정하려면 다음을 수행합니다.

```
# tuna --cpus=cpu_list [command]
```

cpu_list 인수는 쉼표로 구분된 **CPU** 번호 목록입니다. 예를 들면 **--cpus=*0,2*** 입니다. **CPU** 목록도 범위 내에 지정할 수 있습니다(예: **--cpus="*1-3*", CPUs 1, 2, 3**를 선택하는).

현재 ***cpu_list*** 에 특정 **CPU**를 추가하려면 예를 들어 **--cpus=+0** 을 사용합니다.

[***command***]를 **--isolate** 으로 바꿉니다.

- **CPU**를 격리하려면 다음을 수행합니다.

```
# tuna --cpus=cpu_list --isolate
```

- **CPU**를 포함하려면 다음을 수행합니다.

```
# tuna --cpus=cpu_list --include
```

- **4**개 이상의 프로세서가 있는 시스템을 사용하려면 모든 **ssh** 스레드가 **CPU *0*** 및 ***1*** 에서 실행되도록 하는 방법과 **CPU *2*** 및 ***3*** 의 모든 **http** 스레드를 표시하는 방법은 다음과 같습니다.

```
# tuna --cpus=0,1 --threads=ssh\* \
--move --cpus=2,3 --threads=http\* --move
```

이 명령은 다음 작업을 순차적으로 수행합니다.

1. **CPU *0*** 과 ***1*** 을 선택합니다.
2. **ssh** 로 시작하는 모든 스레드를 선택합니다.
3. 선택한 스레드를 선택한 **CPU**로 이동합니다. **tuna**는 **ssh** 로 시작하는 스레드의 선호도 마스크를 적절한 **CPU**에 설정합니다. **CPU**는 ***0***과 ***1*** 로, ***16***진수 마스크에서 ***0*** **x3**으로, 또는 **11**으로 바이너리로 나타낼 수 있습니다.

4. **CPU** 목록을 ***2*** 및 ***3*** 으로 재설정합니다.

5. **http** 로 시작하는 모든 스레드를 선택합니다.

6. 선택한 스레드를 지정된 **CPU**로 이동합니다. **tuna**는 **http** 로 시작하는 스레드의 선호도 마스크를 지정된 **CPU**로 설정합니다. **CPU**는 ***2*** 및 ***3***, **16**진수 마스크에서 **0xC**로, 또는 **1100**으로 바이너리로 나타낼 수 있습니다.

검증 단계

- 현재 구성을 표시하고 변경 사항이 예상대로 수행되었는지 확인합니다.

```
# tuna --threads=gnome-sc\* --show_threads \
--cpus=0 --move --show_threads --cpus=1 \
--move --show_threads --cpus=+0 --move --show_threads

                thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
 3861   OTHER     0      0,1     33997          58 gnome-screensav
                thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
 3861   OTHER     0        0     33997          58 gnome-screensav
                thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
 3861   OTHER     0        1     33997          58 gnome-screensav
                thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
 3861   OTHER     0      0,1     33997          58 gnome-screensav
```

이 명령은 다음 작업을 순차적으로 수행합니다.

1. **gnome-sc** 스레드로 시작하는 모든 스레드를 선택합니다.

2. 사용자가 선호도 마스크 및 **RT** 우선 순위를 확인할 수 있도록 선택한 스레드를 표시합니다.

3. **CPU** ***0*** 을 선택합니다.

4. **gnome-sc 스레드를 지정된 CPU, CPU *0* 으로 이동합니다.**

5. **는 이동 결과를 표시합니다.**

6. **CPU 목록을 CPU *1* 로 재설정합니다.**

7. **gnome-sc 스레드를 지정된 CPU, CPU *1* 로 이동합니다.**

8. **이동 결과를 표시합니다.**

9. **CPU 목록에 CPU *0* 을 추가합니다.**

10. **gnome-sc 스레드를 지정된 CPU, CPU *0* 및 *1* 로 이동합니다.**

11. **이동 결과를 표시합니다.**

추가 리소스

- **/proc/cpuinfo 파일**
- **tuna(8) 도움말 페이지**

4.4. TUNA 툴을 사용하여 IRQ 조정

/proc/interrupts 파일은 IRQ당 인터럽트 수, 인터럽트 유형 및 해당 IRQ에 있는 장치의 이름을 기록합니다.

이 절차에서는 tuna 도구를 사용하여 IRQ를 조정하는 방법을 설명합니다.

사전 요구 사항

- **tuna** 툴이 설치되어 있습니다. 자세한 내용은 Installing tuna tool 에서 참조하십시오.

절차

- 현재 **IRQ** 및 해당 유사성을 보려면 다음을 수행합니다.

```
# tuna --show_irqs
# users          affinity
0 timer              0
1 i8042              0
7 parport0           0
```

- 명령의 영향을 받을 **IRQ** 목록을 지정하려면 다음을 수행합니다.

```
# tuna --irqs=irq_list [command]
```

irq_list 인수는 쉼표로 구분된 **IRQ** 번호 또는 사용자 이름 패턴 목록입니다.

[*command*]를 **--spread** 로 바꿉니다.

- 인터럽트를 지정된 **CPU**로 이동하려면 다음을 수행합니다.

```
# tuna --irqs=128 --show_irqs
  # users          affinity
128 iwlwifi          0,1,2,3

# tuna --irqs=128 --cpus=3 --move
```

128 을 **irq_list** 인수로 바꾸고 ***3*** 을 **cpu_list** 인수로 바꿉니다.

cpu_list 인수는 쉼표로 구분된 **CPU** 번호 목록입니다(예: **--cpus=*0,2*).** 자세한 내용은 tuna 툴을 사용하여 **CPU** 튜닝 을 참조하십시오.

검증 단계

- 인터럽트를 지정된 **CPU**로 이동하기 전후에 선택한 **IRQ**의 상태를 비교합니다.

```
# tuna --irqs=128 --show_irqs
  # users          affinity
128 iwlwifi              3
```

추가 리소스

- /**procs/interrupts** 파일
- **tuna(8)** 도움말 페이지

5장. RHEL 시스템 역할을 사용하여 성능 모니터링

시스템 관리자는 **Metrics RHEL** 시스템 역할을 **Ansible Automation Platform** 제어 노드와 함께 사용하여 시스템의 성능을 모니터링할 수 있습니다.

5.1. RHEL 시스템 역할 소개

RHEL 시스템 역할은 **Ansible** 역할 및 모듈의 컬렉션입니다. **RHEL** 시스템 역할은 여러 **RHEL** 시스템을 원격으로 관리하는 구성 인터페이스를 제공합니다. 이 인터페이스를 사용하면 여러 버전의 **RHEL**에서 시스템 구성을 관리하고 새 주요 릴리스를 채택할 수 있습니다.

Red Hat Enterprise Linux 8에서 인터페이스는 현재 다음 역할로 구성됩니다.

- 인증서 발행 및 업데이트
- **Cockpit**
- **firewalld**
- **HA** 클러스터
- 커널 덤프
- 커널 설정
- 로깅
- 메트릭 **(PCP)**
- **Microsoft SQL Server**

- 네트워킹
- 네트워크 **Bound Disk Encryption** 클라이언트 및 네트워크 **Bound Disk Encryption** 서버
- **Postfix**
- **SELinux**
- **SSH** 클라이언트
- **SSH** 서버
- 스토리지
- 터미널 세션 레코드
- 시간 동기화
- **VPN**

이러한 모든 역할은 **AppStream** 리포지토리에서 사용할 수 있는 **rhel-system-roles** 패키지에서 제공합니다.

추가 리소스

- RHEL (Red Hat Enterprise Linux) 시스템 역할
- **/usr/share/doc/rhel-system-roles/** 디렉터리에 있는 문서 [1]

5.2. RHEL 시스템 역할 용어

이 문서에서는 다음 용어를 찾을 수 있습니다.

Ansible 플레이북

플레이북은 **Ansible**의 구성, 배포 및 오케스트레이션 언어입니다. 원격 시스템이 강제 적용하려는 정책 또는 일반적인 **IT** 프로세스의 단계 집합을 설명할 수 있습니다.

제어 노드

Ansible이 설치된 모든 시스템. 모든 제어 노드에서 **/usr/bin/ansible** 또는 **/usr/bin/ansible-playbook**을 호출하여 명령과 플레이북을 실행할 수 있습니다. **Python**이 제어 노드로 설치된 컴퓨터를 사용할 수 있습니다. 노트북, 공유 데스크탑 및 서버는 모두 **Ansible**을 실행할 수 있습니다. 그러나 **Windows** 머신을 제어 노드로 사용할 수는 없습니다. 여러 제어 노드가 있을 수 있습니다.

인벤토리

관리형 노드 목록. 인벤토리 파일을 **"hostfile"**이라고도 합니다. 인벤토리는 각 관리 노드의 **IP** 주소와 같은 정보를 지정할 수 있습니다. 인벤토리는 쉽게 확장하기 위해 관리 노드, 생성 및 중첩 그룹을 구성할 수도 있습니다. 인벤토리에 대한 자세한 내용은 인벤토리 작업 섹션을 참조하십시오.

관리형 노드

Ansible을 사용하여 관리하는 네트워크 장치, 서버 또는 둘 다입니다. 관리되는 노드를 "호스트"라고도 합니다. **Ansible**은 관리 노드에 설치되지 않습니다.

5.3. 시스템에서 RHEL 시스템 역할 설치

RHEL 시스템 역할을 사용하려면 시스템에 필수 패키지를 설치하십시오.

사전 요구 사항

- **Ansible Core** 패키지는 제어 시스템에 설치됩니다.
- 제어 노드로 사용하려는 시스템에 **Ansible** 패키지가 설치되어 있습니다.

절차

1. 제어 노드로 사용할 시스템에 **rhel-system-roles** 패키지를 설치합니다.

```
# yum install rhel-system-roles
```

2. **Ansible Core** 패키지를 설치합니다.

```
# yum install ansible-core
```

Ansible Core 패키지는 **ansible-playbook CLI, Ansible Vault** 기능 및 **RHEL Ansible** 콘텐츠에 필요한 기본 모듈 및 필터를 제공합니다.

결과적으로 **Ansible** 플레이북을 생성할 수 있습니다.

추가 리소스

- **RHEL(Red Hat Enterprise Linux)** 시스템 역할
- **ansible-playbook** 도움말 페이지.

5.4. 역할 적용

다음 절차에서는 특정 역할을 적용하는 방법을 설명합니다.

사전 요구 사항

- 제어 노드로 사용하려는 시스템에 **rhel-system-roles** 패키지가 설치되어 있는지 확인합니다.

```
# yum install rhel-system-roles
```

1. **Ansible Core** 패키지를 설치합니다.

```
# yum install ansible-core
```

Ansible Core 패키지는 **ansible-playbook CLI, Ansible Vault** 기능 및 **RHEL Ansible** 콘텐츠에 필요한 기본 모듈 및 필터를 제공합니다.

- **Ansible** 인벤토리를 생성할 수 있는지 확인합니다.

인벤토리는 **Ansible** 플레이북에서 사용하는 호스트, 호스트 그룹 및 일부 구성 매개 변수를 나타냅니다.

플레이북은 일반적으로 사람이 읽을 수 있으며 **ini,yaml,json** 및 기타 파일 형식으로 정의됩니다.

- **Ansible** 플레이북을 생성할 수 있는지 확인합니다.

플레이북은 **Ansible**의 구성, 배포 및 오케스트레이션 언어를 나타냅니다. 플레이북을 사용하면 원격 시스템의 구성을 선언 및 관리하고, 여러 개의 원격 시스템을 배포하거나 수동 주문 프로세스의 단계를 오케스트레이션할 수 있습니다.

플레이북은 하나 이상의 플레이 목록입니다. 모든 플레이 에는 **Ansible** 변수, 작업 또는 역할이 포함될 수 있습니다.

플레이북은 사람이 읽을 수 있으며 **yaml** 형식으로 정의됩니다.

절차

1. 관리할 호스트 및 그룹을 포함하는 필수 **Ansible** 인벤토리를 생성합니다. 다음은 **webservers:**이라는 호스트 그룹의 **inventory.ini** 파일을 사용하는 예입니다.

```
[webservers]
host1
host2
host3
```

2. 필요한 역할을 포함하여 **Ansible** 플레이북을 생성합니다. 다음 예는 플레이북에 **roles:** 옵션을 통해 역할을 사용하는 방법을 보여줍니다.

다음 예제는 지정된 플레이에 대한 **roles:** 옵션을 통해 역할을 사용하는 방법을 보여줍니다.

```
---
- hosts: webservers
  roles:
     - rhel-system-roles.network
     - rhel-system-roles.timesync
```

참고

모든 역할에는 역할 사용 방법과 지원되는 매개 변수 값을 설명하는 README 파일이 포함되어 있습니다. 역할의 설명서 디렉터리에서 특정 역할에 대한 플레이북 예제도 찾을 수 있습니다. 이러한 문서 디렉터리는 rhel-system-roles 패키지와 함께 기본적으로 제공되며 다음 위치에서 찾을 수 있습니다.

```
/usr/share/doc/rhel-system-roles/SUBSYSTEM/
```

SUBSYSTEM **을 selinux,kdump,network,timesync 또는 storage 와 같은 필수 역할의 이름으로 바꿉니다.**

3. **특정 호스트에서 플레이북을 실행하려면 다음 중 하나를 수행해야 합니다.**

- **hosts: host1[,host2,…을 사용하도록 플레이북을 편집합니다.] 또는 호스트: all 및 명령을 실행합니다.**

```
# ansible-playbook name.of.the.playbook
```

- **인벤토리를 편집하여 사용할 호스트가 그룹에 정의되어 있는지 확인하고 명령을 실행합니다.**

```
# ansible-playbook -i name.of.the.inventory name.of.the.playbook
```

- **ansible-playbook 명령을 실행할 때 모든 호스트를 지정합니다.**

```
# ansible-playbook -i host1,host2,... name.of.the.playbook
```

중요

i 플래그는 사용 가능한 모든 호스트의 인벤토리를 지정합니다. 여러 대상 호스트가 있지만 플레이북을 실행할 호스트를 선택하려는 경우 플레이북에 변수를 추가하여 호스트를 선택할 수 있습니다. 예를 들면 다음과 같습니다.

```
Ansible Playbook | example-playbook.yml:

- hosts: "{{ target_host }}"
  roles:
    - rhel-system-roles.network
    - rhel-system-roles.timesync
```

플레이북 실행 명령:

```
# ansible-playbook -i host1,..hostn -e target_host=host5 example-
playbook.yml
```

추가 리소스

- Ansible 플레이북
- Ansible 플레이북에서 역할 사용
- Ansible 플레이북의 예
- 인벤토리를 만들고 작업하는 방법은 무엇입니까?
- ansible-playbook 툴

5.5. 지표 시스템 역할 소개

RHEL 시스템 역할은 여러 **RHEL** 시스템을 원격으로 관리하는 일관된 구성 인터페이스를 제공하는 **Ansible** 역할 및 모듈의 컬렉션입니다. **Metrics** 시스템 역할은 로컬 시스템에 대한 성능 분석 서비스를 구성하고, 선택적으로 로컬 시스템에서 모니터링할 원격 시스템 목록을 포함합니다. 지표 시스템 역할을 사용하면 **pcp** 를 개별적으로 구성하지 않고도 **pcp**를 사용하여 플레이북에서 설정 및 배포를 처리하므로 **pcp** 를 사용하여 시스템 성능을 모니터링할 수 있습니다.

표 5.1. 메트릭 시스템 역할 변수

역할 변수	설명	사용 예
metrics_monitored_hosts	대상 호스트에서 분석할 원격 호스트 목록입니다. 이러한 호스트에는 대상 호스트에 기록된 메트릭이 있으므로 각 호스트의 **/var/log** 아래에 디스크 공간이 충분히 있는지 확인합니다.	**metrics_monitored_hosts: ["*webserver.example.com*", "*database.example.com*"]**
metrics_retention_days	삭제하기 전에 성능 데이터 보존을 위한 일 수를 구성합니다.	**metrics_retention_days: 14**
metrics_graph_service	**pcp** 및 **grafana** 를 통해 성능 데이터 시각화를 위해 서비스를 사용하여 호스트를 설정할 수 있는 부울 플래그입니다. 기본적으로 false로 설정합니다.	**metrics_graph_service: no**
metrics_query_service	**redis** 를 통해 기록된 **pcp** 지표를 쿼리하기 위해 시계열 쿼리 서비스로 호스트를 설정할 수 있는 부울 플래그입니다. 기본적으로 false로 설정합니다.	**metrics_query_service: no**
metrics_provider	지표를 제공하는 데 사용할 지표 수집기를 지정합니다. 현재는 **pcp** 가 지원되는 유일한 지표 프로바이더입니다.	**metrics_provider: "pcp"**

참고

metrics_connections 에 사용되는 매개 변수 및 지표 시스템 역할에 대한 자세한 내용은 /usr/share/ansible/roles/rhel-system-roles.metrics/README.md 파일을 참조하십시오.

5.6. 시각화로 로컬 시스템을 모니터링하기 위해 메트릭 시스템 역할을 사용

다음 절차에서는 Metrics RHEL 시스템 역할을 사용하여 Grafana 를 통해 동시에 데이터 시각화를 프로비저닝하는 동시에 로컬 시스템을 모니터링하는 방법을 설명합니다.

사전 요구 사항

- **Ansible Core 패키지는 제어 시스템에 설치됩니다.**

- 모니터링할 시스템에 **rhel-system-roles** 패키지가 설치되어 있습니다.

절차

1. 인벤토리에 다음 콘텐츠를 추가하여 **/etc/ansible/hosts Ansible** 인벤토리에서 **localhost** 를 구성합니다.

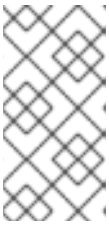
```
localhost ansible_connection=local
```

2. 다음 내용으로 **Ansible** 플레이북을 생성합니다.

```
---
- hosts: localhost
  vars:
    metrics_graph_service: yes
  roles:
    - rhel-system-roles.metrics
```

3. **Ansible** 플레이북을 실행합니다.

```
# ansible-playbook name_of_your_playbook.yml
```



참고

metrics_graph_service 부울이 **value="yes"**로 설정되어 있으므로 **Grafana** 는 데이터 소스로 **pcp** 를 추가하여 자동으로 설치되고 프로비저닝됩니다.

4. 시스템에서 수집 중인 지표의 시각화를 보려면 **Grafana 웹 UI 액세스에** 설명된 대로 **grafana 웹** 인터페이스에 액세스합니다.

5.7. 지표 시스템 역할을 사용하여 자신을 모니터링하기 위해 개별 시스템 세트를 설정

이 절차에서는 메트릭을 사용하여 모니터할 시스템 제품군을 설정하는 데 **Metrics** 시스템 역할을 사용하는 방법을 설명합니다.

사전 요구 사항

- **Ansible Core** 패키지는 제어 시스템에 설치됩니다.
- 플레이북을 실행하는 데 사용할 시스템에 **rhel-system-roles** 패키지가 설치되어 있습니다.
- **SSH** 연결이 설정되어 있습니다.

절차

1. 플레이북을 통해 모니터링할 시스템의 이름 또는 **IP**를 괄호로 묶은 식별 그룹 이름으로 **/etc/ansible/hosts Ansible** 인벤토리 파일에 추가합니다.

```
[remotes]
webserver.example.com
database.example.com
```

2. 다음 내용으로 **Ansible** 플레이북을 생성합니다.

```
---
- hosts: remotes
  vars:
    metrics_retention_days: 0
  roles:
    - rhel-system-roles.metrics
```

3. **Ansible** 플레이북을 실행합니다.

```
# ansible-playbook name_of_your_playbook.yml -k
```

여기서 **-k** 는 원격 시스템에 연결하기 위한 암호를 묻습니다.

5.8. 메트릭 시스템 역할을 사용하여 로컬 시스템을 통해 중앙 집중식으로 시스템 그룹을 모니터링할 수 있습니다.

이 절차에서는 **Metrics** 시스템 역할을 사용하여 시스템을 중앙에서 모니터링하도록 로컬 머신을 설정하는 방법을 설명하고, **grafana** 를 통해 데이터의 시각화를 프로비저닝하고 **redis** 를 통해 데이터를 쿼리합니다.

사전 요구 사항

- **Ansible Core** 패키지는 제어 시스템에 설치됩니다.
- 플레이북을 실행하는 데 사용할 시스템에 **rhel-system-roles** 패키지가 설치되어 있습니다.

절차

1. 다음 내용으로 **Ansible** 플레이북을 생성합니다.

```
---
- hosts: localhost
  vars:
    metrics_graph_service: yes
    metrics_query_service: yes
    metrics_retention_days: 10
    metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
  roles:
    - rhel-system-roles.metrics
```

2. **Ansible** 플레이북을 실행합니다.

```
# ansible-playbook name_of_your_playbook.yml
```

참고

metrics_graph_service 및 **metrics_query_service** 부울은 **value="yes"**로 설정되어 있으므로 **grafana** 는 **pcp** 데이터 기록이 **redis** 에 인덱싱된 데이터 소스로 추가된 **pcp** 쿼리 언어를 자동으로 설치 및 프로비저닝하므로 **pcp** 쿼리 언어를 복잡한 데이터 쿼리에 사용할 수 있습니다.

3. 시스템에서 중앙에서 수집 중인 지표의 그래픽 표시를 보고 데이터를 쿼리하려면 **Grafana 웹 UI 액세스에 설명된 대로 grafana 웹** 인터페이스에 액세스합니다.

5.9. METRICS 시스템 역할을 사용하여 시스템을 모니터링하는 동안 인증 설정

PCP는 **SASL(Simple Authentication Security Layer)** 프레임워크를 통해 **scram-sha-256** 인증 메커니즘을 지원합니다. **Metrics RHEL** 시스템 역할은 **scram-sha-256** 인증 메커니즘을 사용하여 인증을 설정하는 단계를 자동화합니다. 다음 절차에서는 **Metrics RHEL** 시스템 역할을 사용하여 인증을 설정하는 방법을 설명합니다.

사전 요구 사항

- **Ansible Core** 패키지는 제어 시스템에 설치됩니다.
- 플레이북을 실행하는 데 사용할 시스템에 **rhel-system-roles** 패키지가 설치되어 있습니다.

절차

1. 인증을 설정하려는 **Ansible** 플레이북에 다음 변수를 포함합니다.

```
---
 vars:
   metrics_username: your_username
   metrics_password: your_password
```

2. **Ansible** 플레이북을 실행합니다.

```
# ansible-playbook name_of_your_playbook.yml
```

검증 단계

- **the sasl** 구성을 확인합니다.

```
# pminfo -f -h "pcp://ip_adress?username=your_username" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

ip_adress 는 호스트의 **IP** 주소로 교체해야 합니다.

5.10. METRICS 시스템 역할을 사용하여 SQL SERVER에 대한 메트릭 컬렉션을 구성하고 활성화합니다.

이 절차에서는 **Metrics RHEL** 시스템 역할을 사용하여 로컬 시스템의 **pcp** 를 통해 **Microsoft SQL Server**에 대한 메트릭 컬렉션 및 구성을 자동화하는 방법을 설명합니다.

사전 요구 사항

- **Ansible Core** 패키지는 제어 시스템에 설치됩니다.

- 모니터링할 시스템에 **rhel-system-roles** 패키지가 설치되어 있습니다.

- **Microsoft SQL Server for Red Hat Enterprise Linux**를 설치하고 **SQL** 서버에 '신뢰할 수 있는' 연결을 구축했습니다. **SQL Server** 설치를 참조하고 **Red Hat**에 데이터베이스를 만듭니다.

- **Red Hat Enterprise Linux**용 **SQL Server**용 **Microsoft ODBC** 드라이버를 설치했습니다. **Red Hat Enterprise Server** 및 **Oracle Linux** 를 참조하십시오.

절차

1. 인벤토리에 다음 콘텐츠를 추가하여 **/etc/ansible/hosts Ansible** 인벤토리에서 **localhost** 를 구성합니다.

```
localhost ansible_connection=local
```

2. 다음 콘텐츠가 포함된 **Ansible** 플레이북을 생성합니다.

```
---
- hosts: localhost
  roles:
    - role: rhel-system-roles.metrics
      vars:
        metrics_from_mssql: yes
```

3. **Ansible** 플레이북을 실행합니다.

```
# ansible-playbook name_of_your_playbook.yml
```

검증 단계

- **pcp** 명령을 사용하여**mssql(SQL** 서버 **PMDA** 에이전트)이 로드되고 실행 중인지 확인합니다.

```
# pcp
platform: Linux rhel82-2.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC
2019 x86_64
 hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
 timezone: PDT+7
 services: pmcd pmproxy
     pmcd: Version 5.0.2-1, 12 agents, 4 clients
```

```
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
          jbd2 dm
pmlogger: primary logger: /var/log/pcp/pmlogger/rhel82-2.local/20200326.16.31
   pmie: primary engine: /var/log/pcp/pmie/rhel82-2.local/pmie.log
```

추가 리소스

- **Microsoft SQL Server용 Performance Co-Pilot 사용에 대한 자세한 내용은 이 Red Hat Developers 블로그 게시물을 참조하십시오.**

---

[1] 이 문서는 **rhel-system-roles** 패키지를 사용하여 자동으로 설치됩니다.

6장. PCP 설정

PCP(Performance Co-Pilot)는 시스템 수준의 성능 측정을 모니터링, 시각화, 저장 및 분석하기 위한 툴, 서비스 및 라이브러리 제품군입니다.

이 섹션에서는 시스템에 PCP를 설치하고 활성화하는 방법을 설명합니다.

6.1. PCP 개요

Python, Perl, C++ 및 C 인터페이스를 사용하여 성능 지표를 추가할 수 있습니다. 분석 도구는 Python, C++, C 클라이언트 API를 직접 사용할 수 있으며 리치 웹 애플리케이션은 JSON 인터페이스를 사용하여 사용 가능한 모든 성능 데이터를 탐색할 수 있습니다.

실시간 결과를 아카이브된 데이터와 비교하여 데이터 패턴을 분석할 수 있습니다.

PCP의 특징:

- 복잡한 시스템의 중앙 집중식 분석 중에 유용한 경량 분산 아키텍처.
- 실시간 데이터의 모니터링 및 관리를 가능하게 합니다.
- 기록 데이터를 로깅 및 검색할 수 있습니다.

PCP에는 다음과 같은 구성 요소가 있습니다.

- pmcd(성능 지표 수집기 데몬)는 설치된 pmda(성능 지표 도메인 에이전트)에서 성능 데이터를 수집합니다. PMDA 는 개별적으로 시스템에 로드 또는 언로드할 수 있으며 동일한 호스트의 PMCD 에 의해 제어됩니다.
- pminfo 또는 pm stat 와 같은 다양한 클라이언트 도구는 동일한 호스트 또는 네트워크에서 이 데이터를 검색, 표시, 아카이브 및 처리할 수 있습니다.

- pcp 패키지는 명령줄 도구 및 기본 기능을 제공합니다.
- pcp-gui 패키지는 그래픽 애플리케이션을 제공합니다. yum install pcp-gui 명령을 실행하여 pcp-gui 패키지를 설치합니다. 자세한 내용은 PCP Charts 애플리케이션을 사용하여 시각적으로 PCP 로그 아카이브 추적을 참조하십시오.

추가 리소스

- PCP(1) 도움말 페이지
- /usr/share/doc/pcp-doc/ directory
- PCP로 배포되는 툴
- Red Hat 고객 포털의 PCP(Performance Co-Pilot) 문서, 솔루션, 자습서 및 백서 인덱스
- 레거시 툴과 PCP 툴 나란히 비교 Red Hat Knowledgebase 문서
- PCP 업스트림 문서

6.2. PCP 설치 및 활성화

PCP 사용을 시작하려면 필요한 모든 패키지를 설치하고 PCP 모니터링 서비스를 활성화합니다.

다음 절차에서는 pcp 패키지를 사용하여 PCP를 설치하는 방법을 설명합니다. PCP 설치를 자동화하려면 pcp-zeroconf 패키지를 사용하여 설치합니다. pcp-zeroconf 를 사용하여 PCP를 설치하는 방법에 대한 자세한 내용은 pcp-zeroconf로 PCP 설정을 참조하십시오.

절차

1. pcp 패키지를 설치합니다.

```
# yum install pcp
```

2. 호스트 시스템에서 **pmcd** 서비스를 활성화하고 시작합니다.

```
# systemctl enable pmcd

# systemctl start pmcd
```

검증 단계

- **pmcd** 프로세스가 호스트에서 실행 중인지 확인합니다.

```
# pcp

Performance Co-Pilot configuration on workstation:

platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents
pmda: root pmcd proc xfs linux mmv kvm jbd2
```

추가 리소스

- **pmcd(1)** 도움말 페이지
- PCP로 배포되는 툴

6.3. 최소 PCP 설정 배포

최소 **PCP** 설정은 **Red Hat Enterprise Linux**에 대한 성능 통계를 수집합니다. 설정에는 추가 분석을 위해 데이터를 수집하는 데 필요한 프로덕션 시스템에 최소 패키지 수를 추가해야 합니다.

결과 **tar.gz** 파일과 다양한 **PCP** 툴을 사용하여 **pmlogger** 출력 아카이브를 분석하고 다른 성능 정보 소스와 비교할 수 있습니다.

사전 요구 사항

- **PCP**가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.

절차

1. **pmlogger** 구성을 업데이트합니다.

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
```

2. **pmcd** 및 **pmlogger** 서비스를 시작합니다.

```
# systemctl start pmcd.service

# systemctl start pmlogger.service
```

3. 필요한 작업을 실행하여 성능 데이터를 기록합니다.

4. **pmcd** 및 **pmlogger** 서비스를 중지합니다.

```
# systemctl stop pmcd.service

# systemctl stop pmlogger.service
```

5. 출력을 저장하고 호스트 이름과 현재 날짜 및 시간을 기준으로 라는 **tar.gz** 파일에 저장합니다.

```
# cd /var/log/pcp/pmlogger/

# tar -czf $(hostname).$(date +%F-%Hh%M).pcp.tar.gz $(hostname)
```

이 파일을 추출하고 **PCP** 도구를 사용하여 데이터를 분석합니다.

추가 리소스

- **pmlogconf(1), pmlogger(1)** 및 **pmcd(1)** 도움말 페이지
- PCP로 배포되는 툴
- PCP와 함께 배포되는 시스템 서비스

6.4. PCP와 함께 배포되는 시스템 서비스

다음 표에서는 PCP와 함께 배포되는 다양한 시스템 서비스의 역할을 설명합니다.

표 6.1. PCP와 함께 배포되는 시스템 서비스 역할

이름	설명
pmcd	PMCD(성능 지표 수집기 데몬).
pmie	성능 지표 추론 엔진.
pmlogger	성능 지표 로거.
pmproxy	실시간 및 기록 성능 지표 프록시, 시계열 쿼리 및 REST API 서비스.

6.5. PCP로 배포되는 툴

다음 표에서는 PCP와 함께 배포되는 다양한 도구의 사용법을 설명합니다.

표 6.2. PCP와 함께 배포되는 툴 사용

이름	설명
pcp	Performance Co-Pilot 설치의 현재 상태를 표시합니다.
pcp-atop	성능 관점에서 가장 중요한 하드웨어 리소스의 시스템 수준 표시. CPU, 메모리, 디스크 및 네트워크.
pcp-atopsar	다양한 시스템 리소스 활용도에 대한 시스템 수준 활동 보고서를 생성합니다. 이 보고서는 pmlogger 또는 pcp-atop의 -w 옵션을 사용하여 이전에 기록된 원시 로그 파일에서 생성됩니다.
pcp-dmcache	장치 IOP, 캐시 및 메타데이터 장치 사용률과 같은 구성된 장치 매퍼 캐시 대상에 대한 정보와 각 캐시 장치의 읽기 및 쓰기에 대한 적중률 및 누락 비율 정보가 표시됩니다.
pcp-dstat	한 번에 하나의 시스템의 지표를 표시합니다. 여러 시스템의 지표를 표시하려면 **--host** 옵션을 사용합니다.
pcp-free	시스템의 사용 가능한 메모리와 사용 중인 메모리에 대해 보고합니다.

pcp-htop	**top** 명령과 유사하게 명령줄 인수와 함께 시스템에서 실행되는 모든 프로세스를 표시하지만 마우스를 사용하여 수직 및 수평으로 스크롤할 수 있습니다. 또한 트리 형식으로 프로세스를 보고 여러 프로세스에 대해 한 번에 선택하고 실행할 수 있습니다.
pcp-ipcs	은 호출 프로세스가 읽기 액세스 권한이 있는 프로세스 간 통신(IPC) 기능에 대한 정보를 표시합니다.
pcp-numastat	커널 메모리 할당 도우미의 NUMA 할당 통계를 표시합니다.
pcp-pidstat	다음과 같은 시스템에서 실행되는 개별 작업 또는 프로세스에 대한 정보를 표시합니다. CPU 백분율, 메모리 및 스택 사용량, 스케줄링 및 우선 순위. 기본적으로 로컬 호스트에 대한 라이브 데이터를 보고합니다.
pcp-ss	pmdasockets PMDA(성능 지표 도메인 에이전트)에서 수집한 소켓 통계를 표시합니다.
pcp-uptime	는 시스템이 실행된 시간, 현재 로그인한 사용자 수 및 지난 1분 5분, 15분 동안의 시스템 부하 평균을 표시합니다.
pcp-vmstat	5초마다 높은 수준의 시스템 성능 개요를 제공합니다. 프로세스, 메모리, 페이징, 블록 IO, 트랩 및 CPU 활동에 대한 정보를 표시합니다.
pmchart	Performance Co-Pilot 기능을 통해 사용할 수 있는 성능 지표 값을 나타냅니다.
pmclient	PMAPI(성능 지표 애플리케이션 프로그래밍 인터페이스)를 사용하여 고급 시스템 성능 지표를 표시합니다.
pmconfig	구성 매개 변수의 값을 표시합니다.
pmdbg	사용 가능한 Performance Co-Pilot 디버그 제어 플래그 및 해당 값을 표시합니다.
pmdiff	지정된 시간 창에 있는 하나 또는 두 개의 아카이브에 있는 모든 지표의 평균 값을 비교하여 성능 회귀를 검색할 때 관심을 가질 수 있는 변경 사항을 비교합니다.
pmdumplog	Performance Co-Pilot 아카이브 파일의 제어, 메타데이터, 인덱스 및 상태 정보를 표시합니다.
pmdumptext	실시간 또는 Performance Co-Pilot 아카이브에서 수집된 성능 지표 값을 출력합니다.

pmerr	사용 가능한 Performance Co-Pilot 오류 코드 및 해당 오류 메시지를 표시합니다.
pmfind	네트워크에서 PCP 서비스를 찾습니다.
pmie	산술, 논리적 및 규칙 표현식 집합을 주기적으로 평가하는 유추 엔진. 지표는 실시간 시스템 또는 Performance Co-Pilot 아카이브 파일에서 수집됩니다.
pmieconf	구성 가능한 pmie 변수를 표시하거나 설정합니다.
pmiectl	pmie의 기본이 아닌 인스턴스를 관리합니다.
pminfo	성능 지표에 대한 정보를 표시합니다. 지표는 실시간 시스템 또는 Performance Co-Pilot 아카이브 파일에서 수집됩니다.
pmiostat	SCSI 장치(기본값) 또는 장치 매퍼 장치에 대한 I/O 통계를 보고합니다(-x dm 옵션 사용).
pmlc	대화형으로 활성 pmlogger 인스턴스를 구성합니다.
pmlogcheck	Performance Co-Pilot 아카이브 파일에서 잘못된 데이터를 식별합니다.
pmlogconf	pmlogger 구성 파일을 만들고 수정합니다.
pmlogctl	pmlogger의 기본이 아닌 인스턴스를 관리합니다.
pmloglabel	Performance Co-Pilot 아카이브 파일의 레이블을 확인, 수정 또는 복구합니다.
pmlogsummary	Performance Co-Pilot 아카이브 파일에 저장된 성능 지표에 대한 통계 정보를 계산합니다.
pmprobe	성능 지표의 가용성을 결정합니다.
pmrep	선택한 사용자 지정이 쉽고 성능 지표 값에 대한 보고서.
pmsocks	방화벽을 통해 Performance Co-Pilot 호스트에 액세스 가능.
pmstat	주기적으로 시스템 성능 요약을 표시합니다.
pmstore	성능 지표의 값을 수정합니다.
pmtrace	추적 PMDA에 대한 명령줄 인터페이스를 제공합니다.

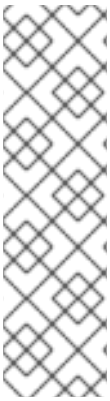
pmval	는 성능 지표의 현재 값을 표시합니다.

6.6. PCP 배포 아키텍처

PCP(Performance Co-Pilot)는 고급 설정을 수행할 수 있는 다양한 옵션을 제공합니다. 다양한 가능한 아키텍처에서 이 섹션에서는 **Red Hat,** 크기 조정 요소 및 구성 옵션으로 설정된 권장 배포를 기반으로 **PCP** 배포를 확장하는 방법을 설명합니다.

PCP는 **PCP** 배포 규모에 따라 여러 배포 아키텍처를 지원합니다.

사용 가능한 확장 배포 설정 변형:



참고

PCP 버전 **5.3.0**은 **Red Hat Enterprise Linux 8.4** 및 이전 마이너 버전에서 사용할 수 없으므로 **Red Hat**은 **localhost** 및 **pmlogger** 팜 아키텍처를 권장합니다.

5.3.0 이전의 **PCP** 버전의 **pmproxy**에서 알려진 메모리 누수에 대한 자세한 내용은 **PCP**의 **pmproxy**에서 메모리 누수를 참조하십시오.

localhost

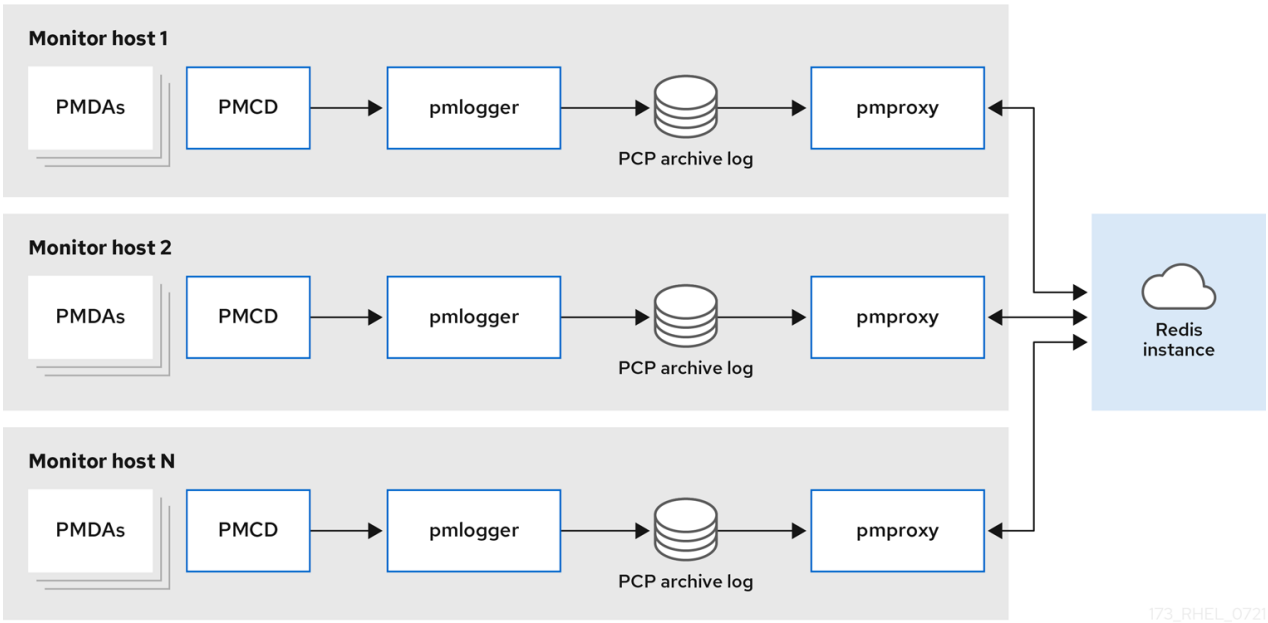
각 서비스는 모니터링된 시스템에서 로컬로 실행됩니다. 구성을 변경하지 않고 서비스를 시작하면 기본 배포가 됩니다. 이 경우 개별 노드를 초과하여 확장할 수 없습니다.

기본적으로 **Redis**의 배포 설정은 독립 실행형 **localhost**입니다. 그러나 **Redis**는 여러 호스트에서 데이터를 공유하는 고가용성 및 확장성이 뛰어난 클러스터형 방식으로 선택적으로 수행할 수 있습니다. 또 다른 실행 가능한 옵션은 클라우드에 **Redis** 클러스터를 배포하거나 클라우드 벤더의 관리형 **Redis** 클러스터를 활용하는 것입니다.

분산

localhost와 분산 설정의 유일한 차이점은 중앙 집중식 **Redis** 서비스입니다. 이 모델에서 호스트는 모니터링되는 각 호스트에서 **pmlogger** 서비스를 실행하고 로컬 **pmcd** 인스턴스에서 지표를 검색합니다. 그런 다음 로컬 **pmproxy** 서비스는 성능 지표를 중앙 **Redis** 인스턴스로 내보냅니다.

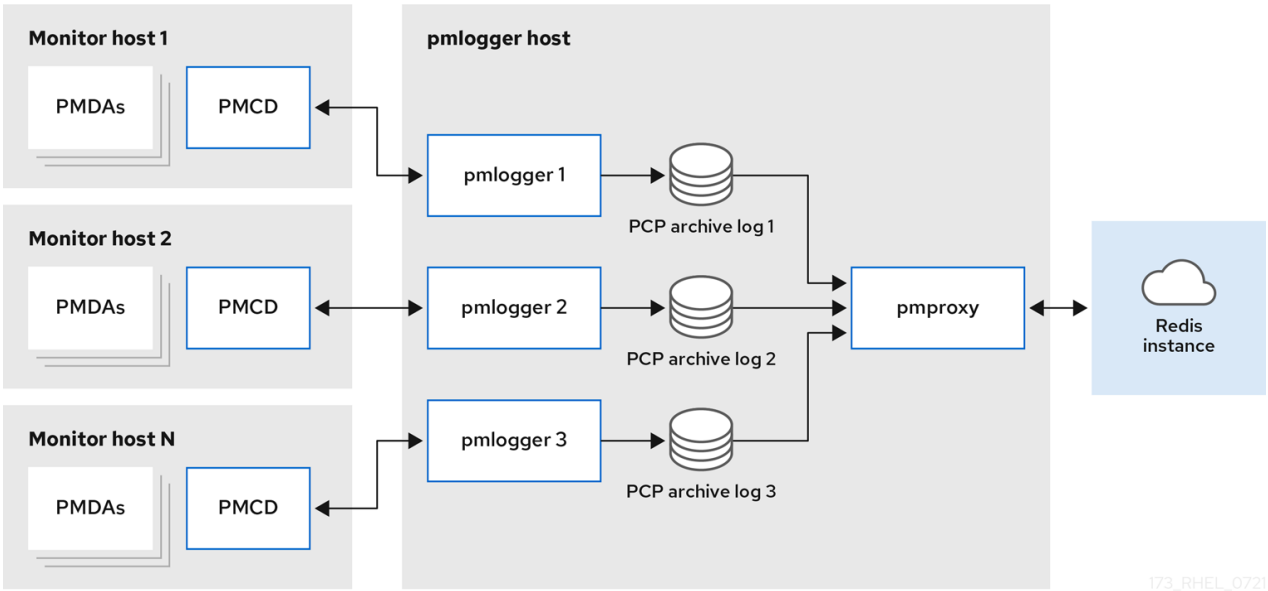
그림 6.1. 분산된 로깅



중앙 집중식 로깅 - pmlogger 팜

모니터링된 호스트의 리소스 사용량이 제한되면 또 다른 배포 옵션은 **pmlogger** 팜으로, 중앙 집중식 로깅이라고도 합니다. 이 설정에서 단일 로거 호스트는 여러 **pmlogger** 프로세스를 실행하고 각각 다른 원격 **pmcd** 호스트에서 성능 지표를 검색하도록 구성됩니다. 또한 중앙 집중식 로거 호스트는 **pmproxy** 서비스를 실행하도록 구성되어 있으며 생성된 **PCP**에서 로그를 검색하고 지표 데이터를 **Redis** 인스턴스로 로드합니다.

그림 6.2. 중앙 집중식 로깅 - pmlogger 팜

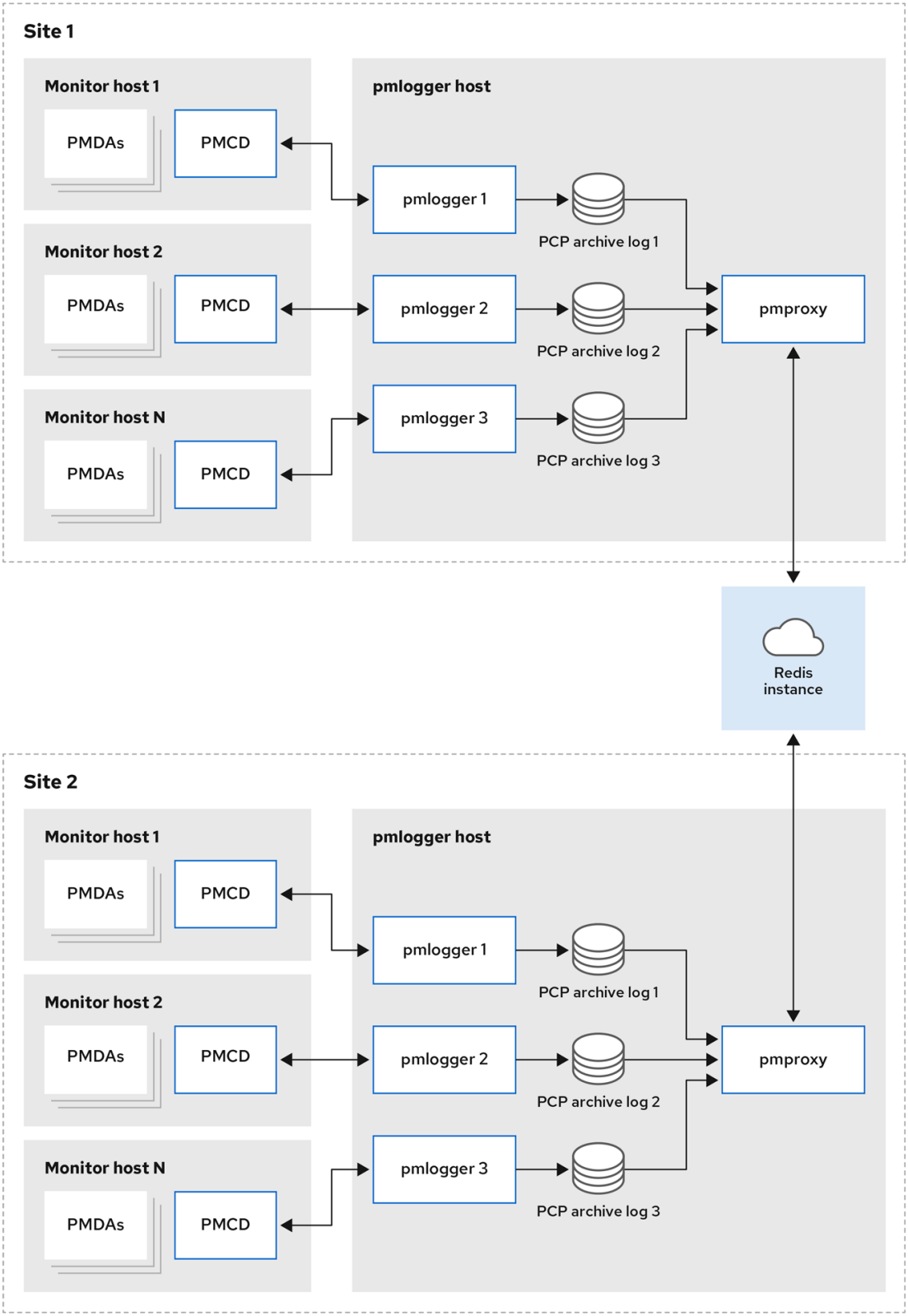


페더레이션 - 여러 pmlogger 팜

대규모 배포의 경우 **Red Hat**은 여러 **pmlogger** 팜을 통합 방식으로 배포할 것을 권장합니다. 예를 들어 랙 또는 데이터 센터당 한 개의 **pmlogger** 팜이 있습니다. 각 **pmlogger** 팜은 지표를 중앙 **Redis**

인스턴스로 로드합니다.

그림 6.3. 페더레이션 - 여러 pmlogger 팜



참고

기본적으로 **Redis**의 배포 설정은 독립 실행형 **localhost**입니다. 그러나 **Redis**는 여러 호스트에서 데이터를 공유하는 고가용성 및 확장성이 뛰어난 클러스터형 방식으로 선택적으로 수행할 수 있습니다. 또 다른 실행 가능한 옵션은 클라우드에 **Redis** 클러스터를 배포하거나 클라우드 벤더의 관리형 **Redis** 클러스터를 활용하는 것입니다.

추가 리소스

- **pp(1), pmlogger(1), pmproxy(1)** 및 **pmcd(1)** 도움말 페이지
- 권장되는 배포 아키텍처

6.7. 권장되는 배포 아키텍처

다음 테이블에서는 모니터링된 호스트 수에 따라 권장되는 배포 아키텍처를 설명합니다.

표 6.3. 권장되는 배포 아키텍처

호스트 수 (N)	1-10	10-100	100-1000
pmcd 서버	N	N	N
pmlogger 서버	1에서 N	N/10 - N	N/100~N
pmproxy 서버	1에서 N	1에서 N	N/100~N
Redis 서버	1에서 N	1에서 N/10	N/100에서 N/10으로
Redis 클러스터	없음	아마도	있음
권장되는 배포 설정	localhost, Decentralized 또는 Centralized 로깅	분산, 중앙 집중식 로깅 또는 페더레이션	분산 또는 페더레이션

6.8. 크기 조정 요소

다음은 스케일링에 필요한 크기 조정 요소입니다.

원격 시스템 크기

CPU, 디스크, 네트워크 인터페이스 및 기타 하드웨어 리소스의 수는 중앙 집중식 로깅 호스트의

각 pmlogger 에서 수집한 데이터 양에 영향을 미칩니다.

기록된 메트릭

기록된 지표의 수와 유형은 중요한 역할을 합니다. 특히 프로세스별 proc.* 메트릭에는 많은 양의 디스크 공간이 필요합니다. 예를 들어 표준 pcp-zeroconf 설정, 10s 로깅 간격, 11MB, proc 지표가 없는 경우 155MB 이상 - 10배 더 많은 디스크 공간 필요. 또한 각 지표의 인스턴스 수(예: CPU 수, 블록 장치 및 네트워크 인터페이스)도 필요한 스토리지 용량에 영향을 미칩니다.

로깅 간격

지표가 기록되는 간격은 스토리지 요구 사항에 영향을 미칩니다. 매일 예상되는 PCP 아카이브 파일 크기는 각 pmlogger 인스턴스의 pmlogger.log 파일에 기록됩니다. 이러한 값은 압축되지 않은 예상입니다. PCP 아카이브는 매우 잘 압축되기 때문에 특정 사이트에 대해 실제 장기 디스크 공간 요구 사항을 결정할 수 있습니다.

pmlogrewrite

모든 PCP 업그레이드 후 pmlogrewrite 툴이 실행되고 이전 버전의 PCP의 지표 메타데이터가 변경된 경우 이전 아카이브를 다시 작성합니다. 이 프로세스 기간은 저장된 아카이브 수에 따라 선형으로 확장됩니다.

추가 리소스

- pmlogrewrite(1) 및 pmlogger(1) 도움말 페이지

6.9. PCP 스케일링을 위한 구성 옵션

다음은 확장에 필요한 구성 옵션입니다.

sysctl 및 rlimit 설정

아카이브 검색이 활성화되면 pmproxy 에는 서비스 로그 및 pm proxy 클라이언트 소켓에 대한 추가 파일 설명자와 함께 모든 pm logger 에 대해 4개의 설명자가 필요합니다. 각 pmlogger 프로세스는 원격 pmcd 소켓에 약 20개의 파일 설명자, 아카이브 파일, 서비스 로그 등을 사용합니다. 합계는 약 200 pmlogger 프로세스를 실행하는 시스템의 기본 1024 소프트 제한을 초과할 수 있습니다. pcp-5.3.0 이상의 pmproxy 서비스는 소프트 제한을 하드 제한으로 자동으로 늘립니다. 이전 버전의 PCP에서는 많은 수의 pmlogger 프로세스를 배포해야 하며 pmlogger 의 소프트 또는 하드 제한을 늘리면 이 작업을 수행할 수 있습니다. 자세한 내용은 systemd에서 실행하는 서비스에 대한 제한 설정(ulimit)을 참조하십시오.

로컬 아카이브

pmlogger 서비스는 로컬 및 원격 pmcds 의 지표를 /var/log/pcp/pmlogger/ 디렉터리에 저장합니다. 로컬 시스템의 로깅 간격을 제어하려면 /etc/pcp/pmlogger/control.d/*configfile* 파일을 업데이트하고 인수에 -t *X* 를 추가합니다. 여기서 *X* 는 초 단위 로깅 간격입니다. 로깅해야 하는 지표를 구성하려면 pmlogconf /var/lib/pcp/config/pmlogger/config.*클라이언트hostname* 을 실행합니다. 이 명

령은 선택적으로 추가 사용자 지정할 수 있는 기본 지표 세트와 함께 구성 파일을 배포합니다. 보존 설정을 지정하려면 이전 **PCP** 아카이브를 삭제할 때 **/etc/sysconfig/pmlogger_timers** 파일을 업데이트하고 **PMLOGGER_DAILY_PARAMS="-E *X*"** 를 지정합니다. 여기서 ***X*** 는 **PCP** 아카이브를 유지할 일수입니다.

redis

pmproxy 서비스는 기록된 지표를 **pmlogger** 에서 **Redis** 인스턴스로 보냅니다. 다음은 **/etc/pcp/pmproxy/pmproxy.conf** 구성 파일에서 보존 설정을 지정하는 데 사용할 수 있는 두 가지 옵션입니다.

- **stream.expire** 는 오래된 지표를 제거해야 하는 경우의 기간을 지정합니다. 이 값은 지정된 시간(초) 내에 업데이트되지 않은 지표입니다.
- **stream.maxlen** 은 호스트당 하나의 지표에 대한 최대 지표 값 수를 지정합니다. 이 설정은 로깅 간격으로 분할된 보존 시간이어야 합니다(예: **14**일 보존 및 **60**일 로깅 간격 **(60*60*24*14/60)**

추가 리소스

- **pmproxy(1), pmlogger(1)** 및 **sysctl(8)** 도움말 페이지

6.10. 예제: 중앙 집중식 로깅 배포 분석

다음 결과는 중앙 집중식 로깅 설정으로 수집되었으며, 기본 **pcp-zeroconf 5.3.0** 설치가 있으며, 각 원격 호스트는 **64 CPU** 코어가 **64**개, **376GB RAM** 및 연결된 서버에서 **pmcd** 를 실행하는 것과 동일한 컨테이너 인스턴스입니다.

로깅 간격은 **10**초이고 원격 노드의 **proc** 지표는 포함되지 않으며, 메모리 값은**RSS(Resident Set Size)** 값을 참조합니다.

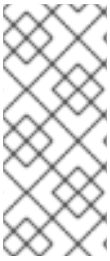
표 6.4. 10초 로깅 간격에 대한 자세한 사용 통계

호스트 수	10	50
PCP 아카이브 스토리지(1일당 스토리지)	91 MB	522MB
pmlogger 메모리	160MB	580MB
Day당 **pmlogger** 네트워크(In)	2MB	9MB

호스트 수	10	50
pmproxy 메모리	1.4GB	6.3GB
일당 Redis 메모리	2.6GB	12GB

표 6.5. 60초 로깅 간격을 위해 모니터링된 호스트에 따라 사용된 리소스

호스트 수	10	50	100
PCP 아카이브 스토리지 (1일당 스토리지)	20MB	120MB	271MB
pmlogger 메모리	104MB	524MB	1049MB
Day당 **pmlogger** 네트워크(In)	0.38 MB	1.75MB	3.48MB
pmproxy 메모리	2.67GB	5.5GB	9GB
일당 Redis 메모리	0.54GB	2.65GB	5.3GB



참고

pmproxy 큐 **Redis**는 **Redis** 쿼리 속도를 높이기 위해 **Redis** 파이프라인을 요청하고 사용합니다. 이로 인해 메모리 사용량이 증가할 수 있습니다. 이 문제를 해결하려면 메모리가 많은 사용량 문제 해결을 참조하십시오.

6.11. 예제: 페더레이션 설정 배포 분석

다음은 3개의 중앙 로깅(**pmlogger** 팜) 설정으로 구성된 페더레이션 팜으로 알려진 페더레이션 설정에서 관찰되었으며, 각 **pmlogger** 팜은 100대의 원격 호스트를 모니터링했으며, 이 설정은 총 300개의 호스트입니다.

pmlogger 팜 설정은 예:에서 언급한 구성과 동일합니다. **Redis** 서버가 클러스터 모드에서 작동 중임을 제외하고 중앙 집중식 로깅 배포 분석 **60s** 로깅 간격.

표 6.6. 60초 로깅 간격 동안 페더레이션 호스트에 따라 사용된 리소스

PCP 아카이브 스토리지(1일당 스토리지)	pmlogger 메모리	날짜당 네트워크 (In/Out)	pmproxy 메모리	일당 Redis 메모리
277MB	1058MB	15.6 MB / 12.3 MB	6-8GB	5.5GB

여기에서 모든 값은 호스트당입니다. 네트워크 대역폭은 **Redis** 클러스터의 노드 간 통신 때문에 더 높습니다.

6.12. 메모리 사용량이 많은 문제 해결

다음 시나리오는 메모리 사용량이 높을 수 있습니다.

- **pmproxy** 프로세스는 새로운 **PCP** 아카이브를 처리하는데 사용되며, **Redis** 요청 및 응답을 처리하는 예비 **CPU** 사이클이 없습니다.
- **Redis** 노드 또는 클러스터는 과부하되어 입력되는 요청을 제때 처리할 수 없습니다.

pmproxy 서비스 데몬은 **Redis** 스트림을 사용하며 구성 매개변수인 **PCP** 튜닝 매개 변수이며 **Redis** 메모리 사용량 및 키 보존에 영향을 미칩니다. **/etc/pcp/pmproxy/pmproxy.conf** 파일에는 **pmproxy** 및 관련 **API**에 사용 가능한 구성 옵션이 나열됩니다.

이 섹션에서는 메모리 사용량이 높은 문제를 해결하는 방법을 설명합니다.

사전 요구 사항

1. **pcp-pmda-redis** 패키지를 설치합니다.

```
# yum install pcp-pmda-redis
```

2. **redis PMDA**를 설치합니다.

```
# cd /var/lib/pcp/pmdas/redis && ./Install
```

절차

- 메모리 사용량이 높은 문제를 해결하려면 다음 명령을 실행하고 진행 열을 관찰합니다.

```
$ pmrep :pmproxy
       backlog  inflight  reqs/s  resp/s   wait req err  resp err  changed  throttled
          byte     count   count/s  count/s  s/s  count/s   count/s  count/s   count/s
14:59:08   0        0       N/A      N/A    N/A    N/A       N/A      N/A       N/A
14:59:09   0        0    2268.9   2268.9    28      0         0      2.0       4.0
14:59:10   0        0       0.0      0.0     0      0         0      0.0       0.0
14:59:11   0        0       0.0      0.0     0      0         0      0.0       0.0
```

이 열에는 진행 중 요청 수가 표시됩니다. 즉, 대기 또는 전송되었음을 의미하며 지금까지 응답이 수신되지 않았습니다.

높은 숫자는 다음 조건 중 하나를 나타냅니다.

  - **pmproxy** 프로세스는 새로운 **PCP** 아카이브를 처리하는데 사용되며, **Redis** 요청 및 응답을 처리하는 예비 **CPU** 사이클이 없습니다.

  - **Redis** 노드 또는 클러스터는 과부하되어 입력되는 요청을 제때 처리할 수 없습니다.

- 메모리 사용량이 많은 문제를 해결하려면 이 팜의 **pmlogger** 프로세스 수를 줄이고 다른 **pmlogger** 팜을 추가합니다. 페더레이션 **-** 여러 **pmlogger** 팜 설정을 사용합니다.

Redis 노드에서 오랜 시간 동안 **100% CPU**를 사용하고 있는 경우 성능이 더 뛰어난 호스트로 이동하거나 클러스터형 **Redis setup**을 대신 사용하십시오.

- **pmproxy.redis.*** 지표를 보려면 다음 명령을 사용합니다.

```
$ pminfo -ftd pmproxy.redis
pmproxy.redis.responses.wait [wait time for responses]
    Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: counter  Units: microsec
    value 546028367374
pmproxy.redis.responses.error [number of error responses]
    Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: counter  Units: count
    value 1164
[...]
pmproxy.redis.requests.inflight.bytes [bytes allocated for inflight requests]
    Data Type: 64-bit int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: discrete  Units: byte
```

```
    value 0

pmproxy.redis.requests.inflight.total [inflight requests]
    Data Type: 64-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: discrete  Units: count
    value 0
[...]
```

진행 중인 **Redis** 요청 수를 보려면 **pmproxy.redis.requests.inflight.total** 메트릭 및 **pmproxy.redis.requests.inflight.bytes** 메트릭을 참조하여 현재 **inflight Redis** 요청에 의해 사용되는 바이트 수를 확인합니다.

일반적으로 **redis** 요청 대기열은 **0**이지만 대규모 **pmlogger** 팜을 사용하여 빌드할 수 있으며 이는 확장성을 제한하고 **pmproxy** 클라이언트에 대한 높은 대기 시간을 초래할 수 있습니다.

- **pminfo** 명령을 사용하여 성능 지표에 대한 정보를 봅니다. 예를 들어 **redis.*** 메트릭을 보려면 다음 명령을 사용합니다.

```
$ pminfo -ftd redis
redis.redis_build_id [Build ID]
    Data Type: string  InDom: 24.0 0x6000000
    Semantics: discrete  Units: count
    inst [0 or "localhost:6379"] value "87e335e57cffa755"
redis.total_commands_processed [Total number of commands processed by the server]
    Data Type: 64-bit unsigned int  InDom: 24.0 0x6000000
    Semantics: counter  Units: count
    inst [0 or "localhost:6379"] value 595627069
[...]

redis.used_memory_peak [Peak memory consumed by Redis (in bytes)]
    Data Type: 32-bit unsigned int  InDom: 24.0 0x6000000
    Semantics: instant  Units: count
    inst [0 or "localhost:6379"] value 572234920
[...]
```

최대 메모리 사용량을 보려면 **redis.used_memory_peak** 지표를 참조하십시오.

추가 리소스

- **pmdaredis(1), pmproxy(1)** 및 **pminfo(1)** 도움말 페이지
- PCP 배포 아키텍처

7장. PMLOGGER를 사용하여 성능 데이터 로깅

PCP 도구를 사용하면 성능 지표 값을 로깅하고 나중에 재생할 수 있습니다. 이를 통해 소급 성능 분석을 수행할 수 있습니다.

pmlogger 도구를 사용하여 다음을 수행할 수 있습니다.

- 시스템에서 선택한 메트릭의 아카이브된 로그를 생성합니다.
- 시스템에 기록되는 메트릭과 빈도를 지정합니다.

7.1. PMLOGCONF를 사용하여 PMLOGGER 구성 파일 수정

pmlogger 서비스가 실행 중이면 PCP는 호스트에 기본 지표 집합을 기록합니다.

pmlogconf 유틸리티를 사용하여 기본 구성을 확인합니다. pmlogger 구성 파일이 없는 경우 pmlogconf 는 기본 지표 값으로 생성합니다.

사전 요구 사항

- PCP가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.

절차

1. pmlogger 구성 파일을 생성하거나 수정합니다.

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
```

2. pmlogconf 프롬프트에 따라 관련 성능 지표 그룹을 활성화 또는 비활성화하고 활성화된 각 그룹의 로깅 간격을 제어합니다.

추가 리소스

- pmlogconf(1) 및 pmlogger(1) 도움말 페이지

- PCP로 배포되는 툴
- PCP와 함께 배포되는 시스템 서비스

7.2. PMLOGGER 구성 파일 수동 편집

특정 지표와 지정된 간격으로 맞춤형 로깅 구성을 생성하려면 **pmlogger** 구성 파일을 수동으로 편집합니다. 기본 **pmlogger** 구성 파일은 **/var/lib/pcp/config/pmlogger/config.default** 입니다. 구성 파일은 기본 로깅 인스턴스에서 로깅하는 지표를 지정합니다.

수동 구성에서는 다음을 수행할 수 있습니다.

- 자동 구성에 나열되지 않은 레코드 지표입니다.
- 사용자 정의 로깅을 선택합니다.
- 애플리케이션 지표로 **PMDA** 를 추가합니다.

사전 요구 사항

- **PCP**가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.

절차

- **/var/lib/pcp/config/pmlogger/config.default** 파일을 열고 편집하여 특정 지표를 추가합니다.

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

log mandatory on every 5 seconds {
    xfs.write
    xfs.write_bytes
    xfs.read
    xfs.read_bytes
}
```

```
log mandatory on every 10 seconds {
    xfs.allocs
    xfs.block_map
    xfs.transactions
    xfs.log

}

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

추가 리소스

- **pmlogger(1)** 도움말 페이지
- PCP로 배포되는 툴
- PCP와 함께 배포되는 시스템 서비스

7.3. PMLOGGER 서비스 활성화

pmlogger 서비스를 시작하고 활성화하여 로컬 시스템의 지표 값을 기록해야 합니다.

이 절차에서는 **pmlogger** 서비스를 활성화하는 방법을 설명합니다.

사전 요구 사항

- **PCP**가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.

절차

- **pmlogger** 서비스를 시작하고 활성화합니다.

```
# systemctl start pmlogger

# systemctl enable pmlogger
```

검증 단계

- **pmlogger** 서비스가 활성화되었는지 확인합니다.

```
# pcp

Performance Co-Pilot configuration on workstation:

platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents, 1 client
pmda: root pmcd proc xfs linux mmv kvm jbd2
pmlogger: primary logger: /var/log/pcp/pmlogger/workstation/20190827.15.54
```

추가 리소스

- **pmlogger(1)** 도움말 페이지
- PCP로 배포되는 툴
- PCP와 함께 배포되는 시스템 서비스
- **/var/lib/pcp/config/pmlogger/config.default file**

7.4. 메트릭 컬렉션을 위한 클라이언트 시스템 설정

다음 절차에서는 중앙 서버가 **PCP**를 실행하는 클라이언트에서 지표를 수집할 수 있도록 클라이언트 시스템을 설정하는 방법을 설명합니다.

사전 요구 사항

- **PCP**가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.

절차

1. **pcp-system-tools** 패키지를 설치합니다.

```
# yum install pcp-system-tools
```

2. **pmcd 의 IP 주소를 구성합니다.**

```
# echo "-i 192.168.4.62" >>/etc/pcp/pmcd/pmcd.options
```

192.168.4.62 **를 IP 주소로 교체하고 클라이언트가 수신 대기해야 합니다.**

기본적으로 pmcd 는 localhost에서 수신 대기 중입니다.

3. **퍼블릭 영역을 영구적으로 추가하도록 방화벽을 구성합니다.**

```
# firewall-cmd --permanent --zone=public --add-port=44321/tcp
success

# firewall-cmd --reload
success
```

4. **SELinux 부울을 설정합니다.**

```
# setsebool -P pcp_bind_all_unreserved_ports on
```

5. **pmcd 및 pmlogger 서비스를 활성화합니다.**

```
# systemctl enable pmcd pmlogger
# systemctl restart pmcd pmlogger
```

검증 단계

- **pmcd 가 구성된 IP 주소에서 올바르게 수신 대기하는지 확인합니다.**

```
# ss -tlp | grep 44321
LISTEN   0   5     127.0.0.1:44321   0.0.0.0:*   users:(("pmcd",pid=151595,fd=6))
LISTEN   0   5  192.168.4.62:44321   0.0.0.0:*   users:(("pmcd",pid=151595,fd=0))
LISTEN   0   5        [::1]:44321      [::]:*   users:(("pmcd",pid=151595,fd=7))
```

추가 리소스

- pmlogger(1), firewall-cmd(1), ss(8) 및 setsebool(8) 도움말 페이지
- PCP로 배포되는 툴
- PCP와 함께 배포되는 시스템 서비스
- /var/lib/pcp/config/pmlogger/config.default file

7.5. 데이터 수집을 위해 중앙 서버 설정

다음 절차에서는 PCP를 실행하는 클라이언트에서 지표를 수집하는 중앙 서버를 생성하는 방법을 설명합니다.

사전 요구 사항

- PCP가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.
- 클라이언트는 지표 수집에 대해 구성됩니다. 자세한 내용은 메트릭 수집을 위한 클라이언트 시스템 설정을 참조하십시오.

절차

1. pcp-system-tools 패키지를 설치합니다.

```
# yum install pcp-system-tools
```

2. 다음 콘텐츠를 사용하여 /etc/pcp/pmlogger/control.d/remote 파일을 만듭니다.

```
# DO NOT REMOVE OR EDIT THE FOLLOWING LINE
$version=1.1

192.168.4.13 n n PCP_ARCHIVE_DIR/rhel7u4a -r -T24h10m -c config.rhel7u4a
192.168.4.14 n n PCP_ARCHIVE_DIR/rhel6u10a -r -T24h10m -c config.rhel6u10a
192.168.4.62 n n PCP_ARCHIVE_DIR/rhel8u1a -r -T24h10m -c config.rhel8u1a
```

192.168.4.13,*192.168.4.14* 및 *192.168.4.62* 를 클라이언트 IP 주소로 바꿉니다.

참고

Red Hat Enterpirse Linux 8.0에서 8.1 및 8.2에서는 제어 파일에서 원격 호스트에 대해 다음 형식을 사용합니다. PCP_LOG_DIR/pmlogger/*host_name*.

3. pmcd 및 pmlogger 서비스를 활성화합니다.

```
# systemctl enable pmcd pmlogger
# systemctl restart pmcd pmlogger
```

검증 단계

- 각 디렉터리에서 최신 아카이브 파일에 액세스할 수 있는지 확인합니다.

```
# for i in /var/log/pcp/pmlogger/rhel*/*.0; do pmdumplog -L $i; done
Log Label (Log Format Version 2)
Performance metrics from host rhel6u10a.local
  commencing Mon Nov 25 21:55:04.851 2019
  ending     Mon Nov 25 22:06:04.874 2019
Archive timezone: JST-9
PID for pmlogger: 24002
Log Label (Log Format Version 2)
Performance metrics from host rhel7u4a
  commencing Tue Nov 26 06:49:24.954 2019
  ending     Tue Nov 26 07:06:24.979 2019
Archive timezone: CET-1
PID for pmlogger: 10941
[..]
```

/var/log/pcp/pmlogger/ 디렉토리의 아카이브 파일을 추가 분석 및 그래프 작성에 사용할 수 있습니다.

추가 리소스

- pmlogger(1) 도움말 페이지
- PCP로 배포되는 툴

- PCP와 함께 배포되는 시스템 서비스
- /var/lib/pcp/config/pmlogger/config.default file

7.6. PMREP를 사용하여 PCP 로그 아카이브 재생

지표 데이터를 기록한 후 PCP 로그 아카이브를 재생할 수 있습니다. 로그를 텍스트 파일로 내보내고 스프레드시트로 가져오려면 pcp2csv,pcp2 xml,pmrep 또는 pm logsummary 와 같은 PCP 유틸리티를 사용합니다.

pmrep 툴을 사용하면 다음을 수행할 수 있습니다.

- 로그 파일 보기
- 선택한 PCP 로그 아카이브를 구문 분석하고 값을 ASCII 테이블로 내보냅니다.
- 명령줄에 개별 지표를 지정하여 전체 아카이브 로그를 추출하거나 로그에서 지표 값만 선택합니다.

사전 요구 사항

- PCP가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.
- pmlogger 서비스가 활성화되었습니다. 자세한 내용은 pmlogger 서비스 활성화를 참조하십시오.
- pcp-system-tools 패키지를 설치합니다.

```
# yum install pcp-gui
```

절차

- 메트릭의 데이터를 표시합니다.

```
$ pmrep --start @3:00am --archive 20211128 --interval 5seconds --samples 10 --output csv
disk.dev.write
Time,"disk.dev.write-sda","disk.dev.write-sdb"
2021-11-28 03:00:00,,
2021-11-28 03:00:05,4.000,5.200
2021-11-28 03:00:10,1.600,7.600
2021-11-28 03:00:15,0.800,7.100
2021-11-28 03:00:20,16.600,8.400
2021-11-28 03:00:25,21.400,7.200
2021-11-28 03:00:30,21.200,6.800
2021-11-28 03:00:35,21.000,27.600
2021-11-28 03:00:40,12.400,33.800
2021-11-28 03:00:45,9.800,20.600
```

언급된 예제에서는 아카이브에 수집된 **disk.dev.write** 지표의 데이터를 쉼표로 구분된 값 형식으로 ***5*초 간격으로 표시합니다.**

참고

이 예제에서 **20211128** 을 데이터를 표시하려는 **pmlogger** 아카이브가 포함된 파일 이름으로 바꿉니다.

추가 리소스

- **pmlogger(1), pmrep(1)** 및 **pmlogsummary(1)** 도움말 페이지
- **PCP**로 배포되는 툴
- **PCP**와 함께 배포되는 시스템 서비스

8장. PERFORMANCE CO-PILOT을 통한 성능 모니터링

PCP(Performance Co-Pilot)는 시스템 수준의 성능 측정을 모니터링, 시각화, 저장 및 분석하기 위한 툴, 서비스 및 라이브러리 제품군입니다.

시스템 관리자는 Red Hat Enterprise Linux 8의 PCP 애플리케이션을 사용하여 시스템 성능을 모니터링할 수 있습니다.

8.1. PMDA-POSTFIX를 사용하여 POSTFIX 모니터링

이 절차에서는 pmda-postfix 를 사용하여 postfix 메일 서버의 성능 지표를 모니터링하는 방법을 설명합니다. 초당 받은 이메일 수를 확인하는 데 도움이 됩니다.

사전 요구 사항

- PCP가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.
- pmlogger 서비스가 활성화되었습니다. 자세한 내용은 pmlogger 서비스 활성화를 참조하십시오.

절차

1. 다음 패키지를 설치합니다.
   a. pcp-system-tools 를 설치합니다.

```
# yum install pcp-system-tools
```

   b. pmda-postfix 패키지를 설치하여 postfix 를 모니터링합니다.

```
# yum install pcp-pmda-postfix postfix
```

   c. 로깅 데몬을 설치합니다.

```
# yum install rsyslog
```

d. **테스트를 위해 메일 클라이언트를 설치합니다.**

```
# yum install mutt
```

2. **postfix 및 rsyslog 서비스를 활성화합니다.**

```
# systemctl enable postfix rsyslog
# systemctl restart postfix rsyslog
```

3. **pmda-postfix 가 필요한 로그 파일에 액세스할 수 있도록 SELinux 부울을 활성화합니다.**

```
# setsebool -P pcp_read_generic_logs=on
```

4. **PMDA 를 설치합니다.**

```
# cd /var/lib/pcp/pmdas/postfix/

# ./Install

Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Waiting for pmcd to terminate ...
Starting pmcd ...
Check postfix metrics have appeared ... 7 metrics and 58 values
```

검증 단계

- **pmda-postfix 작업을 확인합니다.**

```
echo testmail | mutt root
```

- **사용 가능한 지표를 확인합니다.**

```
# pminfo postfix

postfix.received
postfix.sent
postfix.queues.incoming
postfix.queues.maildrop
```

```
postfix.queues.hold
postfix.queues.deferred
postfix.queues.active
```

추가 리소스

- **rsyslogd(8), postfix(1)** 및 **setsebool(8)** 도움말 페이지
- **PCP**로 배포되는 툴
- **PCP**와 함께 배포되는 시스템 서비스
- **/var/lib/pcp/config/pmlogger/config.default file**

8.2. PCP CHARTS 애플리케이션을 사용하여 PCP 로그 아카이브를 시각적으로 추적

지표 데이터를 기록하면 **PCP** 로그 아카이브를 그래프로 재생할 수 있습니다. 지표는 **PCP** 로그 아카이브의 지표 데이터를 기록 데이터 소스로 사용하기 위해 대체 옵션을 사용하여 하나 이상의 라이브 호스트에서 가져옵니다. 성능 지표의 데이터를 표시하도록 **PCP** 차트 애플리케이션 인터페이스를 사용자 지정하려면 선 플롯, 막대 그래프 또는 사용률 그래프를 사용할 수 있습니다.

PCP Charts 애플리케이션을 사용하면 다음을 수행할 수 있습니다.

- **PCP Charts** 애플리케이션 애플리케이션에서 데이터를 재생성하고 그래프를 사용하여 시스템의 실시간 데이터와 함께 소급 데이터를 시각화합니다.
- 성능 지표 값을 그래프에 표시합니다.
- 여러 차트를 동시에 표시합니다.

사전 요구 사항

- **PCP**가 설치되어 있습니다. 자세한 내용은 **PCP** 설치 및 활성화를 참조하십시오.
-

pmlogger를 사용하여 성능 데이터를 기록했습니다. 자세한 내용은 pmlogger로 성능 데이터 로깅 을 참조하십시오.

- pcp-gui 패키지를 설치합니다.

```
# yum install pcp-gui
```

절차

1. 명령줄에서 PCP Charts 애플리케이션을 시작합니다.

```
# pmchart
```

그림 8.1. PCP Charts 애플리케이션

pmtime 서버 설정은 맨 아래에 있습니다. 시작 및 일시 중지 버튼을 사용하면 다음을 제어할 수 있습니다.

- PCP가 메트릭 데이터를 폴링하는 간격
- 기록 데이터의 지표에 대한 날짜 및 시간

2. File (파일)을 클릭한 다음 New Chart (새 차트)를 클릭하여 호스트 이름 또는 주소를 지정하여 로컬 시스템과 원격 시스템 모두에서 지표를 선택합니다. 고급 구성 옵션에는 차트 값을 수동

으로 설정하고 플롯의 컬러를 수동으로 선택할 수 있는 기능이 포함됩니다.

3. **PCP Charts** 애플리케이션에서 생성된 뷰를 기록합니다.

   다음은 **PCP Charts** 애플리케이션에 생성된 뷰를 기록하거나 이미지를 가져오는 옵션입니다.

   - **File**(파일)을 클릭한 다음 **Export** (내보내기)를 클릭하여 현재 보기의 이미지를 저장합니다.
   - **Record**(레코드)를 클릭한 다음 **Start** (시작)를 클릭하여 레코드를 시작합니다. **Record**(레코드)를 클릭한 다음 **Stop** (중지)을 클릭하여 레코드를 중지합니다. 녹화를 중지하고 나면 기록된 지표가 나중에 볼 수 있도록 보관됩니다.

4. 선택 사항: **PCP Charts** (PCP 차트) 애플리케이션에서 뷰로 알려진 기본 구성 파일은 하나 이상의 차트와 연결된 메타데이터를 저장할 수 있습니다. 이 메타데이터는 사용된 메트릭과 차트 열을 포함하여 모든 차트 측면을 설명합니다. **File**(파일)을 클릭한 다음 **Save View** (보기 저장)를 클릭하고 나중에 뷰 구성을 로드하여 사용자 지정 보기 구성을 저장합니다.

   PCP Charts 애플리케이션 보기 구성 파일의 다음 예제에서는 지정된 **XFS** 파일 시스템 **loop1** 에 읽고 쓴 총 바이트 수를 보여주는 스택 차트 그래프를 설명합니다.

```
#kmchart
version 1

chart title "Filesystem Throughput /loop1" style stacking antialiasing off
   plot legend "Read rate"   metric xfs.read_bytes   instance  "loop1"
   plot legend "Write rate"  metric xfs.write_bytes  instance  "loop1"
```

추가 리소스

- **pmchart(1)** 및 **pmtime(1)** 도움말 페이지
- PCP로 배포되는 툴

8.3. PCP를 사용하여 SQL 서버에서 데이터 수집

Red Hat Enterprise Linux 8.2 이상을 사용하면 데이터베이스 성능 문제를 모니터링하고 분석하는 데 도움이 되는 **PCP(Performance Co-Pilot)**에서 **SQL Server** 에이전트를 사용할 수 있습니다.

다음 절차에서는 시스템의 **pcp** 를 통해 **Microsoft SQL Server**의 데이터를 수집하는 방법을 설명합니다.

사전 요구 사항

- **Microsoft SQL Server for Red Hat Enterprise Linux**를 설치하고 **SQL** 서버에 '신뢰할 수 있는' 연결을 구축했습니다.
- **Red Hat Enterprise Linux**용 **SQL Server**용 **Microsoft ODBC** 드라이버를 설치했습니다.

절차

1. **PCP**를 설치합니다.

```
# yum install pcp-zeroconf
```

2. **pyodbc** 드라이버에 필요한 패키지를 설치합니다.

```
# yum install gcc-c++ python3-devel unixODBC-devel

# yum install python3-pyodbc
```

3. **mssql** 에이전트를 설치합니다.

   a. **PCP**용 **Microsoft SQL Server** 도메인 에이전트를 설치합니다.

   ```
   # yum install pcp-pmda-mssql
   ```

   b. **/etc/pcp/mssql/mssql.conf** 파일을 편집하여 **mssql** 에이전트에 대한 **SQL** 서버 계정 사용자 이름과 암호를 구성합니다. 구성한 계정에 성능 데이터에 대한 액세스 권한이 있는지 확인합니다.

   ```
   username: user_name
   password: user_password
   ```

   user_name 을 **SQL Server** 계정으로 바꾸고 ***user_password*** 를 이 계정의 **SQL Server** 사용자 암호로 바꿉니다.

4. 에이전트를 설치합니다.

```
# cd /var/lib/pcp/pmdas/mssql
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check mssql metrics have appeared ... 168 metrics and 598 values
[...]
```

검증 단계

- **pcp** 명령을 사용하여**mssql(SQL** 서버 **PMDA)**이 로드되고 실행 중인지 확인합니다.

```
$ pcp
Performance Co-Pilot configuration on rhel.local:

platform: Linux rhel.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC 2019
x86_64
 hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
 timezone: PDT+7
 services: pmcd pmproxy
     pmcd: Version 5.0.2-1, 12 agents, 4 clients
     pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
           jbd2 dm
 pmlogger: primary logger: /var/log/pcp/pmlogger/rhel.local/20200326.16.31
     pmie: primary engine: /var/log/pcp/pmie/rhel.local/pmie.log
```

- **PCP**가 **SQL Server**에서 수집할 수 있는 전체 메트릭 목록을 확인합니다.

```
# pminfo mssql
```

- 지표 목록을 확인한 후 트랜잭션 비율을 보고할 수 있습니다. 예를 들어 **5**초 동안 초당 전체 트랜잭션 수를 보고하려면 다음을 수행합니다.

```
# pmval -t 1 -T 5 mssql.databases.transactions
```

- **pmchart** 명령을 사용하여 시스템에서 이러한 지표의 그래픽 차트를 봅니다. 자세한 내용은 **PCP Charts** 애플리케이션을 사용하여 시각적으로 **PCP** 로그 아카이브 추적을 참조하십시오.

추가 리소스

- pcp(1), pminfo(1), pmval(1), pmchart(1), and pmdamssql(1) man pages

- Microsoft SQL Server용 Performance Co-Pilot 및 RHEL 8.2 Red Hat Developers 블로그 게시물

9장. PCP를 사용한 XFS 성능 분석

XFS PMDA는 pcp 패키지의 일부로 제공되며 설치 중에 기본적으로 활성화됩니다. PCP(Performance Co-Pilot)에서 XFS 파일 시스템의 성능 지표 데이터를 수집하는 데 사용됩니다.

이 섹션에서는 PCP를 사용하여 XFS 파일 시스템의 성능을 분석하는 방법을 설명합니다.

9.1. 수동으로 XFS PMDA 설치

XFS PMDA가 pcp 구성 출력에 나열되지 않은 경우 수동으로 PMDA 에이전트를 설치합니다.

이 절차에서는 PMDA 에이전트를 수동으로 설치하는 방법을 설명합니다.

사전 요구 사항

- PCP가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.

절차

1. xfs 디렉터리로 이동합니다.

```
# cd /var/lib/pcp/pmdas/xfs/
```

검증 단계

- pmcd 프로세스가 호스트에서 실행 중이며 XFS PMDA가 구성에서 enabled로 나열되는지 확인합니다.

```
# pcp

Performance Co-Pilot configuration on workstation:

platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents
pmda: root pmcd proc xfs linux mmv kvm jbd2
```

추가 리소스

- pmcd(1) 도움말 페이지
- PCP로 배포되는 툴

9.2. PMINFO를 사용하여 XFS 성능 지표 검사

PCP를 사용하면 XFS PMDA가 마운트된 각 XFS 파일 시스템별로 특정 XFS 지표를 보고할 수 있습니다. 이렇게 하면 마운트된 특정 파일 시스템 문제를 파악하고 성능을 쉽게 평가할 수 있습니다.

pminfo 명령은 마운트된 각 XFS 파일 시스템에 대해 장치별 XFS 지표를 제공합니다.

다음 절차에서는 XFS PMDA에서 제공하는 사용 가능한 모든 지표 목록을 표시합니다.

사전 요구 사항

- PCP가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.

절차

- XFS PMDA에서 제공하는 사용 가능한 모든 지표 목록을 표시합니다.

```
# pminfo xfs
```

- 개별 지표에 대한 정보를 표시합니다. 다음 예제에서는 pminfo 툴을 사용하여 특정 XFS 읽기 및 쓰기 지표를 검사합니다.
  - xfs.write_bytes 메트릭에 대한 간단한 설명을 표시합니다.

```
# pminfo --oneline xfs.write_bytes

xfs.write_bytes [number of bytes written in XFS file system write operations]
```

  - xfs.read_bytes 메트릭에 대한 자세한 설명을 표시합니다.

```
# pminfo --helptext xfs.read_bytes

xfs.read_bytes
Help:
This is the number of bytes read via read(2) system calls to files in
XFS file systems. It can be used in conjunction with the read_calls
count to calculate the average size of the read operations to file in
XFS file systems.
```

- **xfs.read_bytes** 지표의 현재 성능 값을 가져옵니다.

```
# pminfo --fetch xfs.read_bytes

xfs.read_bytes
    value 4891346238
```

- **pminfo** 를 사용하여 장치별 **XFS** 지표를 가져옵니다.

```
# pminfo --fetch --oneline xfs.perdev.read xfs.perdev.write

xfs.perdev.read [number of XFS file system read operations]
inst [0 or "loop1"] value 0
inst [0 or "loop2"] value 0

xfs.perdev.write [number of XFS file system write operations]
inst [0 or "loop1"] value 86
inst [0 or "loop2"] value 0
```

추가 리소스

- **pminfo(1)** 도움말 페이지
- XFS용 PCP 지표 그룹
- XFS의 장치당 PCP 지표 그룹

9.3. PMSTORE를 사용하여 XFS 성능 지표 재설정

PCP를 사용하면 메트릭이 **xfs.control.reset** 메트릭과 같은 제어 변수 역할을 하는 경우 특정 메트릭의 값을 수정할 수 있습니다. 지표 값을 수정하려면 **pmstore** 도구를 사용합니다.

다음 절차에서는 **pmstore** 툴을 사용하여 **XFS** 지표를 재설정하는 방법을 설명합니다.

사전 요구 사항

- **PCP**가 설치되어 있습니다. 자세한 내용은 **PCP** 설치 및 활성화를 참조하십시오.

절차

1. 지표 값을 표시합니다.

```
$ pminfo -f xfs.write

xfs.write
    value 325262
```

2. **XFS** 지표를 모두 재설정합니다.

```
# pmstore xfs.control.reset 1

xfs.control.reset old value=0 new value=1
```

검증 단계

- 메트릭을 재설정한 후 정보를 확인합니다.

```
$ pminfo --fetch xfs.write

xfs.write
    value 0
```

추가 리소스

- **pmstore(1)** 및 **pminfo(1)** 도움말 페이지
- **PCP**로 배포되는 툴
- **XFS**용 **PCP** 지표 그룹

9.4. XFS용 PCP 지표 그룹

다음 표에서는 XFS에 사용할 수 있는 PCP 지표 그룹을 설명합니다.

표 9.1. XFS의 지표 그룹

메트릭 그룹	제공되는 지표
XFS.*	읽기 및 쓰기 작업 수, 바이트 수 읽기 및 쓰기 횟수를 포함한 일반 XFS 지표. inode 수에 대한 카운터와 함께 플러시, 클러스터링 및 클러스터 실패 횟수가 함께 제공됩니다.
XFS.allocs.* **xfs.alloc_btree.***	파일 시스템에서 개체 할당과 관련된 지표 범위로, 여기에는 익스텐트 및 블록 생성/무료 수가 포함됩니다. btree에서 레코드 생성 및 삭제 확장과 함께 트리 조회 할당 및 비교.
xfs.block_map.* **xfs.bmap_btree.***	지표에는 읽기/쓰기 및 블록 삭제, 삽입, 삭제 및 조회에 대한 익스텐트 목록 작업, 블록 맵 수가 포함됩니다. 블록 맵의 비교, 조회, 삽입 및 삭제 작업을 위한 작업 카운터도 있습니다.
xfs.dir_ops.*	생성, 항목 삭제, "입력" 작업 수에 대한 XFS 파일 시스템의 디렉터리 작업 카운터입니다.
XFS.transactions.*	메타 데이터 트랜잭션 수에 대한 카운터에는 동기 및 비동기 트랜잭션 수와 빈 트랜잭션 수가 포함됩니다.
xfs.inode_ops.*	운영 체제에서 다양한 결과가 있는 inode 캐시에서 XFS inode를 찾는 횟수의 카운터입니다. 이러한 개수 캐시 적중, 캐시 누락 등.
xfs.log.* **xfs.log_tail.***	XFS 파일 스트림을 통한 로그 버퍼 쓰기 횟수에 대한 카운터에는 디스크에 작성된 블록 수가 포함됩니다. 또한 로그 플러시 및 고정 수에 대한 지표입니다.
xfs.xstrat.*	XFS 플러시 데밍에 의해 플러시되는 파일 데이터 바이트 수와 디스크의 연속되지 않은 공간으로 플러시되는 버퍼 수에 대한 카운터를 계산합니다.
xfs.attr.*	모든 XFS 파일 시스템에 대한 get, set, remove 및 list 속성 수에 대한 개수입니다.
xfs.quota.*	XFS 파일 시스템을 통한 할당량 작업에 대한 메트릭에는 할당량 회수, 할당량 캐시 누락, 캐시 적중 및 할당량 데이터 회수 횟수에 대한 카운터가 포함됩니다.

XFS.buffer.*	XFS 버퍼 오브젝트와 관련된 지표 범위. 카운터에는 페이지를 찾을 때 요청된 버퍼 호출 수, 성공적인 버퍼 잠금, 대기된 버퍼 잠금, 누락_locks, miss_retries 및 버퍼 적중이 포함됩니다.
XFS.btree.*	XFS btree의 작업과 관련된 지표입니다.
xfs.control.reset	XFS 통계의 지표 카운터를 재설정하는 데 사용되는 구성 지표입니다. 제어 지표는 pmstore 툴을 통해 토글됩니다.

9.5. XFS의 장치당 PCP 지표 그룹

다음 테이블에서는 XFS에 사용할 수 있는 장치별 PCP 지표 그룹을 설명합니다.

표 9.2. XFS의 장치당 PCP 지표 그룹

메트릭 그룹	제공되는 지표
xfs.perdev.*	읽기 및 쓰기 작업 수, 바이트 수 읽기 및 쓰기 횟수를 포함한 일반 XFS 지표. inode 수에 대한 카운터와 함께 플러시, 클러스터링 및 클러스터 실패 횟수가 함께 제공됩니다.
xfs.perdev.allocs.* **xfs.perdev.alloc_btree.***	파일 시스템에서 개체 할당과 관련된 지표 범위로, 여기에는 익스텐트 및 블록 생성/무료 수가 포함됩니다. btree에서 레코드 생성 및 삭제 확장과 함께 트리 조회 할당 및 비교.
xfs.perdev.block_map.* **xfs.perdev.bmap_btree.***	지표에는 읽기/쓰기 및 블록 삭제, 삽입, 삭제 및 조회에 대한 익스텐트 목록 작업, 블록 맵 수가 포함됩니다. 블록 맵의 비교, 조회, 삽입 및 삭제 작업을 위한 작업 카운터도 있습니다.
xfs.perdev.dir_ops.*	생성, 항목 삭제, "입력" 작업 수에 대한 XFS 파일 시스템의 디렉터리 작업 카운터입니다.
xfs.perdev.transactions.*	메타 데이터 트랜잭션 수에 대한 카운터에는 동기 및 비동기 트랜잭션 수와 빈 트랜잭션 수가 포함됩니다.
xfs.perdev.inode_ops.*	운영 체제에서 다양한 결과가 있는 inode 캐시에서 XFS inode를 찾는 횟수의 카운터입니다. 이러한 개수 캐시 적중, 캐시 누락 등.

xfs.perdev.log.* **xfs.perdev.log_tail.***	XFS 파일보다 로그 버퍼 쓰기 횟수에 대한 카운터에는 디스크에 작성된 블록 수가 포함됩니다. 또한 로그 플러시 및 고정 수에 대한 지표입니다.
xfs.perdev.xstrat.*	XFS 플러시 데밍에 의해 플러시되는 파일 데이터 바이트 수와 디스크의 연속되지 않은 공간으로 플러시되는 버퍼 수에 대한 카운터를 계산합니다.
xfs.perdev.attr.*	모든 XFS 파일 시스템에 대한 get, set, remove 및 list 속성 수에 대한 개수입니다.
xfs.perdev.quota.*	XFS 파일 시스템을 통한 할당량 작업에 대한 메트릭에는 할당량 회수, 할당량 캐시 누락, 캐시 적중 및 할당량 데이터 회수 횟수에 대한 카운터가 포함됩니다.
xfs.perdev.buffer.*	XFS 버퍼 오브젝트와 관련된 지표 범위. 카운터에는 페이지를 찾을 때 요청된 버퍼 호출 수, 성공적인 버퍼 잠금, 대기된 버퍼 잠금, 누락_locks, miss_retries 및 버퍼 적중이 포함됩니다.
xfs.perdev.btree.*	XFS btree의 작업과 관련된 지표입니다.

10장. PCP 메트릭의 그래픽 표현 설정

pcp,grafana, pcp redis, pcp bpftrace 및 pcp 벡터 의 조합을 사용하면 PCP(Performance Co-Pilot)에서 수집한 라이브 데이터 또는 데이터에 따라 그래프가 제공됩니다.

이 섹션에서는 PCP 지표의 그래픽 표현을 설정하고 액세스하는 방법을 설명합니다.

10.1. PCP-ZEROCONF를 사용하여 PCP 설정

다음 절차에서는 pcp-zeroconf 패키지를 사용하여 시스템에 PCP를 설정하는 방법을 설명합니다. pcp-zeroconf 패키지가 설치되면 시스템은 기본 지표 집합을 보관된 파일에 기록합니다.

절차

- pcp-zeroconf 패키지를 설치합니다.

```
# yum install pcp-zeroconf
```

검증 단계

- pmlogger 서비스가 활성 상태인지 확인하고 메트릭을 보관하기 시작합니다.

```
# pcp | grep pmlogger
 pmlogger: primary logger:
/var/log/pcp/pmlogger/localhost.localdomain/20200401.00.12
```

추가 리소스

- pmlogger 도움말 페이지
- Performance Co-Pilot을 통한 성능 모니터링

10.2. GRAFANA-SERVER 설정

Grafana는 브라우저에서 액세스할 수 있는 그래프를 생성합니다. grafana-server 는 Grafana 대시보드의 백엔드 서버입니다. 기본적으로 모든 인터페이스에서 수신 대기하고 웹 브라우저를 통해 액세스하는 웹 서비스를 제공합니다. grafana-pcp 플러그인은 백엔드의 pmproxy 프로토콜과 상호 작용합니다.

이 절차에서는 **grafana-server** 를 설정하는 방법을 설명합니다.

사전 요구 사항

- **PCP**가 구성되어 있습니다. 자세한 내용은 **pcp-zeroconf**를 사용하여 **PCP** 설정을 참조하십시오.

절차

1. 다음 패키지를 설치합니다.

```
# yum install grafana grafana-pcp
```

2. 다음 서비스를 재시작하고 활성화합니다.

```
# systemctl restart grafana-server
# systemctl enable grafana-server
```

3. **Grafana** 서비스에 대한 네트워크 트래픽에 대한 서버의 방화벽을 엽니다.

```
# firewall-cmd --permanent --add-service=grafana
success

# firewall-cmd --reload
success
```

검증 단계

- **grafana-server** 가 수신 대기하고 요청에 응답하는지 확인합니다.

```
# ss -ntlp | grep 3000
LISTEN  0  128  *:3000  *:*  users:(("grafana-server",pid=19522,fd=7))
```

- **grafana-pcp** 플러그인이 설치되어 있는지 확인합니다.

```
# grafana-cli plugins ls | grep performancecopilot-pcp-app

performancecopilot-pcp-app @ 3.1.0
```

추가 리소스

- pmproxy(1) 및 grafana-server 도움말 페이지

10.3. GRAFANA 웹 UI에 액세스

다음 절차에서는 Grafana 웹 인터페이스에 액세스하는 방법을 설명합니다.

Grafana 웹 인터페이스를 사용하여 다음을 수행할 수 있습니다.

- PCP Redis, PCP bpftrace 및 PCP 벡터 데이터 소스 추가
- 대시보드 만들기
- 유용한 메트릭의 개요 보기
- PCP Redis에서 경고 생성

사전 요구 사항

1. PCP가 구성되어 있습니다. 자세한 내용은 pcp-zeroconf를 사용하여 PCP 설정을 참조하십시오.
2. grafana-server 가 구성되어 있습니다. 자세한 내용은 Settings up a grafana-server 를 참조하십시오.

절차

1. 클라이언트 시스템에서 브라우저를 열고 *http://192.0.2.0*:3000 링크를 사용하여 포트 3000에서 grafana-server 에 액세스합니다.

   192.0.2.0 을 사용자 시스템 IP로 바꿉니다.

2. 첫 번째 로그인에 대해 Email(이메일) 또는 username (사용자 이름) 및 Password(암호) 필

드에 모두 **admin** 을 입력합니다.

Grafana는 보안 계정을 생성하기 위해 새 암호를 설정하라는 메시지를 표시합니다. 나중에 설정하려면 **Skip** (뛰기)를 클릭합니다.

3. 메뉴에서

구성 아이콘을 마우스로 이동한 다음 플러그 인 을 클릭합니다.

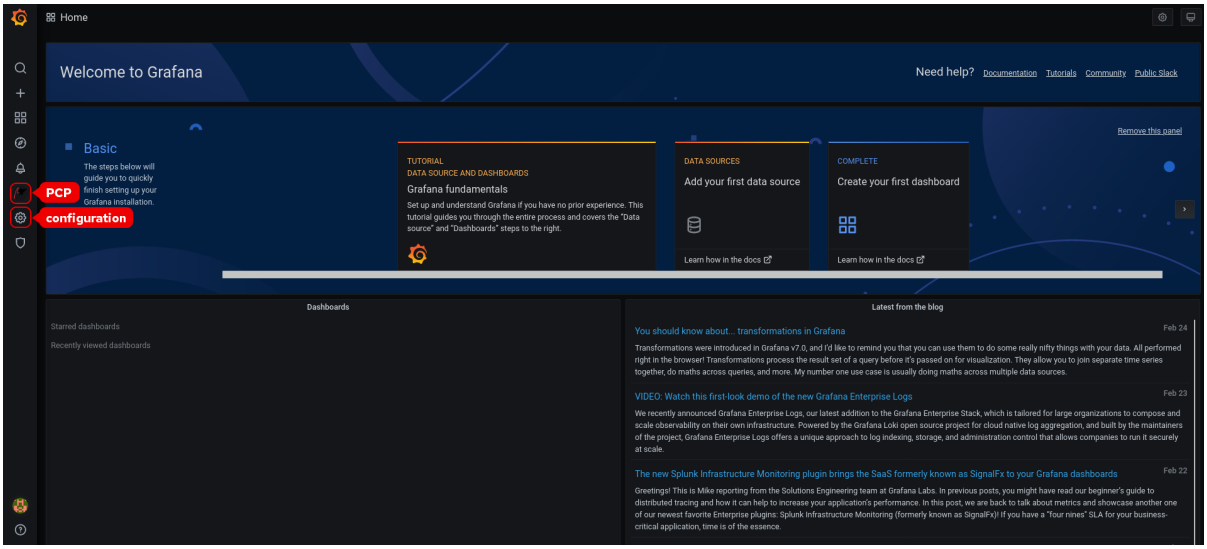
4. 플러그 인 탭에서 **Search by name**(이름으로 검색)에 **performance co-pilot**을 입력하거나 텍스트 상자를 입력한 다음 **PCP(Performance Co-Pilot)** 플러그인을 클릭합니다.

5. **Plugins / Performance Co-Pilot** 창에서 **Enable**(활성화)을 클릭합니다.

6. **Grafana**

아이콘을 클릭합니다. **Grafana** 홈 페이지가 표시됩니다.

그림 **10.1.** 홈 대시보드



참고

화면의 상단 모서리에는

아이콘이 비슷하지만 일반 대시보드 설정을 제어합니다.

7. **Grafana** 홈 페이지에서 첫 번째 데이터 소스 추가를 클릭하여 **PCP Redis, PCP bpftrace** 및 **PCP Vector** 데이터 소스를 추가합니다. 데이터 소스 추가에 대한 자세한 내용은 다음을 참조하십시오.

   - **pcp redis** 데이터 소스를 추가하려면 기본 대시보드 보기, 패널 만들기 및 경고 규칙을 보려면 **PCP Redis** 데이터 소스에서 패널 및 경고를 생성합니다.
   - **pcp bpftrace** 데이터 소스를 추가하고 기본 대시보드를 보려면 **PCP bpftrace System Analysis** 대시보드 보기를 참조하십시오.
   - **pcp** 벡터 데이터 소스를 추가하려면 기본 대시보드를 보고 벡터 체크리스트를 보려면 **PCP** 벡터 체크리스트 보기를 참조하십시오.

8. 선택 사항: 메뉴에서 **admin** 프로필

아이콘 위로 마우스를 가져가면 **Edit Profile** (프로필 편집),**Change Password** (암호 변경) 또는 **Sign out** (로그아웃)을 포함한 **Preferences** (기본 설정)를 변경합니다.

추가 리소스

- **grafana-cli** 및 **grafana-server** 도움말 페이지

10.4. PCP REDIS 구성

이 섹션에서는 **PCP Redis** 데이터 소스를 구성하는 방법에 대해 설명합니다.

PCP Redis 데이터 소스를 사용하여 다음을 수행합니다.

- 데이터 아카이브 보기
- **pmseries** 언어를 사용하여 시계열 쿼리
- 여러 호스트에서 데이터 분석

사전 요구 사항

1. **PCP**가 구성되어 있습니다. 자세한 내용은 **pcp-zeroconf**를 사용하여 **PCP** 설정을 참조하십시오.
2. **grafana-server** 가 구성되어 있습니다. 자세한 내용은 **Settings up a grafana-server** 를 참조하십시오.

절차

1. **redis** 패키지를 설치합니다.

```
# yum install redis
```

2. 다음 서비스를 시작하고 활성화합니다.

```
# systemctl start pmproxy redis
# systemctl enable pmproxy redis
```

3. 메일 전송 에이전트(예: **sendmail** 또는 **postfix**)가 설치 및 구성되어 있습니다.
4. **grafana.ini** 파일에서 **allow_loading_unsigned_plugins** 매개변수가 **PCP Redis** 데이터베이스로 설정되어 있는지 확인합니다.

```
# vi /etc/grafana/grafana.ini

allow_loading_unsigned_plugins = pcp-redis-datasource
```

5. **grafana-server** 를 다시 시작합니다.

```
# systemctl restart grafana-server
```

검증 단계

- **pmproxy** 및 **redis** 가 작동 중인지 확인합니다.

```
# pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

redis 패키지가 설치되지 않은 경우 이 명령은 데이터를 반환하지 않습니다.

추가 리소스

- **PMseries(1)** 도움말 페이지

10.5. PCP REDIS 데이터 소스에서 패널 및 경고 생성

PCP Redis 데이터 소스를 추가한 후에는 유용한 지표의 개요로 대시보드를 보고, 쿼리를 추가하여 부하 그래프를 시각화하고, 발생한 후 시스템 문제를 확인하는 데 도움이 되는 경고를 만들 수 있습니다.

사전 요구 사항

1. **PCP Redis**가 구성됩니다. 자세한 내용은 **PCP Redis** 구성을 참조하십시오.
2. **grafana-server** 에 액세스할 수 있습니다. 자세한 내용은 **Grafana** 웹 **UI** 액세스를 참조하십시오.

절차

1. **Grafana** 웹 **UI**에 로그인합니다.
2. **Grafana** 홈 페이지에서 첫 번째 데이터 소스 추가를 클릭합니다.
3. **Add data source** (데이터 소스 추가) 창에서 **Filter by name**(이름으로 필터링)에 **redis**를 입력하거나 텍스트 상자를 입력한 다음 **PCP Redis** 를 클릭합니다.

4.

Data Sources / PCP Redis 창에서 다음을 수행합니다.

a.

URL 필드에 **http://localhost:44322** 을 추가한 다음 **Save & Test** (저장 및 테스트)를 클릭합니다.

b.

Dashboards 탭 → **Import** → **PCP Redis**를 클릭합니다. 호스트 개요 는 유용한 메트릭에 대한 개요가 있는 대시보드를 확인합니다.

그림 10.2. PCP 빨간색: 호스트 개요

5.

새 패널을 추가합니다.

a.

메뉴에서

Create icon Dashboard (대시보드 만들기) 아이콘 → **Add new panel**(새 패널 추가) 아이콘을 커서로 이동하여 패널을 추가합니다.

b.

쿼리 탭의 선택한 기본 옵션 대신 쿼리 목록에서 **PCP Redis** 를 선택하고 텍스트 필드에서 **metric** (예: **kernel .all.load**)를 입력하여 커널 로드 그래프를 시각화합니다.

c.

선택 사항: 패널 제목과 설명을 추가하고 **Settings** (설정)에서 다른 옵션을 업데이트합니다.

d.

Save(저장)를 클릭하여 변경 사항을 적용하고 대시보드를 저장합니다. 대시보드 이름 추가.

e.

Apply(적용)를 클릭하여 변경 사항을 적용하고 대시보드로 돌아갑니다.

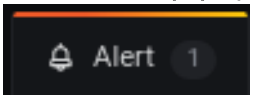
그림 10.3. PCP Redis 쿼리 패널

6.

경고 규칙을 생성합니다.

a.

PCP Redis 쿼리 패널 에서



경고를 클릭한 다음 **Create Alert** (경고 생성)를 클릭합니다.

b.

Rule (규칙)에서 **Name** (이름),**Evaluate**(평가 쿼리) 및 **For** (예) 필드를 편집하고 경고에 대한 **Conditions** (조건)를 지정합니다.

c.

Save(저장)를 클릭하여 변경 사항을 적용하고 대시보드를 저장합니다. **Apply**(적용)를 클릭하여 변경 사항을 적용하고 대시보드로 돌아갑니다.

그림 10.4. PCP Redis 패널에서 경고 생성

d.

선택 사항: 동일한 패널에서 아래로 스크롤하고 **Delete**(삭제) 아이콘을 클릭하여 생성된 규칙을 삭제합니다.

e.

선택 사항: 메뉴에서

경고 아이콘을 클릭하여 다양한 경고 상태로 생성된 경고 규칙을 보거나, 경고 규칙을 편집하거나, **Alert Rules** (경고 규칙) 탭에서 기존 규칙을 일시 중지합니다.

Grafana에서 경고 알림을 받기 위해 생성된 경고 규칙에 대한 알림 채널을 추가하려면 알림 채널 추가를 참조하십시오.

10.6. 경고에 대한 알림 채널 추가

알림 채널을 추가하면 경고 규칙 조건이 충족되고 시스템에서 추가 모니터링이 필요한 경우 **Grafana**로부터 알림 알림을 받을 수 있습니다.

이 경고는 **DingDing,Discord,Email,Google¢outs Chat,HipChat,Kafka REST Proxy,,Microsoft Teams,OpsGenie** 가 포함된 지원되는 알림기 목록에서 한 가지 유형을 선택한 후 수신할 수 있습니다. **PagerDuty,Prometheus Alertmanager,Pushover,Sensu,Slack,, threema Gateway,VictorOps** 및 **Webhook.**

사전 요구 사항

1.

grafana-server 에 액세스할 수 있습니다. 자세한 내용은 Grafana 웹 UI 액세스를 참조하십시오.

2. 경고 규칙이 생성됩니다. 자세한 내용은 **PCP Redis 데이터 소스에서 패널 및 경고 생성을** 참조하십시오.

3. **SMTP**를 구성하고 **grafana/grafana.ini** 파일에 유효한 발신자의 이메일 주소를 추가합니다.

```
# vi /etc/grafana/grafana.ini

[smtp]
enabled = true
from_address = abc@gmail.com
```

abc@gmail.com 을 유효한 이메일 주소로 바꿉니다.

절차

1. 메뉴에서 
경고 아이콘 → 알림 채널 추가 채널을클릭합니다.

2. **Add notification** 채널 세부 정보 창에서 다음을 수행합니다.

   a. **Name** (이름) 텍스트 상자에 이름을 입력합니다.

   b. 통신 유형 (예: **Email**)을 선택하고 이메일 주소를 입력합니다. 구분 기호를 사용하여 여러 이메일 주소를 추가할 수 있습니다.

   c. 선택 사항: 선택적 이메일 설정 및 알림 설정 구성.

3. 저장을 클릭합니다.

4. 경고 규칙에서 알림 채널을 선택합니다.

   a. 메뉴에서

경고 아이콘을 마우스로 이동한 다음 **Alert** 규칙을 클릭합니다.

b. **Alert Rules**(경고 규칙) 탭에서 생성된 경고 규칙을 클릭합니다.

c. **Notifications** (알림) 탭의 **Send to** (보내기) 옵션에서 알림 채널 이름을 선택한 다음 경고 메시지를 추가합니다.

d. **Apply**(적용)를 클릭합니다.

추가 리소스

- 경고 알림에 대한 업스트림 **Grafana** 문서

10.7. PCP 구성 요소 간 인증 설정

SASL(Simple Authentication Security Layer) 프레임워크를 통해 **PCP**에서 지원하는 **scram-sha-256** 인증 메커니즘을 사용하여 인증을 설정할 수 있습니다.

참고

Red Hat Enterprise Linux 8.3에서 **PCP**는 **scram-sha-256** 인증 메커니즘을 지원합니다.

절차

1. **scram-sha-256** 인증 메커니즘을 위한 **the sasl** 프레임워크를 설치합니다.

```
# yum install cyrus-sasl-scram cyrus-sasl-lib
```

2. **pmcd.conf** 파일에서 지원되는 인증 메커니즘 및 사용자 데이터베이스 경로를 지정합니다.

```
# vi /etc/sasl2/pmcd.conf

mech_list: scram-sha-256
```

```
sasldb_path: /etc/pcp/passwd.db
```

3. 새 사용자를 생성합니다.

```
# useradd -r metrics
```

메트릭을 사용자 이름으로 바꿉니다.

4. 사용자 데이터베이스에 생성된 사용자를 추가합니다.

```
# saslpasswd2 -a pmcd metrics

Password:
Again (for verification):
```

생성된 사용자를 추가하려면 *지표* 계정 암호를 입력해야 합니다.

5. 사용자 데이터베이스의 권한을 설정합니다.

```
# chown root:pcp /etc/pcp/passwd.db
# chmod 640 /etc/pcp/passwd.db
```

6. **pmcd** 서비스를 다시 시작하십시오.

```
# systemctl restart pmcd
```

검증 단계

- the sasl 구성을 확인합니다.

```
# pminfo -f -h "pcp://127.0.0.1?username=metrics" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

추가 리소스

- **saslauthd(8), pminfo(1) 및 sha256 도움말 페이지**

- **RHEL 8.2의 PMDA 및 pmcd와 같은 PCP 구성 요소 간에 인증을 설정하려면 어떻게 해야 합니까?**

10.8. PCP BPFTRACE 설치

PCP bpftrace 에이전트를 설치하여 시스템을 세부 검사하고 커널 및 사용자 공간 추적 지점에서 지표를 수집합니다.

bpftrace 에이전트는 **bpftrace** 스크립트를 사용하여 지표를 수집합니다. **bpftrace** 스크립트는**eBPF(Berkeley Packet Filter)**를 사용합니다.

다음 절차에서는 **pcp bpftrace** 를 설치하는 방법을 설명합니다.

사전 요구 사항

1. **PCP**가 구성되어 있습니다. 자세한 내용은 **pcp-zeroconf**를 사용하여 **PCP** 설정을 참조하십시오.

2. **grafana-server** 가 구성되어 있습니다. 자세한 내용은 **Settings up a grafana-server** 를 참조하십시오.

3. **scram-sha-256** 인증 메커니즘이 구성되어 있습니다. 자세한 내용은 **PCP** 구성 요소 간 인증 설정을 참조하십시오.

절차

1. **pcp-pmda-bpftrace** 패키지를 설치합니다.

```
# yum install pcp-pmda-bpftrace
```

2. **bpftrace.conf** 파일을 편집하고 **{setting-up-authentication- betweenween-pcp-components}**에서 생성한 사용자를 추가합니다.

```
# vi /var/lib/pcp/pmdas/bpftrace/bpftrace.conf

[dynamic_scripts]
enabled = true
auth_enabled = true
allowed_users = root,metrics
```

메트릭을 사용자 이름으로 바꿉니다.

3. **bpftrace PMDA**를 설치합니다.

```
# cd /var/lib/pcp/pmdas/bpftrace/
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check bpftrace metrics have appeared ... 7 metrics and 6 values
```

pmda-bpftrace 가 이제 설치되었으며 사용자를 인증한 후에만 사용할 수 있습니다. 자세한 내용은 **PCP bpftrace System Analysis dashboard** 보기를 참조하십시오.

추가 리소스

- **pmdabpftrace(1)** 및 **bpftrace** 도움말 페이지

10.9. PCP BPFTRACE 시스템 분석 대시보드 보기

PCP bpftrace 데이터 소스를 사용하여 **pmlogger** 또는 아카이브에서 일반 데이터로 사용할 수 없는 소스에서 라이브 데이터에 액세스할 수 있습니다.

PCP bpftrace 데이터 소스에서 유용한 지표의 개요를 사용하여 대시보드를 볼 수 있습니다.

사전 요구 사항

1. **PCP bpftrace**가 설치되어 있습니다. 자세한 내용은 **PCP bpftrace** 설치를 참조하십시오.

2. **grafana-server** 에 액세스할 수 있습니다. 자세한 내용은 **Grafana 웹 UI 액세스**를 참조하십시오.

절차

1. **Grafana 웹 UI**에 로그인합니다.

2. **Grafana** 홈 페이지에서 첫 번째 데이터 소스 추가를 클릭합니다.

3. **Add data source** (데이터 소스 추가) 창에서 **Filter by name**(이름으로 필터링)에 **bpftrace**를 입력하거나 텍스트 상자에 **bpftrace**를 입력한 다음 **PCP bpftrace** 를 클릭합니다.

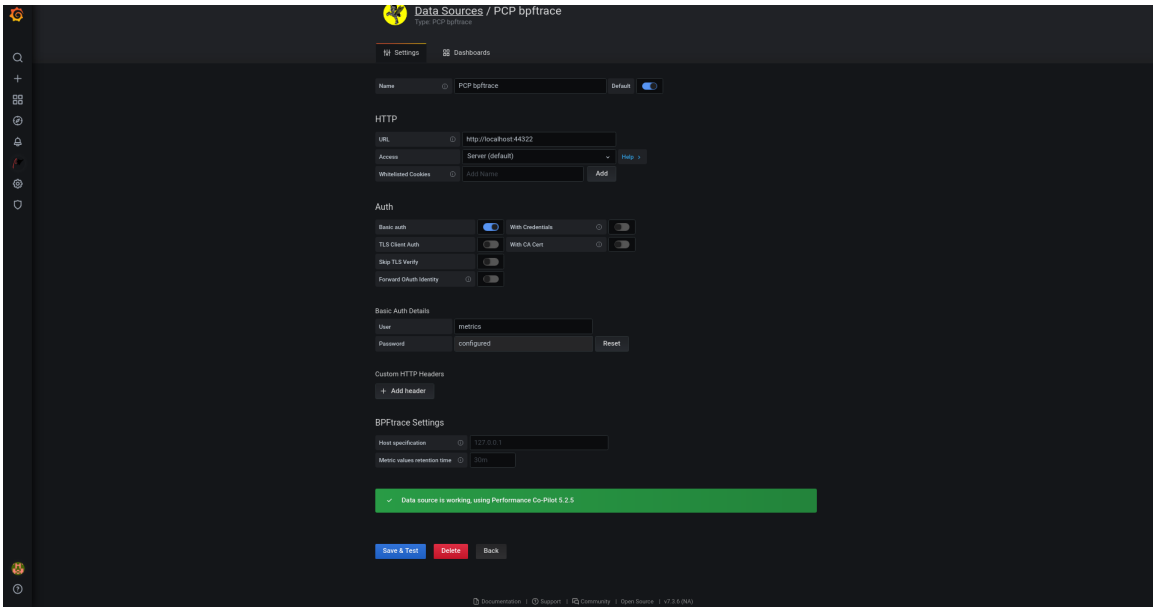
4. **Data Sources / PCP bpftrace** 창에서 다음을 수행합니다.

   a. **URL** 필드에 **http://localhost:44322** 을 추가합니다.

   b. **Basic Auth** (기본 인증) 옵션을 토글하고 **User and Password**(사용자 및 암호) 필드에 생성된 사용자 자격 증명을 추가합니다.

   c. **Save & Test** (저장 및 테스트)를 클릭합니다.

   그림 10.5. 데이터 소스에 **PCP bpftrace** 추가



   d. 대시보드 탭 → 가져오기 → **PCP bpftrace**를 클릭합니다. 시스템 분석 은/는 유용한 메트릭의 개요가 있는 대시보드를 확인합니다.

그림 10.6. PCP bpftrace: 시스템 분석

10.10. PCP 벡터 설치

다음 절차에서는 **pcp** 벡터 를 설치하는 방법을 설명합니다.

사전 요구 사항

1. **PCP**가 구성되어 있습니다. 자세한 내용은 **pcp-zeroconf**를 사용하여 **PCP** 설정을 참조하십시오.

2. **grafana-server** 가 구성되어 있습니다. 자세한 내용은 **Settings up a grafana-server** 를 참조하십시오.

절차

1. **pcp-pmda-bcc** 패키지를 설치합니다.

```
# yum install pcp-pmda-bcc
```

2. **bcc PMDA**를 설치합니다.

```
# cd /var/lib/pcp/pmdas/bcc
# ./Install
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: Initializing, currently in 'notready' state.
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: Enabled modules:
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: ['biolatency', 'sysfork',
```

```
[...]
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check bcc metrics have appeared ... 1 warnings, 1 metrics and 0 values
```

추가 리소스

- **pmdabcc(1)** 도움말 페이지

10.11. PCP 벡터 체크리스트 보기

PCP 벡터 데이터 소스는 라이브 지표를 표시하고 **pcp** 지표를 사용합니다. 개별 호스트에 대한 데이터를 분석합니다.

PCP 벡터 데이터 소스를 추가한 후에는 유용한 메트릭에 대한 개요를 통해 대시보드를 보고 체크리스트에서 관련 문제 해결 또는 참조 링크를 볼 수 있습니다.

사전 요구 사항

1. **PCP** 벡터가 설치되어 있습니다. 자세한 내용은 **PCP** 벡터 설치를 참조하십시오.
2. **grafana-server** 에 액세스할 수 있습니다. 자세한 내용은 **Grafana** 웹 **UI** 액세스를 참조하십시오.

절차

1. **Grafana** 웹 **UI**에 로그인합니다.
2. **Grafana** 홈 페이지에서 첫 번째 데이터 소스 추가를 클릭합니다.
3. **Add data source** (데이터 소스 추가) 창에서 **Filter by name**(이름으로 필터링)에ector를 입력하거나 텍스트 상자를 입력한 다음 **PCP Vector** (**PCP** 벡터)를 클릭합니다.
4. **Data Sources** / **PCP Vector** 창에서 다음을 수행합니다.
   a.

URL 필드에 **http://localhost:44322** 을 추가한 다음 **Save & Test** (저장 및 테스트)를 클릭합니다.

b.

대시보드 탭 → 가져오기 → **PCP** 벡터를 클릭합니다. 호스트 개요 는 유용한 메트릭에 대한 개요가 있는 대시보드를 확인합니다.

그림 10.7. PCP 벡터: 호스트 개요

5.

메뉴에서

Performance Co-Pilot 플러그 인을 마우스로 이동한 다음 **PCP** 벡터 검사 목록을 클릭합니다.

PCP 체크리스트에서

도움말 또는

경고 아이콘을 클릭하여 관련 문제 해결 또는 참조 링크를 확인합니다.

그림 10.8. Performance Co-Pilot / PCP 벡터 체크리스트

10.12. GRAFANA 문제 해결

이 섹션에서는 **Grafana** 문제와 같은 **Grafana** 문제를 해결하는 방법을 설명합니다(예: **Grafana**는 데이터를 표시하지 않음), 대시보드는 차단된 문제 또는 유사한 문제를 표시합니다.

절차

- 다음 명령을 실행하여 **pmlogger** 서비스가 시작되어 실행 중인지 확인합니다.

```
$ systemctl status pmlogger
```

- 다음 명령을 실행하여 파일이 생성되었는지 또는 디스크에 수정되었는지 확인합니다.

```
$ ls /var/log/pcp/pmlogger/$(hostname)/ -rlt
total 4024
-rw-r--r--. 1 pcp pcp   45996 Oct 13  2019 20191013.20.07.meta.xz
-rw-r--r--. 1 pcp pcp     412 Oct 13  2019 20191013.20.07.index
-rw-r--r--. 1 pcp pcp   32188 Oct 13  2019 20191013.20.07.0.xz
-rw-r--r--. 1 pcp pcp   44756 Oct 13  2019 20191013.20.30-00.meta.xz
[..]
```

- 다음 명령을 실행하여 **pmproxy** 서비스가 실행 중인지 확인합니다.

```
$ systemctl status pmproxy
```

- **pmproxy** 가 실행 중이고 시계열 지원이 활성화되었으며 **/var/log/pcp/pmproxy/pmproxy.log** 파일을 보고 다음 텍스트가 포함되어 있는지 확인하여

Redis에 대한 연결이 설정되었는지 확인합니다.

```
pmproxy(1716) Info: Redis slots, command keys, schema version setup
```

여기서 **1716** 은 **pmproxy**의 **PID**로, **pmproxy** 의 각 호출마다 다릅니다.

- 다음 명령을 실행하여 **Redis** 데이터베이스에 키가 있는지 확인합니다.

```
$ redis-cli dbsize
(integer) 34837
```

- **PCP** 지표가 **Redis** 데이터베이스에 있고 **pmproxy** 가 다음 명령을 실행하여 액세스할 수 있는지 확인합니다.

```
$ pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df

$ pmseries "disk.dev.read[count:10]"
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
   [Mon Jul 26 12:21:10.085468000 2021] 117971
70e83e88d4e1857a3a31605c6d1333755f2dd17c
   [Mon Jul 26 12:21:00.087401000 2021] 117758
70e83e88d4e1857a3a31605c6d1333755f2dd17c
   [Mon Jul 26 12:20:50.085738000 2021] 116688
70e83e88d4e1857a3a31605c6d1333755f2dd17c
[...]
```

```
$ redis-cli --scan --pattern "*$(pmseries 'disk.dev.read')"

pcp:metric.name:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:values:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:desc:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelvalue:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:instances:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelflags:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

- 다음 명령을 실행하여 **Grafana** 로그에 오류가 있는지 확인합니다.

```
$ journalctl -e -u grafana-server
-- Logs begin at Mon 2021-07-26 11:55:10 IST, end at Mon 2021-07-26 12:30:15 IST. --
Jul 26 11:55:17 localhost.localdomain systemd[1]: Starting Grafana instance...
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-
26T11:55:17+0530 lvl=info msg="Starting Grafana" logger=server version=7.3.6 c>
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-
26T11:55:17+0530 lvl=info msg="Config loaded from" logger=settings file=/usr/s>
```

```
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-
26T11:55:17+0530 lvl=info msg="Config loaded from" logger=settings file=/etc/g>
[...]
```

11장. 웹 콘솔을 사용하여 시스템 성능 최적화

RHEL 웹 콘솔에서 성능 프로파일을 설정하여 선택한 작업에 대한 시스템의 성능을 최적화하는 방법을 알아봅니다.

11.1. 웹 콘솔의 성능 튜닝 옵션

Red Hat Enterprise Linux 8은 다음 작업을 위해 시스템을 최적화하는 여러 성능 프로필을 제공합니다.

- 데스크탑을 사용하는 시스템
- 처리량 성능
- 대기 시간 성능
- 네트워크 성능
- 낮은 전력 소비
- 가상 머신

tuned 서비스는 선택한 프로필과 일치하도록 시스템 옵션을 최적화합니다.

웹 콘솔에서 시스템에서 사용하는 성능 프로필을 설정할 수 있습니다.

추가 리소스

- TuneD 시작하기

11.2. 웹 콘솔에서 성능 프로필 설정

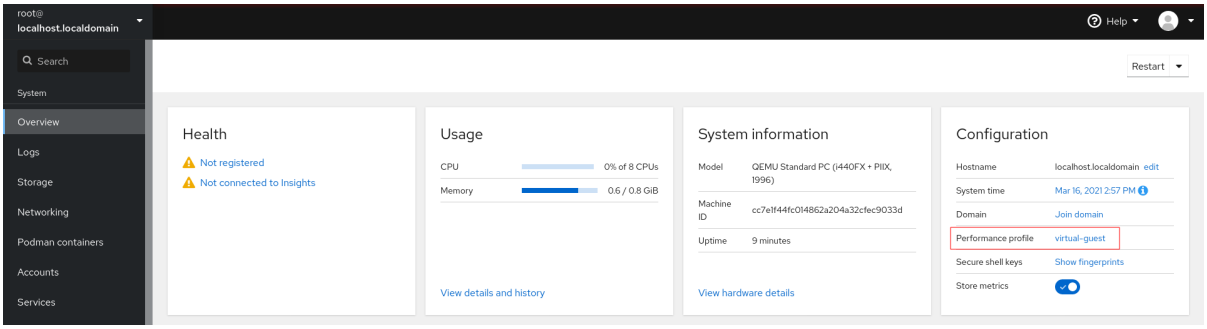
이 절차에서는 웹 콘솔을 사용하여 선택한 작업에 대한 시스템 성능을 최적화합니다.

사전 요구 사항

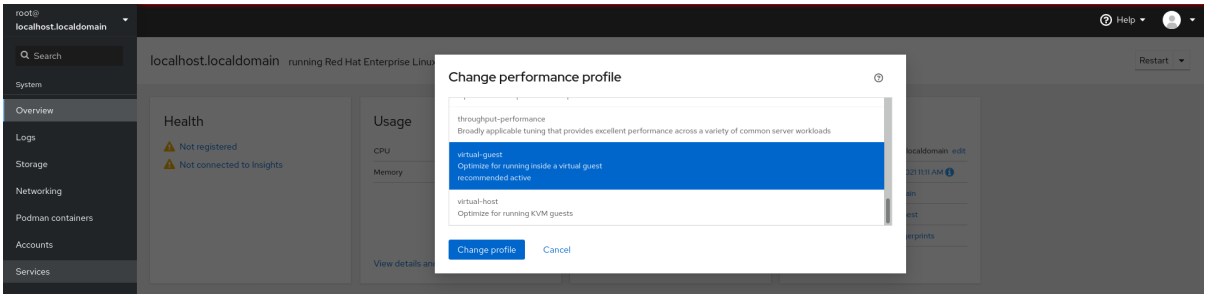
- 웹 콘솔이 설치되어 있고 액세스할 수 있는지 확인합니다. 자세한 내용은 웹 콘솔 설치를 참조하십시오.

절차

1. RHEL 웹 콘솔에 로그인합니다. 자세한 내용은 웹 콘솔에 로그인을 참조하십시오.
2. Overview(개요)를 클릭합니다.
3. Performance Profile (성능 프로필) 필드에서 현재 성능 프로필을 클릭합니다.



4. 필요한 경우 성능 프로필 변경 대화 상자에서 프로필을 변경합니다.
5. Change Profile (프로필 변경)을 클릭합니다.



검증 단계

- 이제 **Overview**(개요) 탭에 선택한 성능 프로필이 표시됩니다.

11.3. 웹 콘솔을 사용하여 성능 모니터링

Red Hat의 웹 콘솔에서는 문제 해결을 위해 **USE(Utilization Saturation and Errors)** 방법을 사용합니다. 새 성능 지표 페이지에는 최신 데이터와 함께 시간순으로 구성된 데이터 기록 보기가 있습니다.

여기에서 리소스 사용률 및 포화 상태에 대한 이벤트, 오류 및 그래픽 표현을 볼 수 있습니다.

사전 요구 사항

1. 웹 콘솔이 설치되어 있고 액세스할 수 있는지 확인합니다. 자세한 내용은 웹 콘솔 설치를 참조하십시오.

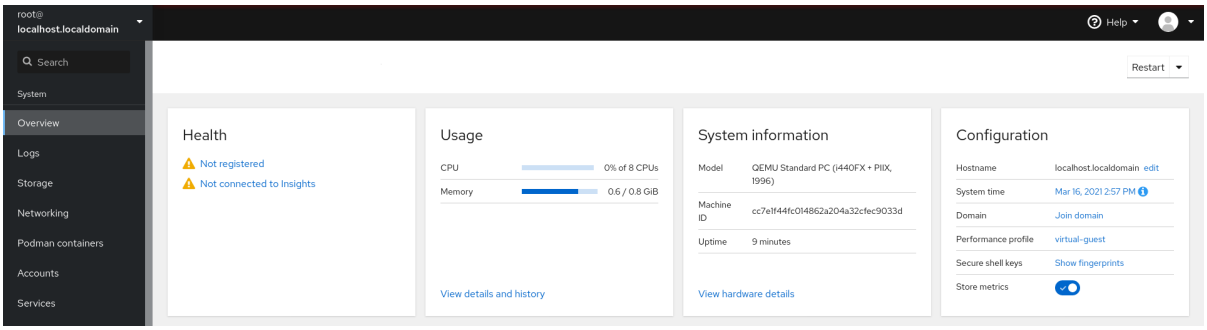
2. 성능 지표를 수집할 수 있는 **cockpit-pcp** 패키지를 설치합니다.

```
# yum install cockpit-pcp
```

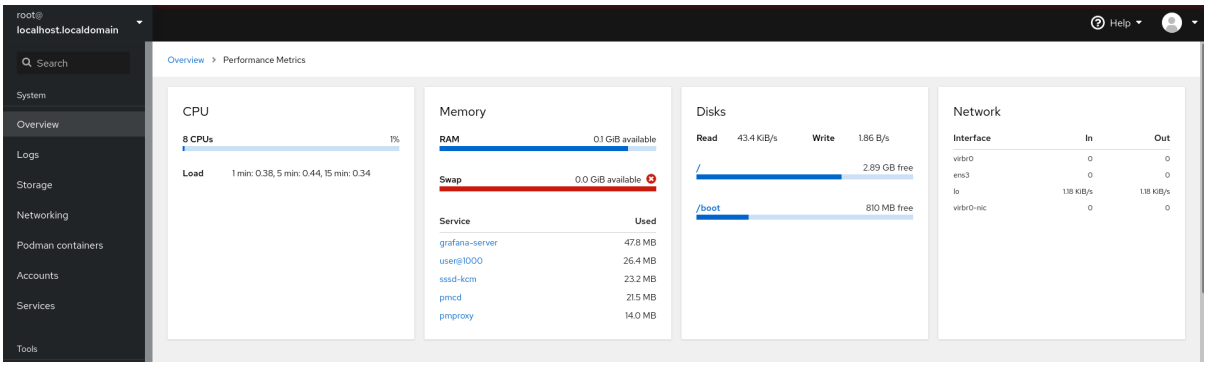
절차

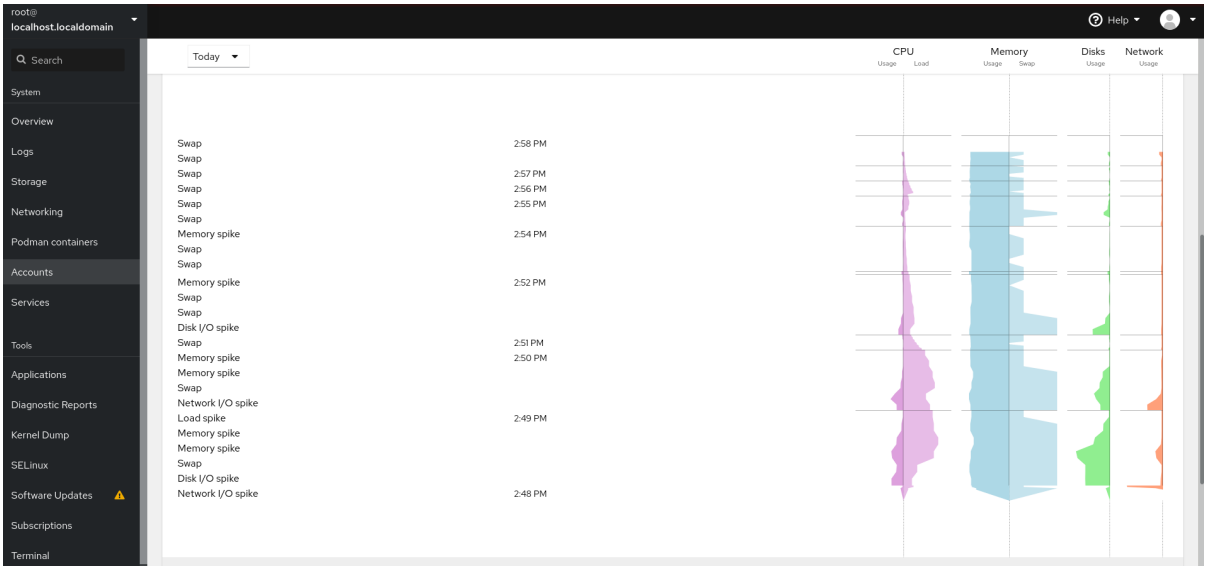
1. **RHEL 8** 웹 콘솔에 로그인합니다. 자세한 내용은 웹 콘솔에 로그인을 참조하십시오.

2. **Overview**(개요)를 클릭합니다.



3. **View details and history (세부 정보 및 기록 보기)**를 클릭하여 성능 지표를 확인합니다.





12장. 디스크 스케줄러 설정

디스크 스케줄러는 스토리지 장치에 제출된 I/O 요청을 주문합니다.

여러 가지 방법으로 스케줄러를 구성할 수 있습니다.

- TuneD 를 사용하여 디스크 스케줄러 설정에서 설명하는 TuneD를 사용하여 스케줄러를 설정합니다. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/monitoring_and_managing_system_status_and_performance/setting-the-disk-scheduler_monitoring-and-managing-system-status-and-performance#setting-the-disk-scheduler-using-tuned_setting-the-disk-scheduler

- udev 규칙을 사용하여 디스크 스케줄러 설정에 설명된 대로 udev를 사용하여 스케줄러 설정

- 특정 디스크에 대한 스케줄러를 일시적으로 설정하지 않은 경우 설명된 대로 실행 중인 시스템에서 스케줄러를 일시적으로변경합니다.

참고

Red Hat Enterprise Linux 8에서 블록 장치는 다중 대기열 스케줄링만 지원합니다. 이를 통해 솔리드 스테이트 드라이브(SSD) 및 멀티 코어 시스템으로 블록 계층 성능을 확장할 수 있습니다.

Red Hat Enterprise Linux 7 및 이전 버전에서 사용할 수 있는 기존의 단일 대기열 스케줄러가 제거되었습니다.

12.1. 사용 가능한 디스크 스케줄러

Red Hat Enterprise Linux 8에서는 다음과 같은 멀티 큐 디스크 스케줄러가 지원됩니다.

none

FIFO(선입선출) 스케줄링 알고리즘을 구현합니다. 간단한 마지막 캐시를 통해 일반 블록 계층의 요청을 병합합니다.

mq-deadline

요청이 스케줄러에 도달하는 시점의 요청에 대해 보장된 대기 시간을 제공합니다.

mq-deadline 스케줄러는 대기 중인 I/O 요청을 읽기 또는 쓰기 배치로 정렬한 다음, 논리 블록 주소 지정(LBA) 순서를 늘리도록 실행하도록 예약합니다. 애플리케이션이 읽기 I/O 작업에서 차단될 가능성이 높기 때문에 기본적으로 읽기 배치는 쓰기 배치보다 우선합니다. **mq-deadline** 이 배치를 처리한 후 쓰기 작업이 프로세서 시간이 지난 시간을 확인하고 다음 읽기 또는 쓰기 배치를 적절하게 예약합니다.

이 스케줄러는 대부분의 사용 사례에 적합하지만 특히 쓰기 작업이 주로 비동기식으로 사용됩니다.

bfq

데스크탑 시스템 및 대화형 작업을 대상으로 합니다.

bfq 스케줄러는 단일 애플리케이션이 모든 대역폭을 사용하지 않도록 합니다. 결과적으로 스토리지 장치는 항상 유휴 상태인 것처럼 반응합니다. 기본 구성에서 **bfq** 는 최대 처리량을 달성하는 대신 가장 낮은 대기 시간을 전달하는 데 중점을 둡니다.

BFQ 는 **cfq** 코드를 기반으로 합니다. 고정된 시간 슬라이스에 대해 각 프로세스에 디스크에 부여하지 않지만 섹터 수로 측정된 *예산을* 프로세스에 할당합니다.

이 스케줄러는 대용량 파일을 복사하는 동안 적합하며 이 경우 시스템이 응답하지 않습니다.

kyber

스케줄러는 블록 I/O 계층에 제출된 모든 I/O 요청 대기 시간을 계산하여 대기 시간 목표를 달성하도록 자체적으로 조정합니다. 캐시 누락 및 동기 쓰기 요청의 경우 읽기, 에 대한 대상 대기 시간을 구성할 수 있습니다.

이 스케줄러는 **NVMe, SSD** 또는 기타 짧은 대기 시간이 짧은 장치 등 빠른 장치에 적합합니다.

12.2. 다양한 사용 사례에 맞는 다른 디스크 스케줄러

시스템에서 수행하는 작업에 따라 분석 및 튜닝 작업 전에 다음 디스크 스케줄러를 기준으로 사용하는 것이 좋습니다.

표 12.1. 다양한 사용 사례의 디스크 스케줄러

사용 사례	디스크 스케줄러
SCSI 인터페이스가 있는 기존 HDD	**mq-deadline** 또는 **bfq** 를 사용합니다.
고성능 SSD 또는 빠른 스토리지가 있는 CPU 바운드 시스템	특히 엔터프라이즈 애플리케이션을 실행할 때는 **none** 을 사용합니다. 또는 **kyber** 를 사용합니다.
데스크탑 또는 대화형 작업	**bfq** 사용.
가상 게스트	**mq-deadline** 사용. 멀티 큐를 사용할 수 있는 HBA(호스트 버스 어댑터) 드라이버에서는 **none** 을 사용합니다.

12.3. 기본 디스크 스케줄러

블록 장치는 다른 스케줄러를 지정하지 않는 한 기본 디스크 스케줄러를 사용합니다.

참고

특히 비volatile Memory Express(NVMe) 블록 장치의 경우 기본 스케줄러는 **none** 이며 **Red Hat**은 변경하지 않는 것이 좋습니다.

커널은 장치 유형에 따라 기본 디스크 스케줄러를 선택합니다. 자동 선택된 스케줄러는 일반적으로 최적의 설정입니다. 다른 스케줄러가 필요한 경우 **Red Hat**은 **udev** 규칙 또는 **TuneD** 애플리케이션을 사용하여 구성하는 것이 좋습니다. 선택한 장치와 일치하고 해당 장치에 대해서만 스케줄러를 전환합니다.

12.4. 활성 디스크 스케줄러 확인

다음 절차에서는 지정된 블록 장치에서 현재 활성화된 디스크 스케줄러를 결정합니다.

절차

- **/sys/block/*device*/queue/scheduler** 파일의 내용을 읽습니다.

```
# cat /sys/block/device/queue/scheduler
[mq-deadline] kyber bfq none
```

파일 이름에서 ***device*** 를 **exampledc**의 블록 장치 이름으로 바꿉니다.

활성 스케줄러는 대괄호([])로 나열됩니다.

12.5. TUNED를 사용하여 디스크 스케줄러 설정

이 절차에서는 선택한 블록 장치에 지정된 디스크 스케줄러를 설정하는 TuneD 프로필을 생성하고 활성화합니다. 이 설정은 시스템을 재부팅해도 유지됩니다.

다음 명령 및 구성에서 교체합니다.

- 블록 *장치의* 이름이 있는 장치(예: sdf)
- 장치에 설정할 디스크 스케줄러가 있는 select *-scheduler* (예: bfq)

사전 요구 사항

- tuned 서비스가 설치 및 활성화되었습니다. 자세한 내용은 TuneD 설치 및 사용을 참조하십시오.

절차

1. 선택 사항: 프로필을 기반으로 할 기존 TuneD 프로필을 선택합니다. 사용 가능한 프로필 목록은 RHEL로 배포된 TuneD 프로필을 참조하십시오.

   현재 활성화된 프로필을 확인하려면 다음을 사용합니다.

   ```
   $ tuned-adm active
   ```

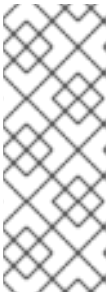
2. TuneD 프로필을 유지할 새 디렉토리를 만듭니다.

   ```
   # mkdir /etc/tuned/my-profile
   ```

3. 선택한 블록 장치의 시스템 고유 식별자를 찾습니다.

   ```
   $ udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'
   ```

```
ID_WWN=0x5002538d00000000_
ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
ID_SERIAL_SHORT=20120501030900000
```



참고

이 예제의 명령은 지정된 블록 장치와 연결된 **WWN(World Wide Name)** 또는 일련 번호로 식별된 모든 값을 반환합니다. **WWN**을 사용하는 것이 바람직하지만 **WWN**은 지정된 장치에 항상 사용 가능한 것은 아니며 예제 명령에서 반환된 값은 *장치 시스템 고유* ***ID*** 로 사용할 수 있습니다.

4. **/etc/tuned/*my-profile*/tuned.conf** 구성 파일을 만듭니다. 파일에서 다음 옵션을 설정합니다.

   a. 선택 사항: 기존 프로필을 포함합니다.

   ```
   [main]
   include=existing-profile
   ```

   b. **WWN** 식별자와 일치하는 장치에 대해 선택한 디스크 스케줄러를 설정합니다.

   ```
   [disk]
   devices_udev_regex=IDNAME=device system unique id
   elevator=selected-scheduler
   ```

   여기:

   - ***IDNAME*** 을 사용 중인 식별자의 이름으로 바꿉니다(예: **ID_WWN**).
   - *장치 시스템 고유* ***ID*** 를 선택한 식별자 값으로 바꿉니다(예: **0x5002538d0000**).

   devices_udev_regex 옵션의 여러 장치를 일치시키려면 식별자를 괄호로 묶고 세로 막대로 구분합니다.

   ```
   devices_udev_regex=(ID_WWN=0x5002538d00000000)|
   (ID_WWN=0x1234567800000000)
   ```

5. 프로필을 활성화합니다.

```
# tuned-adm profile my-profile
```

검증 단계

1. **TuneD** 프로필이 활성화되어 적용되었는지 확인합니다.

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

2. **/sys/block/***장치***/queue/scheduler** 파일의 내용을 읽습니다.

```
# cat /sys/block/device/queue/scheduler

[mq-deadline] kyber bfq none
```

파일 이름에서 ***device*** 를 **exampledc**의 블록 장치 이름으로 바꿉니다.

활성 스케줄러는 대괄호(**[]**)로 나열됩니다.

추가 리소스

- **TuneD** 프로필 사용자 지정.

12.6. UDEV 규칙을 사용하여 디스크 스케줄러 설정

이 절차에서는 **udev** 규칙을 사용하여 특정 블록 장치에 지정된 디스크 스케줄러를 설정합니다. 이 설정은 시스템을 재부팅해도 유지됩니다.

다음 명령 및 구성에서 교체합니다.

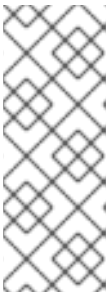
- 블록 *장치의* 이름이 있는 장치(예: **sdf**)

- 장치에 설정할 디스크 스케줄러가 있는 **select *-scheduler*** (예: **bfq**)

절차

1. 블록 장치의 시스템 고유 식별자를 찾습니다.

```
$ udevadm info --name=/dev/device | grep -E '(WWN|SERIAL)'
E: ID_WWN=0x5002538d00000000
E: ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
E: ID_SERIAL_SHORT=20120501030900000
```



참고

이 예제의 명령은 지정된 블록 장치와 연결된 **WWN(World Wide Name)** 또는 일련 번호로 식별된 모든 값을 반환합니다. **WWN**을 사용하는 것이 바람직하지만 **WWN**은 지정된 장치에 항상 사용 가능한 것은 아니며 예제 명령에서 반환된 값은 *장치 시스템 고유* ***ID*** 로 사용할 수 있습니다.

2. **udev** 규칙을 구성합니다. 다음 콘텐츠를 사용하여 **/etc/udev/rules.d/99-scheduler.rules** 파일을 생성합니다.

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{IDNAME}=="device system unique id", ATTR{queue/scheduler}="selected-scheduler"
```

여기:

- ***IDNAME*** 을 사용 중인 식별자의 이름으로 바꿉니다(예: **ID_WWN**).
- *장치 시스템 고유* ***ID*** 를 선택한 식별자 값으로 바꿉니다(예: **0x5002538d0000**).

3. **udev** 규칙을 다시 로드합니다.

```
# udevadm control --reload-rules
```

4. 스케줄러 구성을 적용합니다.

```
# udevadm trigger --type=devices --action=change
```

검증 단계

- 활성 스케줄러를 확인합니다.

```
# cat /sys/block/device/queue/scheduler
```

12.7. 특정 디스크에 대한 임시로 스케줄러 설정

이 절차에서는 특정 블록 장치에 지정된 디스크 스케줄러를 설정합니다. 이 설정은 시스템을 재부팅해도 유지되지 않습니다.

절차

- 선택한 스케줄러의 이름을 **/sys/block/*장치*/queue/scheduler** 파일에 작성합니다.

```
# echo selected-scheduler > /sys/block/device/queue/scheduler
```

파일 이름에서 ***device*** 를 **exampledc**의 블록 장치 이름으로 바꿉니다.

검증 단계

- 스케줄러가 장치에서 활성화되어 있는지 확인합니다.

```
# cat /sys/block/device/queue/scheduler
```

13장. SAMBA 서버의 성능 튜닝

이 장에서는 특정 상황에서 Samba의 성능을 향상시킬 수 있는 설정과 성능에 부정적인 영향을 미칠 수 있는 설정을 설명합니다.

이 섹션의 일부는 Samba Wiki에 게시된 성능 튜닝 설명서에서 채택되었습니다. 라이센스: CC BY 4.0. 저자 및 기여자: Wiki 페이지에서 히스토리 탭을 참조하십시오.

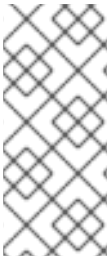
사전 요구 사항

- Samba가 파일 또는 인쇄 서버로 설정됨

  서버로 Samba 사용을 참조하십시오.

13.1. SMB 프로토콜 버전 설정

각각의 새 SMB 버전은 기능을 추가하고 프로토콜의 성능을 향상시킵니다. 최근 Windows 및 Windows Server 운영 체제는 항상 최신 프로토콜 버전을 지원합니다. Samba도 최신 프로토콜 버전을 사용하는 경우 Samba에 연결하는 Windows 클라이언트는 성능 향상을 통해 이점을 얻을 수 있습니다. Samba에서 서버 max 프로토콜의 기본값은 지원되는 최신 안정적인 SMB 프로토콜 버전으로 설정됩니다.



참고

항상 안정적인 최신 SMB 프로토콜 버전을 활성화하려면 server max protocol 매개 변수를 설정하지 마십시오. 매개 변수를 수동으로 설정하는 경우 최신 프로토콜 버전이 활성화되도록 각 새 버전의 SMB 프로토콜로 설정을 수정해야 합니다.

다음 절차에서는 server max protocol 매개 변수에서 기본값을 사용하는 방법을 설명합니다.

절차

1. /etc/samba/smb.conf 파일의 [global] 섹션에서 server max protocol 매개 변수를 제거합니다.

2. Samba 설정을 다시 로드합니다

```
# smbcontrol all reload-config
```

13.2. 많은 수의 파일이 포함된 디렉토리와의 공유 조정

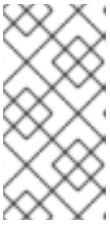
Linux는 대소문자를 구분하는 파일 이름을 지원합니다. 이러한 이유로 **Samba**는 파일을 검색하거나 액세스할 때 대문자 및 소문자로 된 파일 이름에 대해 디렉토리를 스캔해야 합니다. 소문자 또는 대문자로만 새 파일을 생성하도록 공유를 구성할 수 있으므로 성능이 향상됩니다.

사전 요구 사항

- **Samba**가 파일 서버로 구성되어 있습니다.

절차

1. 공유의 모든 파일의 이름을 소문자로 바꿉니다.



참고

이 절차의 설정을 사용하면 소문자가 아닌 이름이 있는 파일이 더 이상 표시되지 않습니다.

2. 공유의 섹션에서 다음 매개변수를 설정합니다.

```
case sensitive = true
default case = lower
preserve case = no
short preserve case = no
```

매개 변수에 대한 자세한 내용은 **smb.conf(5)** 도움말 페이지에서 해당 설명을 참조하십시오.

3. **/etc/samba/smb.conf** 파일을 확인합니다.

```
# testparm
```

4. **Samba** 구성을 다시 로드합니다.

```
# smbcontrol all reload-config
```

이러한 설정을 적용한 후 이 공유에서 새로 생성된 모든 파일의 이름은 소문자를 사용합니다. 이러한 설정으로 인해 **Samba**가 더 이상 디렉터리를 대문자 및 소문자로 스캔할 필요가 없으므로 성능이 향상됩니다.

13.3. 성능에 부정적인 영향을 줄 수 있는 설정

기본적으로 **Red Hat Enterprise Linux**의 커널은 높은 네트워크 성능을 위해 조정됩니다. 예를 들어 커널은 버퍼 크기에 자동 조정 메커니즘을 사용합니다. **/etc/samba/smb.conf** 파일에서 **socket options** 매개 변수를 설정하면 이러한 커널 설정이 재정의됩니다. 결과적으로 이 매개 변수를 설정하면 대부분의 경우 **Samba** 네트워크 성능이 저하됩니다.

커널에서 최적화된 설정을 사용하려면 **/etc/samba/smb.conf** 의 **[global]** 섹션에서 **socket options** 매개 변수를 제거합니다.

14장. 가상 머신 성능 최적화

VM(가상 시스템)은 항상 호스트와 비교하여 일정 수준의 성능 저하를 경험합니다. 다음 섹션에서는 이러한 문제가 발생하는 이유를 설명하고, 하드웨어 인프라 리소스를 최대한 효율적으로 사용할 수 있도록 RHEL 8의 가상화 성능 영향을 최소화하는 방법에 대한 지침을 제공합니다.

14.1. 가상 머신 성능에 어떤 영향을 주는가

VM은 호스트에서 사용자 공간 프로세스로 실행됩니다. 따라서 하이퍼바이저는 VM에서 사용할 수 있도록 호스트의 시스템 리소스를 변환해야 합니다. 결과적으로 변환에서 리소스의 일부를 소비하므로 VM은 호스트와 동일한 성능 효율성을 달성할 수 없습니다.

시스템 성능에 가상화가 미치는 영향

VM 성능 손실의 구체적인 이유는 다음과 같습니다.

- 가상 CPU(vCPU)는 Linux 스케줄러에서 처리하는 호스트의 스레드로 구현됩니다.
- VM은 호스트 커널에서 NUMA 또는 대규모 페이지와 같은 최적화 기능을 자동으로 상속하지 않습니다.
- 호스트의 디스크 및 네트워크 I/O 설정은 VM에 상당한 성능 영향을 줄 수 있습니다.
- 네트워크 트래픽은 일반적으로 소프트웨어 기반 브릿지를 통해 VM으로 이동합니다.
- 호스트 장치 및 해당 모델에 따라 특정 하드웨어의 에뮬레이션으로 인해 상당한 오버헤드가 발생할 수 있습니다.

VM 성능에 대한 가상화 영향의 심각도는 다음과 같은 다양한 요인의 영향을 받습니다.

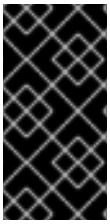
- 동시에 실행 중인 VM 수입니다.
- 각 VM에서 사용하는 가상 장치의 양입니다.

- VM에서 사용하는 장치 유형입니다.

VM 성능 손실 감소

RHEL 8에서는 가상화의 부정적인 성능 효과를 줄이는 데 사용할 수 있는 여러 가지 기능을 제공합니다. 특히:

- tuned 서비스를 사용하면 VM의 리소스 배포 및 성능을 자동으로 최적화할 수 있습니다.
- 블록 I/O 튜닝을 사용하면 디스크와 같은 VM 블록 장치의 성능을 향상시킬 수 있습니다.
- NUMA 튜닝 은 vCPU 성능을 향상시킬 수 있습니다.
- 가상 네트워킹 을 다양한 방식으로 최적화할 수 있습니다.



중요

VM 성능 튜닝은 다른 가상화 기능에 부정적인 영향을 미칠 수 있습니다. 예를 들어 수정된 VM을 마이그레이션하기가 더 어려워질 수 있습니다.

14.2. TUNED를 사용하여 가상 머신 성능 최적화

tuned 유틸리티는 CPU 집약적인 작업 또는 스토리지 네트워크 처리량 응답에 대한 요구 사항과 같은 특정 워크로드 특성에 맞게 RHEL을 조정하는 튜닝 프로필 제공 메커니즘입니다. 다수의 특정 사용 사례에서 성능을 향상시키고 전력 소비를 줄이기 위해 사전 구성된 여러 튜닝 프로필을 제공합니다. 이러한 프로필을 편집하거나 새 프로필을 생성하여 가상화 환경을 포함한 환경에 맞는 성능 솔루션을 만들 수 있습니다.

가상화를 위해 RHEL 8을 최적화하려면 다음 프로필을 사용하십시오.

- RHEL 8 가상 시스템의 경우 virtual-guest 프로필을 사용합니다. 일반적으로 적용 가능한 *throughput-performance* 프로필을 기반으로 하지만 가상 메모리의 스왑성도 줄어듭니다.
- RHEL 8 가상화 호스트의 경우 virtual-host 프로필을 사용합니다. 이렇게 하면 더티 메모리 페이지의 보다 적극적인 쓰기가 가능하여 호스트 성능이 향상됩니다.

사전 요구 사항

- tuned 서비스가 설치 및 활성화되었습니다.

절차

특정 tuned 프로필을 활성화하려면 다음을 수행합니다.

1. 사용 가능한 tuned 프로필을 나열합니다.

```
# tuned-adm list

Available profiles:
- balanced             - General non-specialized tuned profile
- desktop              - Optimize for the desktop use-case
[...]
- virtual-guest        - Optimize for running inside a virtual guest
- virtual-host         - Optimize for running KVM guests
Current active profile: balanced
```

2. 선택 사항: 새 tuned 프로필을 생성하거나 기존 tuned 프로필을 편집합니다.

   자세한 내용은 tuned 프로필 사용자 지정 을 참조하십시오.

3. tuned 프로필을 활성화합니다.

```
# tuned-adm profile selected-profile
```

   - 가상화 호스트를 최적화하려면 *virtual-host* 프로필을 사용합니다.

```
# tuned-adm profile virtual-host
```

   - RHEL 게스트 운영 체제에서 *virtual-guest* 프로필을 사용합니다.

```
# tuned-adm profile virtual-guest
```

추가 리소스

- 시스템 상태 및 성능 모니터링 및 관리

14.3. 가상 머신 메모리 구성

VM(가상 시스템)의 성능을 개선하기 위해 VM에 추가 호스트 RAM을 할당할 수 있습니다. 마찬가지로 호스트 메모리를 다른 VM 또는 작업에 할당할 수 있도록 VM에 할당된 메모리 양을 줄일 수 있습니다.

이러한 작업을 수행하려면 웹 콘솔 또는 명령줄 인터페이스를 사용할 수 있습니다.

14.3.1. 웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거

VM(가상 머신)의 성능을 개선하거나 사용 중인 호스트 리소스를 확보하려면 웹 콘솔을 사용하여 VM에 할당된 메모리 양을 조정할 수 있습니다.

사전 요구 사항

- 게스트 OS는 메모리 balloon 드라이버를 실행하고 있습니다. 이러한 경우인지 확인하려면 다음을 수행합니다.

  1. VM 구성에 memballoon 장치가 포함되어 있는지 확인합니다.

     ```
     # virsh dumpxml testguest | grep memballoon
     <memballoon model='virtio'>
       </memballoon>
     ```

     이 명령은 출력을 표시하며 모델을 none 으로 설정하지 않으면 memballoon 장치가 있습니다.

  2. guest OS에서 balloon 드라이버가 실행 중인지 확인합니다.

     - Windows 게스트에서 드라이버는 virtio-win 드라이버 패키지의 일부로 설치됩니다. 자세한 내용은 Windows 가상 머신용 반가상화 KVM 드라이버 설치를 참조하십시오.

     - Linux 게스트에서는 일반적으로 드라이버가 기본적으로 포함되며 memballoon 장치가 있는 경우 활성화됩니다.

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 선택 사항: 최대 메모리와 현재 VM에 사용된 메모리에 대한 정보를 가져옵니다. 그러면 변경 사항과 확인을 위한 기준선으로 사용할 수 있습니다.

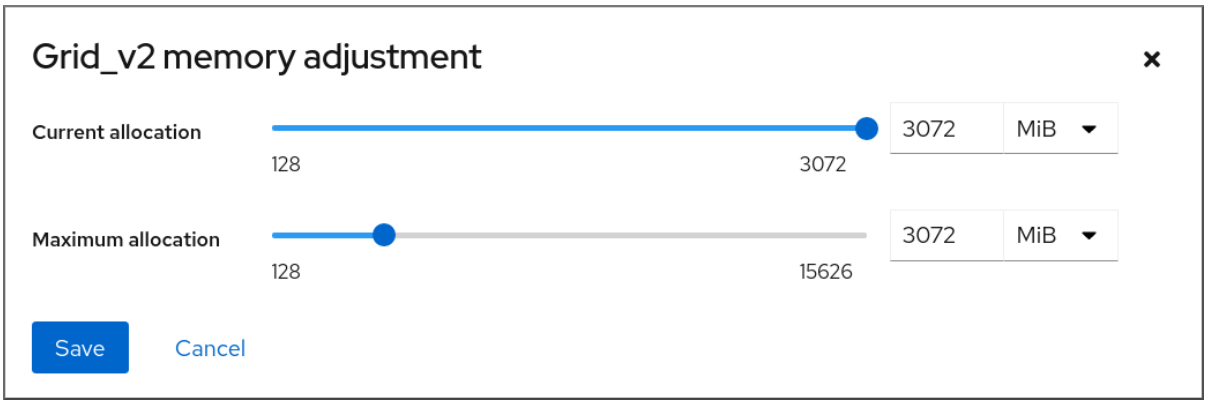
```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.

VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

3. Overview(개요) 창에서 Memory(메모리) 행 옆에 있는 Edit(편집)를 클릭합니다.

Memory Adjustment(메모리 조정) 대화 상자가 표시됩니다.



4. 선택한 VM에 대해 가상 CPU를 구성합니다.

- max allocation (최대 할당) - VM이 프로세스에 사용할 수 있는 최대 호스트 메모리 양을 설정합니다. VM을 생성할 때 최대 메모리를 지정하거나 나중에 늘릴 수 있습니다. 메모리를 MiB 또는 GiB의 배수로 지정할 수 있습니다.

  메모리 할당 최대 조정은 종료 VM에서만 가능합니다.

-

current allocation (현재 할당) - VM에 할당된 실제 메모리 양을 설정합니다. 이 값은 최대 할당보다 작을 수 있지만 초과할 수는 없습니다. 해당 프로세스에 대해 VM에서 사용할 수 있는 메모리를 제어하도록 값을 조정할 수 있습니다. 메모리를 MiB 또는 GiB의 배수로 지정할 수 있습니다.

이 값을 지정하지 않으면 기본 할당은 최대 할당 값입니다.

5. 저장을 클릭합니다.

VM의 메모리 할당이 조정됩니다.

추가 리소스

- 명령줄 인터페이스를 사용하여 가상 머신 메모리 추가 및 제거
- 가상 머신 CPU 성능 최적화

14.3.2. 명령줄 인터페이스를 사용하여 가상 머신 메모리 추가 및 제거

VM(가상 머신)의 성능을 개선하거나 사용 중인 호스트 리소스를 확보하려면 CLI를 사용하여 VM에 할당된 메모리 양을 조정할 수 있습니다.

사전 요구 사항

- 게스트 OS는 메모리 balloon 드라이버를 실행하고 있습니다. 이러한 경우인지 확인하려면 다음을 수행합니다.

  1. VM 구성에 memballoon 장치가 포함되어 있는지 확인합니다.

  ```
  # virsh dumpxml testguest | grep memballoon
  <memballoon model='virtio'>
    </memballoon>
  ```

  이 명령은 출력을 표시하며 모델을 none 으로 설정하지 않으면 memballoon 장치가 있습니다.

2. **balloon** 드라이버가 게스트 **OS**에서 실행 중인지 확인합니다.

   - **Windows** 게스트에서 드라이버는 **virtio-win** 드라이버 패키지의 일부로 설치됩니다. 자세한 내용은 **Windows** 가상 머신용 반가상화 **KVM** 드라이버 설치를 참조하십시오.

   - **Linux** 게스트에서는 일반적으로 드라이버가 기본적으로 포함되며 **memballoon** 장치가 있는 경우 활성화됩니다.

절차

1. 선택 사항: 최대 메모리와 현재 VM에 사용된 메모리에 대한 정보를 가져옵니다. 그러면 변경 사항과 확인을 위한 기준선으로 사용할 수 있습니다.

```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. VM에 할당된 최대 메모리를 조정합니다. 이 값을 늘리면 VM의 성능 잠재력이 향상되고 해당 값을 줄이면 VM이 호스트에 있는 성능 공간을 줄일 수 있습니다. 이러한 변경 사항은 종료 VM에서만 수행할 수 있으므로 실행 중인 VM을 조정하려면 재부팅을 적용해야 합니다.

   예를 들어 *testguest* VM에서 사용할 수 있는 최대 메모리를 4096MiB로 변경하려면 다음을 수행합니다.

```
# virt-xml testguest --edit --memory memory=4096,currentMemory=4096
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

   실행 중인 VM의 최대 메모리 수를 늘리려면 메모리 장치를 VM에 연결할 수 있습니다. 이를 메모리 핫 플러그 라고도 합니다. 자세한 내용은 가상 머신에 장치 연결을 참조하십시오.

주의

실행 중인 VM(메모리 핫 플러그라고도 함)에서 메모리 장치를 제거하는 것은 지원되지 않으며 Red Hat의 디스크가 높습니다.

3. 선택 사항: VM에서 현재 사용 중인 메모리를 최대 할당까지 조정할 수도 있습니다. 이렇게 하면 최대 VM 할당을 변경하지 않고 VM이 호스트에 있는 메모리 부하를 다음 번 재부팅할 때까지 조정합니다.

```
# virsh setmem testguest --current 2048
```

검증

1. VM에서 사용하는 메모리가 업데이트되었는지 확인합니다.

```
# virsh dominfo testguest
Max memory:     4194304 KiB
Used memory:    2097152 KiB
```

2. 선택 사항: 현재 VM 메모리를 조정한 경우 VM의 메모리 증대 통계를 가져와 메모리 사용을 효과적으로 규제하는 방법을 평가할 수 있습니다.

```
 # virsh domstats --balloon testguest
Domain: 'testguest'
  balloon.current=365624
  balloon.maximum=4194304
  balloon.swap_in=0
  balloon.swap_out=0
  balloon.major_fault=306
  balloon.minor_fault=156117
  balloon.unused=3834448
  balloon.available=4035008
  balloon.usable=3746340
  balloon.last-update=1587971682
  balloon.disk_caches=75444
  balloon.hugetlb_pgalloc=0
  balloon.hugetlb_pgfail=0
  balloon.rss=1005456
```

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거
- 가상 머신 CPU 성능 최적화

14.3.3. 추가 리소스

- 가상 머신에 장치 연결.

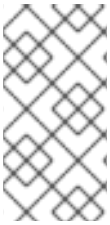
14.4. 가상 머신 I/O 성능 최적화

VM(가상 시스템)의 입력 및 출력(I/O) 기능은 VM의 전반적인 효율성을 크게 제한할 수 있습니다. 이 문제를 해결하기 위해 블록 I/O 매개변수를 구성하여 VM의 I/O를 최적화할 수 있습니다.

14.4.1. 가상 머신의 블록 I/O 튜닝

하나 이상의 VM에서 여러 블록 장치를 사용하는 경우 I/O *가중치를 수정하여 특정 가상 장치의 I/O 우선 순위를 조정하는 것이 중요할* 수 있습니다.

장치의 I/O 가중치를 늘리면 I/O 대역폭의 우선 순위가 증가하므로 더 많은 호스트 리소스를 제공합니다. 마찬가지로 장치의 가중치를 줄이면 호스트 리소스를 더 적게 소비할 수 있습니다.



참고

각 장치의 가중치 값은 1 00~1000 범위 내에 있어야 합니다. 또는 장치별 목록에서 해당 장치를 제거하는 값이 0 일 수 있습니다.

절차

VM의 블록 I/O 매개변수를 표시하고 설정하려면 다음을 수행합니다.

1. VM의 현재 <blkio> 매개변수를 표시합니다.

```
# virsh dumpxml VM-name
```

```
<domain>
 [...]
 <blkiotune>
   <weight>800</weight>
   <device>
     <path>/dev/sda</path>
     <weight>1000</weight>
   </device>
   <device>
     <path>/dev/sdb</path>
     <weight>500</weight>
```

```
  </device>
 </blkiotune>
 [...]
</domain>
```

2. 지정된 장치의 I/O 가중치를 편집합니다.

```
# virsh blkiotune VM-name --device-weights device, I/O-weight
```

예를 들어 다음에서는 riserul *VM* 의 */dev/sda* 장치의 가중치를 500으로 변경합니다.

```
# virsh blkiotune liftbrul --device-weights /dev/sda, 500
```

14.4.2. 가상 머신의 디스크 I/O 제한

여러 개의 VM이 동시에 실행되는 경우 과도한 디스크 I/O를 사용하여 시스템 성능을 방해할 수 있습니다. KVM 가상화의 디스크 I/O 제한은 VM에서 호스트 시스템으로 전송된 디스크 I/O 요청에 제한을 설정하는 기능을 제공합니다. 이렇게 하면 VM이 공유 리소스를 과도하게 활용하고 다른 VM의 성능에 영향을 미치지 않도록 할 수 있습니다.

디스크 I/O 제한을 활성화하려면 VM에 연결된 각 블록 장치에서 호스트 시스템으로 전송된 디스크 I/O 요청에 대한 제한을 설정합니다.

절차

1. virsh domblklist 명령을 사용하여 지정된 VM의 모든 디스크 장치 이름을 나열합니다.

```
# virsh domblklist rollin-coal
Target     Source
------------------------------------------------
vda        /var/lib/libvirt/images/rollin-coal.qcow2
sda        -
sdb        /home/horridly-demanding-processes.iso
```

2. 제한할 가상 디스크가 마운트된 호스트 블록 장치를 찾습니다.

예를 들어 이전 단계에서 sdb 가상 디스크를 제한하려면 다음 출력에서 디스크가 /dev/nvme0n1p3 파티션에 마운트되어 있음을 보여줍니다.

```
$ lsblk
NAME                                MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
```

```
zram0                                   252:0    0     4G  0 disk  [SWAP]
nvme0n1                                   259:0    0 238.5G  0 disk
├─nvme0n1p1                                 259:1    0   600M  0 part  /boot/efi
├─nvme0n1p2                                 259:2    0     1G  0 part  /boot
└─nvme0n1p3                                 259:3    0 236.9G  0 part
  └─luks-a1123911-6f37-463c-b4eb-fxzy1ac12fea 253:0    0 236.9G  0 crypt /home
```

3. **virsh blkiotune** 명령을 사용하여 블록 장치의 I/O 제한을 설정합니다.

```
# virsh blkiotune VM-name --parameter device,limit
```

다음 예제에서는 **rollin-coal** VM의 **sdb** 디스크를 초당 읽기 및 쓰기 I/O 작업 수가 **1000**개, 초당 **50MB**의 읽기 및 쓰기 처리량으로 제한합니다.

```
# virsh blkiotune rollin-coal --device-read-iops-sec /dev/nvme0n1p3,1000 --device-write-iops-sec /dev/nvme0n1p3,1000 --device-write-bytes-sec /dev/nvme0n1p3,52428800 --device-read-bytes-sec /dev/nvme0n1p3,52428800
```

추가 정보

- 디스크 I/O 제한은 서로 다른 고객에게 속한 VM이 동일한 호스트에서 실행 중이거나 다른 VM에 대해 서비스 품질이 제공되는 경우와 같이 다양한 상황에서 유용할 수 있습니다. 디스크 I/O 제한도 느린 디스크를 시뮬레이션하는 데 사용할 수 있습니다.
- I/O 제한은 VM에 연결된 각 블록 장치에 독립적으로 적용할 수 있으며 처리량 및 I/O 작업에 대한 제한을 지원합니다.
- Red Hat은 **virsh blkdeviotune** 명령을 사용하여 VM에서 I/O 제한 구성을 지원하지 않습니다. RHEL 8을 VM 호스트로 사용할 때 지원되지 않는 기능에 대한 자세한 내용은 RHEL 8 가상화의 지원되지 않는 기능을 참조하십시오.

14.4.3. 다중 대기열 virtio-scsi 활성화

VM(가상 시스템)에서 **virtio-scsi** 스토리지 장치를 사용할 때 *다중 대기열 virtio-scsi* 기능을 사용하면 스토리지 성능과 확장성이 향상됩니다. 이를 통해 각 가상 CPU(vCPU)에 다른 vCPU에 영향을 주지 않고 사용할 별도의 대기열과 인터럽트가 있을 수 있습니다.

절차

- 특정 VM에 대한 다중 대기열 **virtio-scsi** 지원을 활성화하려면 VM의 XML 구성에 다음을 추가합니다. 여기서 *N*은 총 vCPU 대기열 수입니다.

```
<controller type='scsi' index='0' model='virtio-scsi'>
  <driver queues='N' />
</controller>
```

14.5. 가상 머신 CPU 성능 최적화

호스트 시스템의 물리적 CPU와 마찬가지로 vCPU는 VM(가상 시스템) 성능에 매우 중요합니다. 결과적으로 vCPU를 최적화하면 VM의 리소스 효율성에 큰 영향을 미칠 수 있습니다. vCPU를 최적화하려면 다음을 수행합니다.

1. VM에 할당된 호스트 CPU 수를 조정합니다. CLI 또는 웹 콘솔을 사용하여 이 작업을 수행할 수 있습니다.

2. vCPU 모델이 호스트의 CPU 모델과 일치하는지 확인합니다. 예를 들어 호스트의 CPU 모델을 사용하도록 *testguest1* VM을 설정하려면 다음을 수행합니다.

```
# virt-xml testguest1 --edit --cpu host-model
```

3. KSM(커널 동일 페이지 병합)을 비활성화합니다.

4. 호스트 시스템에서 NUMA(Non-Uniform Memory Access)를 사용하는 경우 해당 VM에 NUMA를 구성할 수도 있습니다. 이를 통해 호스트의 CPU 및 메모리 프로세스를 가능한 VM의 CPU 및 메모리 프로세스에 긴밀하게 매핑합니다. 실제로 NUMA 튜닝은 VM에 할당된 시스템 메모리에 대한 보다 능률적인 액세스를 제공하므로 vCPU 처리 효율성이 향상될 수 있습니다.

   자세한 내용은 가상 머신에서 NUMA 구성 및 샘플 vCPU 성능 튜닝 시나리오를 참조하십시오.

14.5.1. 명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거

VM(가상 머신)의 CPU 성능을 늘리거나 최적화하려면 VM에 할당된 가상 CPU(vCPU)를 추가하거나 제거할 수 있습니다.

실행 중인 VM에서 수행되는 경우 vCPU 핫 플러그 및 핫 플러그 해제라고도 합니다. 그러나 vCPU 핫 언플러그는 RHEL 8에서 지원되지 않으며 Red Hat은 사용하지 않습니다.

사전 요구 사항

- 선택 사항: 대상 VM에서 vCPU의 현재 상태를 확인합니다. 예를 들어 *testguest* VM의 vCPU 수를 표시하려면 다음을 수행합니다.

```
# virsh vcpucount testguest
maximum      config         4
maximum      live           2
current      config         2
current      live           1
```

이 출력은 *testguest* 가 현재 1 vCPU를 사용 중이며, VM의 성능을 높이기 위해 1개의 vCPu를 핫 플러그로 연결할 수 있음을 나타냅니다. 그러나 재부팅 후 *testguest* 에서 사용하는 vCPU 수는 2개로 변경되며 2개의 vCPU를 핫 플러그로 연결할 수 있습니다.

절차

1. VM에 연결할 수 있는 최대 vCPU 수를 조정하여 다음 부팅 시 적용됩니다.

   예를 들어 *testguest* VM의 최대 vCPU 수를 8로 늘리려면 다음을 수행합니다.

```
# virsh setvcpus testguest 8 --maximum --config
```

   CPU 토폴로지, 호스트 하드웨어, 하이퍼바이저 및 기타 요인에 의해 최대값이 제한될 수 있습니다.

2. VM에 연결된 현재 vCPU 수를 이전 단계에서 구성한 최대값까지 조정합니다. 예를 들면 다음과 같습니다.

   - 실행 중인 *testguest* VM에 연결된 vCPU 수를 4로 늘리려면 다음을 수행합니다.

```
# virsh setvcpus testguest 4 --live
```

     이로 인해 VM의 다음 부팅까지 VM의 성능 및 호스트 부하 공간이 *testguest* 의 호스트 부하가 증가합니다.

   - *testguest* VM에 연결된 vCPU 수를 영구적으로 1로 줄입니다.

```
# virsh setvcpus testguest 1 --config
```

이를 통해 VM의 다음 부팅 후 *testguest* 의 VM 성능 및 호스트 부하 공간이 줄어듭니다. 그러나 필요한 경우 VM에 추가 vCPU를 핫플러그하여 일시적으로 성능을 향상시킬 수 있습니다.

검증

- VM의 현재 vCPU 상태에 변경 사항이 반영되는지 확인합니다.

```
# virsh vcpucount testguest
maximum      config         8
maximum      live           4
current      config         1
current      live           4
```

추가 리소스

- 웹 콘솔을 사용하여 가상 CPU 관리

14.5.2. 웹 콘솔을 사용하여 가상 CPU 관리

RHEL 8 웹 콘솔을 사용하면 웹 콘솔이 연결된 VM(가상 머신)에서 사용하는 가상 CPU를 검토하고 구성할 수 있습니다.

사전 요구 사항

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

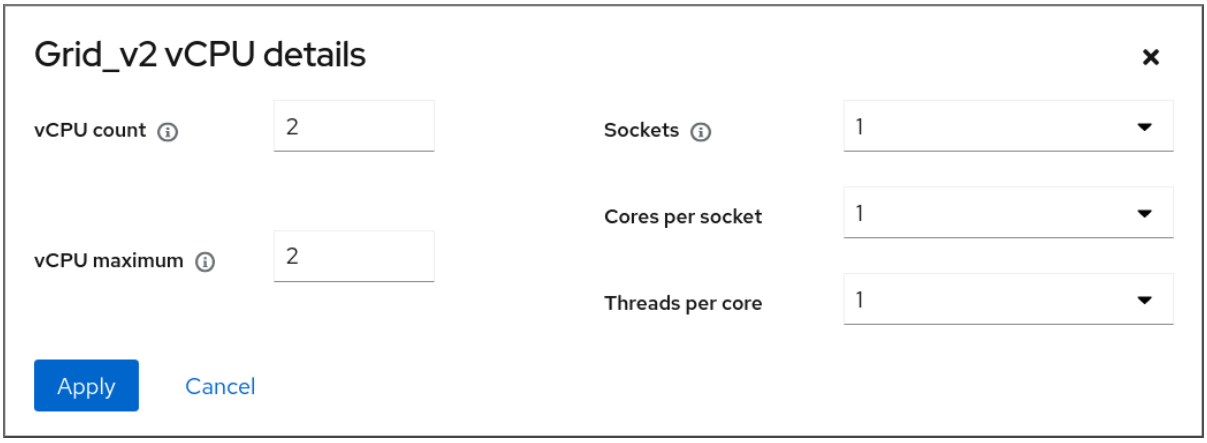
절차

1. Virtual Machines (가상 시스템) 인터페이스에서 표시할 정보가 있는 VM을 클릭합니다.

   VM의 그래픽 인터페이스에 액세스하기 위한 선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 Overview(개요) 섹션이 포함된 새 페이지가 열립니다.

2. Overview(개요) 창에서 vCPU의 수 옆에 있는 Edit(편집)를 클릭합니다.

   vCPU 세부 정보 대화 상자가 나타납니다.



1. 선택한 VM에 대해 가상 CPU를 구성합니다.

- vCPU 개수 - 현재 사용 중인 vCPU 수입니다.

참고

vCPU 수는 vCPU 최대보다 클 수 없습니다.

- vCPU 최대 - VM에 대해 구성할 수 있는 최대 가상 CPU 수입니다. 이 값이 vCPU 개수보다 크면 VM에 추가 vCPU를 연결할 수 있습니다.

- sockets - VM에 표시할 소켓 수입니다.

- Cores per socket (소켓당 코어 수) - 각 소켓이 VM에 노출될 코어 수입니다.

- 코어당 스레드 - 각 코어가 VM에 노출될 스레드 수입니다.

소켓,소켓당 코어, 코어 옵션당 스레드 는 VM의 CPU 토폴로지를 조정합니다. 이는 vCPU 성능에 도움이 될 수 있으며 게스트 OS의 특정 소프트웨어의 기능에 영향을 줄 수 있습니다. 배포에서 다른 설정이 필요하지 않은 경우 기본값을 유지합니다.

2. Apply(적용)를 클릭합니다.

VM의 가상 CPU가 구성됩니다.

참고

가상 CPU 설정 변경 사항은 VM을 다시 시작한 후에만 적용됩니다.

추가 리소스

- 명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거

14.5.3. 가상 머신에서 NUMA 구성

다음 방법을 사용하여 RHEL 8 호스트에서 가상 머신(VM)의 NUMA(Non-Uniform Memory Access) 설정을 구성할 수 있습니다.

사전 요구 사항

- 호스트는 NUMA 호환 시스템입니다. 이 경우 virsh nodeinfo 명령을 사용하고 NUMA 셀 행을 참조하십시오.

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              48
CPU frequency:       1200 MHz
CPU socket(s):       1
Core(s) per socket:  12
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         67012964 KiB
```

행의 값이 2 이상이면 호스트는 NUMA와 호환됩니다.

절차

사용하기 쉽도록 자동화된 유틸리티 및 서비스를 사용하여 VM의 NUMA 구성을 설정할 수 있습니다. 그러나 수동 NUMA 설정으로 성능이 크게 향상될 가능성이 높습니다.

자동 방법

- VM의 NUMA 정책을 Preferred 로 설정합니다. 예를 들어 *testguest5* VM에서 이를 수행하려면 다음을 수행합니다.

-

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- 호스트에서 자동 **NUMA** 분산을 활성화합니다.

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- **numad** 명령을 사용하여 **VM CPU**를 메모리 리소스와 자동으로 정렬합니다.

```
# numad
```

수동 방법

1. 특정 **vCPU** 스레드를 특정 호스트 **CPU** 또는 **CPU** 범위에 고정합니다. 이는 **NUMA**가 아닌 호스트와 **VM**에서도 사용할 수 있으며 **vCPU** 성능을 안전하게 개선하는 방법으로 권장됩니다.

   예를 들어 다음 명령은 각각 **CPU 1, 3, 5, 7, 9, 11**을 호스팅하도록 ***testguest6*** **VM**의 **0~5 vCPU** 스레드를 고정합니다.

```
# virsh vcpupin testguest6 0 1
# virsh vcpupin testguest6 1 3
# virsh vcpupin testguest6 2 5
# virsh vcpupin testguest6 3 7
# virsh vcpupin testguest6 4 9
# virsh vcpupin testguest6 5 11
```

   이후에 이 작업이 성공했는지 확인할 수 있습니다.

```
# virsh vcpupin testguest6
VCPU   CPU Affinity
----------------------
0      1
1      3
2      5
3      7
4      9
5      11
```

2. **vCPU** 스레드를 고정한 후 지정된 **VM**과 연결된 **QEMU** 프로세스 스레드를 특정 호스트 **CPU** 또는 **CPU** 범위에 고정할 수도 있습니다. 예를 들어 다음 명령은 ***testguest6*** 의 **QEMU** 프로세스 스레드를 **CPU 13** 및 **15**에 고정하고 성공했는지 확인합니다.

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
----------------------------------
       *: 13,15
```

3. 마지막으로 특정 VM에 특히 할당될 호스트 NUMA 노드를 지정할 수도 있습니다. 그러면 VM vCPU에 의해 호스트 메모리 사용량이 향상될 수 있습니다. 예를 들어 다음 명령은 호스트 NUMA 노드 3을 5로 사용하도록 *testguest6* 를 설정하고 성공했는지 확인합니다.

```
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```

참고

최상의 성능 결과를 얻으려면 위에 나열된 수동 튜닝 방법을 모두 사용하는 것이 좋습니다.

확인된 문제

- 현재 IBM Z 호스트에서는 NUMA 튜닝을 수행할 수 없습니다.

추가 리소스

- 샘플 vCPU 성능 튜닝 시나리오
- numastat 유틸리티를 사용하여 시스템의 현재 NUMA 구성 보기

14.5.4. 샘플 vCPU 성능 튜닝 시나리오

최상의 vCPU 성능을 얻으려면 다음 시나리오와 같이 manual vcpupin,에뮬레이터pin 및 numatune 설정을 함께 사용하는 것이 좋습니다.

시나리오 시작

- 호스트에는 다음과 같은 하드웨어 세부 사항이 있습니다.
  - NUMA 노드 2개

- 각 노드의 **CPU** 코어 **3**개
- 각 코어에 두 개의 스레드

이러한 머신의 **virsh nodeinfo** 출력은 다음과 유사합니다.

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              12
CPU frequency:       3661 MHz
CPU socket(s):       2
Core(s) per socket:  3
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         31248692 KiB
```

- 기존 **VM**을 **8**개의 **vCPU**로 변경하려고 하므로 단일 **NUMA** 노드에 맞지 않습니다.

따라서 각 **NUMA** 노드에 **4**개의 **vCPU**를 배포하고 **vCPU** 토폴로지가 호스트 토폴로지와 최대한 긴밀하게 일치하도록 해야 합니다. 즉, 지정된 물리적 **CPU**의 스레드로 실행되는 **vCPU**는 동일한 코어의 호스트 스레드에 고정되어야 합니다. 자세한 내용은 다음 *솔루션*을 참조하십시오.

해결책

1. 호스트 토폴로지에 대한 정보를 가져옵니다.

```
# virsh capabilities
```

출력에는 다음과 유사한 섹션이 포함되어야 합니다.

```
<topology>
  <cells num="2">
    <cell id="0">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="10" />
        <sibling id="1" value="21" />
      </distances>
      <cpus num="6">
```

```
      <cpu id="0" socket_id="0" core_id="0" siblings="0,3" />
      <cpu id="1" socket_id="0" core_id="1" siblings="1,4" />
      <cpu id="2" socket_id="0" core_id="2" siblings="2,5" />
      <cpu id="3" socket_id="0" core_id="0" siblings="0,3" />
      <cpu id="4" socket_id="0" core_id="1" siblings="1,4" />
      <cpu id="5" socket_id="0" core_id="2" siblings="2,5" />
    </cpus>
   </cell>
   <cell id="1">
    <memory unit="KiB">15624346</memory>
    <pages unit="KiB" size="4">3906086</pages>
    <pages unit="KiB" size="2048">0</pages>
    <pages unit="KiB" size="1048576">0</pages>
    <distances>
     <sibling id="0" value="21" />
     <sibling id="1" value="10" />
    </distances>
    <cpus num="6">
     <cpu id="6" socket_id="1" core_id="3" siblings="6,9" />
     <cpu id="7" socket_id="1" core_id="4" siblings="7,10" />
     <cpu id="8" socket_id="1" core_id="5" siblings="8,11" />
     <cpu id="9" socket_id="1" core_id="3" siblings="6,9" />
     <cpu id="10" socket_id="1" core_id="4" siblings="7,10" />
     <cpu id="11" socket_id="1" core_id="5" siblings="8,11" />
    </cpus>
   </cell>
  </cells>
</topology>
```

2. **선택 사항: 적용 가능한 툴 및 유틸리티를 사용하여 VM의 성능을 테스트합니다.**

3. **호스트에 1GiB 대규모 페이지를 설정하고 마운트합니다.**

   a. **호스트의 커널 명령줄에 다음 행을 추가합니다.**

   ```
   default_hugepagesz=1G hugepagesz=1G
   ```

   b. **다음 콘텐츠를 사용하여 /etc/systemd/system/hugetlb-gigantic-pages.service 파일을 만듭니다.**

   ```
   [Unit]
   Description=HugeTLB Gigantic Pages Reservation
   DefaultDependencies=no
   Before=dev-hugepages.mount
   ConditionPathExists=/sys/devices/system/node
   ConditionKernelCommandLine=hugepagesz=1G

   [Service]
   ```

```
Type=oneshot
RemainAfterExit=yes
ExecStart=/etc/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

c. 다음 콘텐츠를 사용하여 **/etc/systemd/hugetlb-reserve-pages.sh** 파일을 만듭니다.

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
 echo "ERROR: $nodes_path does not exist"
 exit 1
fi

reserve_pages()
{
 echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages 4 node1
reserve_pages 4 node2
```

그러면 *node1에서 4GiB의 대규모 페이지와 node 2* 에서 4GiB의 대규모 페이지가 예약됩니다.

d. 이전 단계에서 생성한 스크립트를 실행 파일로 만듭니다.

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e. 부팅 시 대규모 페이지 예약을 활성화합니다.

```
# systemctl enable hugetlb-gigantic-pages
```

4. **virsh edit** 명령을 사용하여 최적화하려는 VM의 XML 구성을 편집합니다. 이 예제 ***super-VM***:

```
# virsh edit super-vm
```

5. 다음과 같은 방식으로 VM의 XML 구성을 조정합니다.

a.

8개의 정적 vCPU를 사용하도록 VM을 설정합니다. 이 작업을 수행하려면 <vcpu/> 요소를 사용합니다.

b.

각 vCPU 스레드를 토폴로지에서 미러링하는 해당 호스트 CPU 스레드에 고정합니다. 이렇게 하려면 < cputune> 섹션에서 <vcpupin/ > 요소를 사용합니다.

위의 virsh 기능 유틸리티에 표시된 대로 호스트 CPU 스레드는 해당 코어에서 순차적으로 정렬되지 않습니다. 또한 vCPU 스레드는 동일한 NUMA 노드에서 사용 가능한 최고 수준의 호스트 코어 세트에 고정되어야 합니다. 테이블 그림은 아래의 샘플 토폴로지 섹션을 참조하십시오.

a 및 b. 단계를 위한 XML 구성은 다음과 유사합니다.

```
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='4'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
</cputune>
```

c.

1GiB 대규모 페이지를 사용하도록 VM을 설정합니다.

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
  </hugepages>
</memoryBacking>
```

d.

호스트의 해당 NUMA 노드의 메모리를 사용하도록 VM의 NUMA 노드를 구성합니다. 이렇게 하려면 < numatune/> 섹션에서 <memnode /> 요소를 사용합니다.

```
<numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

e.

CPU 모드가 host-passthrough 로 설정되어 있고 CPU가 passthrough 모드에서 캐시

를 사용하는지 확인합니다.

```
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
```

검증

1. 결과적으로 VM의 XML 구성에 다음과 유사한 섹션이 포함되어 있는지 확인합니다.

```
[...]
 <memoryBacking>
  <hugepages>
   <page size='1' unit='GiB'/>
  </hugepages>
 </memoryBacking>
 <vcpu placement='static'>8</vcpu>
 <cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='4'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
 </cputune>
 <numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
 </numatune>
 <cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
  <numa>
   <cell id="0" cpus="0-3" memory="2" unit="GiB">
    <distances>
     <sibling id="0" value="10"/>
     <sibling id="1" value="21"/>
    </distances>
   </cell>
   <cell id="1" cpus="4-7" memory="2" unit="GiB">
    <distances>
     <sibling id="0" value="21"/>
     <sibling id="1" value="10"/>
    </distances>
   </cell>
  </numa>
 </cpu>
</domain>
```

2. 선택 사항: 적용 가능한 툴 및 유틸리티를 사용하여 VM의 성능을 테스트하여 VM 최적화의 영향을 평가합니다.

토폴로지 샘플

- 다음 표에서는 고정해야 하는 vCPU와 호스트 CPU 간의 연결을 보여줍니다.

표 14.1. 호스트 토폴로지

CPU 스레드	0	3	1	4	2	5	6	9	7	10	8	11
코어	0		1		2		3		4		5	
소켓	0						1					
NUMA 노드	0						1					

표 14.2. VM 토폴로지

vCPU 스레드	0	1	2	3	4	5	6	7
코어	0		1		2		3	
소켓	0				1			
NUMA 노드	0				1			

표 14.3. 결합된 호스트 및 VM 토폴로지

vCPU 스레드			0	1	2	3			4	5	6	7
호스트 CPU 스레드	0	3	1	4	2	5	6	9	7	10	8	11
코어	0		1		2		3		4		5	
소켓	0						1					
NUMA 노드	0						1					

이 시나리오에는 2개의 NUMA 노드와 8개의 vCPU가 있습니다. 따라서 4개의 vCPU 스레드를 각 노드에 고정해야 합니다.

또한 **Red Hat**은 호스트 시스템 작업을 위해 각 노드에서 적어도 하나의 **CPU** 스레드를 사용할 수 있도록 하는 것이 좋습니다.

이 예에서 각 **NUMA** 노드에는 각각 **2**개의 호스트 **CPU** 스레드가 있는 **3**개의 코어가 있으므로 노드 **0**의 세트는 다음과 같이 변환됩니다.

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

14.5.5. 커널 동일 페이지 병합 비활성화

커널 동일 페이지 병합(**KSM**)은 메모리 밀도를 향상하지만 **CPU** 사용률을 높이며 워크로드에 따라 전반적인 성능에 부정적 영향을 미칠 수 있습니다. 이러한 경우 **KSM**을 비활성화하여 **VM**(가상 머신) 성능을 향상시킬 수 있습니다.

요구 사항에 따라 단일 세션에서 **KSM**을 비활성화하거나 영구적으로 설정할 수 있습니다.

절차

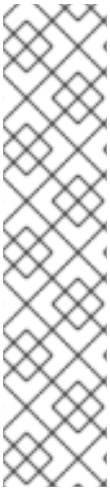
- 단일 세션에서 **KSM**을 비활성화하려면 **systemctl** 유틸리티를 사용하여 **ksm** 및 **ksmtuned** 서비스를 중지합니다.

```
# systemctl stop ksm

# systemctl stop ksmtuned
```

- **KSM**을 영구적으로 비활성화하려면 **systemctl** 유틸리티를 사용하여 **ksm** 및 **ksmtuned** 서비스를 비활성화합니다.

```
# systemctl disable ksm
Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
# systemctl disable ksmtuned
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```



참고

KSM을 비활성화하기 전에 VM 간에 공유되는 메모리 페이지는 공유됩니다. 공유를 중지하려면 다음 명령을 사용하여 시스템의 모든 PageKSM 페이지를 삭제합니다.

```
# echo 2 > /sys/kernel/mm/ksm/run
```

익명 페이지가 KSM 페이지를 교체한 후 khugepaged 커널 서비스는 VM의 물리적 메모리에서 투명한 대규모 페이지를 다시 빌드합니다.

14.6. 가상 머신 네트워크 성능 최적화

VM의 NIC(네트워크 인터페이스 카드)의 가상 특성으로 인해 VM은 할당된 호스트 네트워크 대역폭의 일부를 손실하여 VM의 전체 워크로드 효율성을 줄일 수 있습니다. 다음 팁은 가상 NIC(vNIC) 처리량에 가상화의 부정적인 영향을 최소화할 수 있습니다.

절차

다음 방법을 사용하여 VM 네트워크 성능에 도움이 되는지 확인합니다.

vhost_net 모듈 활성화

호스트에서 vhost_net 커널 기능이 활성화되어 있는지 확인합니다.

```
# lsmod | grep vhost
vhost_net          32768  1
vhost              53248  1 vhost_net
tap                24576  1 vhost_net
tun                57344  6 vhost_net
```

이 명령의 출력이 비어 있으면 vhost_net 커널 모듈을 활성화합니다.

```
# modprobe vhost_net
```

다중 대기열 virtio-net 설정

VM에 대한 *다중 대기열 virtio-net* 기능을 설정하려면 virsh edit 명령을 사용하여 VM의 XML 구성으로 편집합니다. XML에서 <devices> 섹션에 다음을 추가하고 N 을 VM의 vCPU 수로, 최대 16까지 교체합니다.

```
<interface type='network'>
    <source network='default'/>
```

```
    <model type='virtio'/>
    <driver name='vhost' queues='N'/>
</interface>
```

VM이 실행 중인 경우 변경 사항이 적용되도록 VM을 다시 시작하십시오.

네트워크 패킷 배치

긴 전송 경로가 있는 Linux VM 구성에서는 커널에 제출하기 전에 패킷을 일괄 처리하면 캐시 사용률이 향상될 수 있습니다. 패킷 일괄 처리를 설정하려면 호스트에서 다음 명령을 사용하고, VM에서 사용하는 네트워크 인터페이스 이름으로 *tap0* 을 바꿉니다.

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

호스트 NIC에서 SR-IOV를 지원하는 경우 vNIC에 SR-IOV 장치 할당을 사용합니다. 자세한 내용은 SR-IOV 장치 관리를 참조하십시오.

추가 리소스

- 가상 네트워킹 이해

14.7. 가상 머신 성능 모니터링 툴

가장 많은 VM 리소스를 사용하는 항목과 VM 성능에 최적화가 필요한 측면을 식별하기 위해 일반적인 성능 진단 툴과 VM별 툴을 사용할 수 있습니다.

기본 OS 성능 모니터링 툴

표준 성능 평가의 경우 호스트 및 게스트 운영 체제에서 기본적으로 제공하는 유틸리티를 사용할 수 있습니다.

- RHEL 8 호스트에서 root로 top 유틸리티 또는 시스템 모니터 애플리케이션을 사용하고 출력에서 qemu 및 virt 을 찾습니다. 그러면 VM에서 사용하는 호스트 시스템 리소스의 양을 확인할 수 있습니다.
  - 모니터링 도구가 qemu 또는 virt 프로세스에서 호스트 CPU 또는 메모리 용량의 대부분을 사용하는 것을 표시하는 경우 perf 유틸리티를 사용하여 조사합니다. 자세한 내용은 아래를 참조하십시오.

- 또한 *vhost_net-1234* 와 같은 vhost_net 스레드 프로세스가 과도한 호스트 CPU 용량을 소비하는 것으로 표시되는 경우 가상 네트워크 최적화 기능 (예: multi-queue virtio-net) 을 사용하는 것이 좋습니다.

- 게스트 운영 체제에서 시스템에서 사용할 수 있는 성능 유틸리티 및 애플리케이션을 사용하여 가장 많은 시스템 리소스를 사용하는 프로세스를 평가합니다.

  - Linux 시스템에서는 top 유틸리티를 사용할 수 있습니다.

  - Windows 시스템에서 Task Manager 응용 프로그램을 사용할 수 있습니다.

perf kvm

perf 유틸리티를 사용하여 RHEL 8 호스트의 성능에 대한 가상화 관련 통계를 수집하고 분석할 수 있습니다. 이렇게 하려면 다음을 수행합니다.

1. 호스트에서 *perf* 패키지를 설치합니다.

```
# yum install perf
```

2. 가상화 호스트에 대한 perf kvm stat 명령 중 하나를 사용하여 가상화 호스트에 대한 perf 통계를 표시합니다.

   - 하이퍼바이저를 실시간으로 모니터링하려면 perf kvm stat live 명령을 사용합니다.

   - 일정 기간 동안 하이퍼바이저의 perf 데이터를 기록하려면 perf kvm stat record 명령을 사용하여 로깅을 활성화합니다. 명령이 취소되거나 중단되면 데이터는 perf kvm stat report 명령을 사용하여 분석할 수 있는 perf.data.guest 파일에 저장됩니다.

3. VM-EXIT 이벤트 유형 및 해당 배포에 대한 perf 출력을 분석합니다. 예를 들어, PAUSE_INSTRUCTION 이벤트는 자주 일치하지 않지만 다음 출력에서는 호스트 CPU가 실행 중인 vCPU를 잘 처리하지 않는 것으로 제안합니다. 이러한 시나리오에서는 활성 VM 일부 종료, 이러한 VM에서 vCPU 제거 또는 vCPU의 성능을 튜닝 하는 것이 좋습니다.

```
# perf kvm stat report

Analyze events for all VMs, all VCPUs:
```

```
             VM-EXIT    Samples  Samples%     Time%    Min Time    Max Time         Avg time

  EXTERNAL_INTERRUPT     365634    31.59%    18.04%      0.42us  58780.59us    204.08us ( +-   0.99% )
         MSR_WRITE     293428    25.35%     0.13%      0.59us  17873.02us      1.80us ( +-   4.63% )
    PREEMPTION_TIMER     276162    23.86%     0.23%      0.51us  21396.03us      3.38us ( +-   5.19% )
  PAUSE_INSTRUCTION     189375    16.36%    11.75%      0.72us  29655.25us    256.77us ( +-   0.70% )
                HLT      20440     1.77%    69.83%      0.62us  79319.41us  14134.56us ( +-   0.79% )
             VMCALL      12426     1.07%     0.03%      1.02us   5416.25us      8.77us ( +-   7.36% )
      EXCEPTION_NMI         27     0.00%     0.00%      0.69us      1.34us      0.98us ( +-   3.50% )
      EPT_MISCONFIG          5     0.00%     0.00%      5.15us     10.85us      7.88us ( +-  11.67% )

Total Samples:1157497, Total events handled time:413728274.66us.
```

perf kvm stat 의 출력에서 문제를 신호를 보낼 수 있는 기타 이벤트 유형은 다음과 같습니다.

- **INSN_EMULATION** - 차선 VM I/O 구성 을 제안합니다.

perf 를 사용하여 가상화 성능을 모니터링하는 방법에 대한 자세한 내용은 **perf-kvm** 도움말 페이지를 참조하십시오.

numastat

시스템의 현재 **NUMA** 구성을 보려면 **numa ctl** 패키지를 설치하여 제공되는 **numastat** 유틸리티를 사용할 수 있습니다.

다음은 각각 여러 **NUMA** 노드에서 메모리를 가져오는 **4**개의 실행 중인 **VM**이 있는 호스트를 보여줍니다. 이는 **vCPU** 성능에 적합하지 않으며 조정 보증:

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)     68     16    357   6936      2      3    147    598  8128
51747 (qemu-kvm)    245     11      5     18   5172   2532      1     92  8076
53736 (qemu-kvm)     62    432   1661    506   4851    136     22    445  8116
```

```
53773 (qemu-kvm)   1393     3     1     2    12     0     0   6702  8114
--------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total              1769    463   2024   7462  10037   2672    169   7837 32434
```

반면 다음은 단일 노드에서 각 VM에 제공되는 메모리를 보여줍니다. 이 메모리는 훨씬 더 효율적입니다.

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
--------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51747 (qemu-kvm)     0     0     7     0  8072     0     1     0  8080
53736 (qemu-kvm)     0     0     7     0     0     0  8113     0  8120
53773 (qemu-kvm)     0     0     7     0     0     0     1  8110  8118
59065 (qemu-kvm)     0     0  8050     0     0     0     0     0  8051
--------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                0     0  8072     0  8072     0  8114  8110 32368
```

14.8. 추가 리소스

- Windows 가상 머신 최적화

15장. 전원 관리의 중요성

컴퓨터 시스템의 전체 전력 소비를 줄이는 것은 비용 절감에 도움이 됩니다. 각 시스템 구성 요소의 에너지 사용을 효과적으로 최적화하면 시스템에서 수행하는 다양한 작업을 학습하고 해당 작업의 성능이 적합하도록 각 구성 요소를 구성하는 작업이 포함됩니다. 특정 구성 요소 또는 시스템의 전력 소비를 전체적으로 줄임으로써 열과 성능을 낮출 수 있습니다.

적절한 전원 관리 결과는 다음과 같습니다.

- 서버 및 컴퓨팅 센터의 **Heat** 감소
- 냉각, 공간, 케이블, 발전기 및 무정전 전력 공급 장치(**UPS**) 등 **2**차 비용 절감
- 노트북의 배터리 수명 연장
- 탄소 배출량 감소
- 친환경 **IT**와 관련된 정부 규정 또는 법적 요건 충족(예: **Energy Star**)
- 새로운 시스템에 대한 회사 지침 충족

이 섹션에서는 **Red Hat Enterprise Linux** 시스템의 전원 관리에 관한 정보를 설명합니다.

15.1. 전원 관리 기본 사항

효과적인 전원 관리는 다음과 같은 원칙에 따라 구축됩니다:

유휴 CPU는 필요할 때만 작동 중지해야 합니다.

Red Hat Enterprise Linux 6부터 커널은 틱리스 를 실행하므로 이전의 주기적인 타이머 인터럽트가 온디맨드 인터럽트로 교체되었습니다. 따라서 처리를 위해 새 작업이 대기 상태가 될 때까지 유휴 상태로 남아 있고 더 낮은 전원 상태를 입력한 **CPU**는 이러한 상태에 더 오래 유지될 수 있습니다. 그러나 불필요한 타이머 이벤트를 생성하는 애플리케이션이 시스템에 있는 경우 이 기능의 이점을 오프셋

할 수 있습니다. 볼륨 변경 사항 또는 마우스 이동 검사와 같은 폴링 이벤트는 이러한 이벤트의 예입니다.

Red Hat Enterprise Linux에는 CPU 사용량을 기준으로 애플리케이션을 식별하고 감사할 수 있는 툴이 포함되어 있습니다. 자세한 내용은 감사 및 분석 개요 및 감사 도구에서 참조하십시오.

사용하지 않는 하드웨어 및 장치를 완전히 비활성화해야합니다.

이는 이동 부품이 있는 장치(예: 하드 디스크)에 적용됩니다. 이 외에도 일부 애플리케이션은 사용되지 않지만 활성화된 장치를 "열기" 상태로 둘 수 있습니다. 이 경우 커널은 장치가 사용 중이라고 가정하여 장치가 절전 상태로 전환되지 않도록 할 수 있습니다.

낮은 활동은 낮은 wattage로 전환해야 합니다.

그러나 대부분의 경우 이는 최신 하드웨어와 올바른 BIOS 구성 또는 최신 시스템의 UEFI(x86이 아닌 아키텍처 포함)에 따라 달라집니다. 시스템에 최신 공식 펌웨어를 사용하고 있으며 BIOS의 전원 관리 또는 장치 구성 섹션에서 전원 관리 기능이 활성화되어 있는지 확인합니다. 검색할 몇 가지 기능은 다음과 같습니다.

- ARM64를 위한 공동 작업 프로세스 성능 제어(CPPC) 지원
- IBM Power Systems에 대한 PowerNV 지원
- SpeedStep
- 파워트워트!
- Coff''Quiet
- ACPI(C-state)
- 스마트

하드웨어가 이러한 기능을 지원하고 BIOS에서 활성화되어 있는 경우 Red Hat Enterprise Linux는 기본적으로 이러한 기능을 사용합니다.

다양한 형식의 CPU 상태 및 해당 영향

최신 CPU와 고급 구성 및 전원 인터페이스(ACPI)는 서로 다른 전원 상태를 제공합니다. 세 가지 상태는 다음과 같습니다.

- 절전 (C-상태)
- 빈도 및 온도 (P-states)
- Heat 출력(T-states 또는 열 상태)

절전 상태에서 실행되는 CPU는 가장 적은 watt를 사용하지만 필요할 때 해당 상태에서 벗어나는 데 시간이 훨씬 더 걸립니다. 드문 경우지만 이로 인해 CPU가 수면 상태가 될 때마다 즉시 작동해야 할 수 있습니다. 이 상황으로 인해 효과적으로 사용 가능한 CPU가 발생하고 다른 상태가 사용되면 잠재적인 절전이 손실됩니다.

끄기된 시스템은 최소 전력을 사용합니다.

전력 절감을 위한 가장 좋은 방법 중 하나는 시스템을 끄는 것입니다. 예를 들어, 회사에서는 휴식 시간이나 집이 이동하는 경우 시스템을 끄는 지침으로 "신규 IT" 인식에 초점을 맞춘 기업 문화를 개발할 수 있습니다. 또한 Red Hat Enterprise Linux와 함께 제공되는 가상화 기술을 사용하여 여러 물리적 서버를 하나의 더 큰 서버에 통합하고 가상화 기술을 사용하여 가상화할 수도 있습니다.

15.2. 감사 및 분석 개요

단일 시스템의 상세 수동 감사, 분석 및 튜닝은 일반적으로 시간과 비용이 이러한 마지막 시스템 튜닝에서 얻은 이점보다 우선하기 때문입니다.

그러나 거의 동일한 수의 시스템에서 이러한 작업을 한 번 수행하면 모든 시스템에 대해 동일한 설정을 재사용할 수 있는 매우 유용할 수 있습니다. 예를 들어, 수천 대의 데스크탑 시스템 또는 시스템이 거의 동일한 HPC 클러스터를 배포하는 경우를 생각해 보십시오. 감사 및 분석을 수행하는 또 다른 이유는 향후 회귀 또는 시스템 동작 변경을 식별할 수 있는 비교 기반을 제공하기 위한 것입니다. 이 분석 결과는 하드웨어, BIOS 또는 소프트웨어 업데이트가 정기적으로 수행되고 전력 소비와 관련하여 놀라운 일이 발생하지 않는 경우에 매우 유용할 수 있습니다. 일반적으로 철저한 감사 및 분석을 통해 특정 시스템에서 실제로 발생하는 상황을 훨씬 더 잘 이해할 수 있습니다.

사용 가능한 최신 시스템을 사용하더라도 전력 소비와 관련하여 시스템을 감사하고 분석하는 것이 비교적 어렵습니다. 대부분의 시스템은 소프트웨어를 통해 전력 사용을 측정하는 데 필요한 수단을 제공하지 않습니다. 예외가 있습니다.

-

Hewlett Packard 서버 시스템의 ILO 관리 콘솔에는 웹을 통해 액세스할 수 있는 전원 관리 모듈이 있습니다.

- IBM은 BladeCenter 전원 관리 모듈에서 유사한 솔루션을 제공합니다.
- 일부 Dell 시스템에서 IT 도우미는 전원 모니터링 기능도 제공합니다.

다른 벤더는 해당 서버 플랫폼에 대해 유사한 기능을 제공할 수 있지만 모든 벤더가 지원하는 단일 솔루션은 없다는 것을 알 수 있습니다. 전력 소비의 직접 측정은 최대한의 비용 절감 효과를 극대화하기 위해서만 종종 필요합니다.

15.3. 감사를 위한 툴

Red Hat Enterprise Linux 8은 시스템 감사 및 분석을 수행할 수 있는 도구를 제공합니다. 대부분 이미 발견한 내용을 확인하거나 특정 부분에 대한 보다 심층적인 정보가 필요한 경우 추가 정보 소스로 사용할 수 있습니다.

이러한 도구는 대부분 다음과 같은 성능 튜닝에도 사용됩니다.

PowerTOP

CPU를 자주 손상시키는 커널 및 사용자 공간 애플리케이션의 특정 구성 요소를 식별합니다. powertop 명령을 root로 사용하여 PowerTop 툴과 powertop --calibrate 를 시작하여 전력 추정 엔진을 조정합니다. PowerTop에 대한 자세한 내용은 PowerTOP를 사용하여 전력 소비 관리를 참조하십시오.

diskdevstat 및 netdevstat

시스템에서 실행되는 모든 애플리케이션의 디스크 활동 및 네트워크 활동에 대한 자세한 정보를 수집하는 SystemTap 툴입니다. 이러한 툴을 통해 수집된 통계를 사용하여 더 적은 수의 대규모 작업보다 많은 소규모 I/O 작업을 통해 전력을 낭비하는 애플리케이션을 식별할 수 있습니다. yum install tuned-utils-systemtap kernel-debuginfo 명령을 root로 사용하여 diskdevstat 및 netdevstat 도구를 설치합니다.

디스크 및 네트워크 활동에 대한 세부 정보를 보려면 다음을 사용합니다.

```
# diskdevstat

PID  UID  DEV  WRITE_CNT  WRITE_MIN  WRITE_MAX  WRITE_AVG  READ_CNT
READ_MIN  READ_MAX  READ_AVG  COMMAND
```

```
3575  1000  dm-2   59        0.000     0.365       0.006       5        0.000       0.000      0.000
mozStorage #5
3575  1000  dm-2   7         0.000     0.000       0.000       0        0.000       0.000      0.000
localStorage DB
[...]


# netdevstat

PID   UID   DEV     XMIT_CNT   XMIT_MIN   XMIT_MAX   XMIT_AVG   RECV_CNT
RECV_MIN   RECV_MAX   RECV_AVG   COMMAND
3572  991   enp0s31f6   40      0.000      0.882      0.108       0       0.000       0.000      0.000
openvpn
3575  1000 enp0s31f6    27      0.000      1.363      0.160       0       0.000       0.000      0.000
Socket Thread
[...]
```

이러한 명령을 사용하여 **update_interval,total_duration, display_ histogram** 의 세 가지 매개변수를 지정할 수 있습니다.

TuneD

이는 **udev** 장치 관리자를 사용하여 연결된 장치를 모니터링하고 시스템 설정의 정적 및 동적 튜닝을 모두 활성화하는 프로필 기반 시스템 튜닝 도구입니다. **tuned-adm recommend** 명령을 사용하여 특정 제품에 가장 적합한 프로필로 **Red Hat**이 권장하는 프로필을 확인할 수 있습니다. **TuneD**에 대한 자세한 내용은 **TuneD** 로 시작하기 및 **TuneD** 프로필 사용자 지정 을 참조하십시오. **powertop2tuned** 유틸리티 를 사용하면 **PowerTOP** 제안 사항에서 사용자 지정 **TuneD** 프로필을 만들 수 있습니다. **powertop2tuned** 유틸리티에 대한 자세한 내용은 전력 소비 최적화를 참조하십시오.

가상 메모리 통계(**vmstat**)

procps-ng 패키지에서 제공합니다. 이 도구를 사용하면 프로세스, 메모리, 페이징, 블록 **I/O**, 트랩, **CPU** 활동에 대한 세부 정보를 볼 수 있습니다.

이 정보를 보려면 다음을 사용합니다.

```
$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
r  b  swpd  free    buff   cache   si   so  bi   bo   in  cs  us  sy id  wa  st
1  0   0   5805576 380856 4852848   0    0  119  73  814  640  2   2 96   0   0
```

vmstat -a 명령을 사용하여 활성 및 비활성 메모리를 표시할 수 있습니다. 다른 **vmstat** 옵션에 대한 자세한 내용은 **vmstat** 도움말 페이지를 참조하십시오.

iostat

sysstat 패키지에서 제공합니다. 이 도구는 **vmstat** 와 유사하지만 블록 장치에서 **I/O**만 모니터링합니다. 또한 더 자세한 출력 및 통계를 제공합니다.

시스템 I/O를 모니터링하려면 다음을 사용합니다.

```
$ iostat
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          2.05    0.46    1.55    0.26    0.00   95.67

Device     tps     kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
nvme0n1    53.54     899.48     616.99      3445229     2363196
dm-0       42.84     753.72     238.71      2886921      914296
dm-1        0.03       0.60       0.00         2292           0
dm-2       24.15     143.12     379.80       548193     1454712
```

blktrace

I/O 하위 시스템에서 시간이 소비되는 방법에 대한 자세한 정보를 제공합니다.

이 정보를 사람이 읽을 수 있는 형식으로 보려면 다음을 사용합니다.

```
# blktrace -d /dev/dm-0 -o - | blkparse -i -

253,0   1    1   0.000000000  17694  Q   W 76423384 + 8 [kworker/u16:1]
253,0   2    1   0.001926913     0   C   W 76423384 + 8 [0]
[...]
```

여기서 첫 번째 열 253,0 은 장치 주 및 부사입니다. 두 번째 열인 1 은 CPU에 대한 정보를 제공하고 그 뒤에 IO 프로세스를 실행하는 프로세스의 타임스탬프 및 PID에 대한 열을 제공합니다.

여섯 번째 열인 Q 는 이벤트 유형, 쓰기 작업의 경우 7번째 열, 8번째 열, 76423384 를 나타내며 + 8 은 요청된 블록 수입니다.

마지막 필드인 [kworker/u16:1] 은 프로세스 이름입니다.

기본적으로 blktrace 명령은 프로세스가 명시적으로 종료될 때까지 영구적으로 실행됩니다. w 옵션을 사용하여 런타임 기간을 지정합니다.

turbostat

kernel-tools 패키지에서 제공합니다. x86-64 프로세서의 프로세서 토폴로지, 빈도, 유휴 전력 상태 통계, 온도 및 전력 사용량에 대해 보고합니다.

이 요약을 보려면 다음을 사용합니다.

```
# turbostat

CPUID(0): GenuineIntel 0x16 CPUID levels; 0x80000008 xlevels; family:model:stepping
0x6:8e:a (6:142:10)
CPUID(1): SSE3 MONITOR SMX EIST TM2 TSC MSR ACPI-TM HT TM
CPUID(6): APERF, TURBO, DTS, PTM, HWP, HWPnotify, HWPwindow, HWPepp, No-
HWPpkg, EPB
[...]
```

기본적으로 **turbostat** 는 전체 화면에 대한 카운터 결과 요약을 출력하고 **5**초마다 카운터 결과를 표시합니다. **i** 옵션을 사용하여 카운터 결과 간에 다른 기간을 지정합니다. 예를 들어 **turbostat -i 10** 을 실행하여 대신 **10**초마다 결과를 출력합니다.

Turbostat 는 전력 사용 또는 유휴 시간 측면에서 비효율적인 서버를 식별하는 데도 유용합니다. 또한 시스템에서 발생하는 **SMI**(시스템 관리 인터럽트)의 속도를 식별하는 데 도움이 됩니다. 또한 전원 관리 튜닝의 영향을 확인하는 데 사용할 수 있습니다.

cpupower

IT는 프로세서의 전력 절약 기능을 검사하고 조정하는 도구 모음입니다. **cpupower** 명령을 **frequency-info,frequency- set, idle- info,idle- set,set,info, monitor** 옵션과 함께 사용하여 프로세서 관련 값을 표시 및 설정합니다.

예를 들어 사용 가능한 **cpufreq governor**를 보려면 다음을 사용합니다.

```
$ cpupower frequency-info --governors
analyzing CPU 0:
 available cpufreq governors: performance powersave
```

cpupower 에 대한 자세한 내용은 **CPU** 관련 정보 보기를 참조하십시오.

GNOME Power Manager

GNOME 데스크탑 환경의 일부로 설치된 데몬입니다. **GNOME Power Manager**는 시스템 전원 상태의 변경 사항을 알립니다. 예를 들어, 배터리에서 **AC** 전원으로 변경됨. 또한 배터리 상태를 보고하고 배터리 전력이 낮을 때 경고를 표시합니다.

추가 리소스

- **PowerTOP(1), disk devstat(8), netdevstat(8), tuned(8), vmstat(8), iostat(1), bl ktrace(8),**

blkparse(8) 및 **turbostat(8)** 도움말 페이지

- **cpupower(1), cpu power-set(1), cpu power-info(1), cpu power-idle(1), cpu power-frequency-set(1), cpu power-frequency-info(1)** 및 **cpupower-monitor(1)** 도움말 페이지

16장. POWERTOP를 사용하여 전력 소비 관리

시스템 관리자는 PowerTOP 툴을 사용하여 전력 소비를 분석하고 관리할 수 있습니다.

16.1. POWERTOP의 목적

PowerTOP 는 전력 소비와 관련된 문제를 진단하고 배터리 수명을 연장하는 방법에 대한 제안을 제공하는 프로그램입니다.

PowerTOP 툴은 시스템의 총 전력 사용량을 추정하고 각 프로세스, 장치, 커널 작업자, 타이머 및 인터럽트 핸들러에 대한 개별 전원 사용량을 제공할 수 있습니다. 또한 이 도구는 CPU를 자주 시작하는 커널 및 사용자 공간 애플리케이션의 특정 구성 요소를 식별할 수 있습니다.

Red Hat Enterprise Linux 8은 PowerTOP 버전 2.x를 사용합니다.

16.2. POWERTOP 사용

사전 요구 사항

- PowerTOP 를 사용하려면 powertop 패키지가 시스템에 설치되어 있는지 확인합니다.

```
# yum install powertop
```

16.2.1. PowerTOP 시작

절차

- PowerTOP 를 실행하려면 다음 명령을 사용합니다.

```
# powertop
```

중요

power top 명령을 실행할 때 노트북이 배터리 전원 에서 실행해야 합니다.

16.2.2. PowerTOP 조정

절차

1. 랩탑에서 다음 명령을 실행하여 전원 추정 엔진을 조정할 수 있습니다.

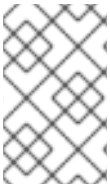
```
# powertop --calibrate
```

2. 프로세스 중에 시스템과 상호 작용하지 않고 위로 마무리합니다.

프로세스가 다양한 테스트를 수행하기 때문에 시간이 걸립니다. 온/오프 레벨 및 스위치 장치를 통해 순환합니다.

3. 일단 프로세스가 완료되면 **PowerTOP** 가 정상적으로 시작됩니다. 약 **1**시간 동안 데이터를 수집하도록 합니다.

충분한 데이터가 수집되면 출력 테이블의 첫 번째 열에 전원 추정 그림이 표시됩니다.



참고

powertop --calibrate 는 랩탑에서만 사용할 수 있습니다.

16.2.3. 측정 간격 설정

기본적으로 **PowerTOP** 는 **20**초 간격으로 측정합니다.

이 측정 빈도를 변경하려면 다음 절차를 사용하십시오.

절차

- **powertop** 명령을 **--time** 옵션으로 실행합니다.

```
# powertop --time=time in seconds
```

16.2.4. 추가 리소스

PowerTOP 사용 방법에 대한 자세한 내용은 **powertop** 도움말 페이지를 참조하십시오.

16.3. POWERTOP 통계

실행되는 동안 **PowerTOP** 는 시스템에서 통계를 수집합니다.

PowerTOP의 출력은 여러 탭을 제공합니다.

- 개요
- 유휴 상태 통계
- 빈도 통계
- 장치 통계
- 튜닝 가능 항목
- **WakeUp**

Tab 및 **Shift+Tab** 키를 사용하여 이러한 탭을 순환할 수 있습니다.

16.3.1. 개요 탭

Overview(개요) 탭에서는 가장 자주 발생하는 **CPU** 또는 가장 많은 전원을 사용하는 구성 요소 목록을 볼 수 있습니다. 프로세스, 인터럽트, 장치 및 기타 리소스를 포함하여 **Overview(개요)** 탭 내의 항목은 해당 사용률에 따라 정렬됩니다.

Overview(개요) 탭 내의 인접한 열에는 다음과 같은 정보가 있습니다.

사용법

리소스 사용 방법을 전원 추정합니다.

이벤트/s

초당 시작 수. 초당 시작 횟수는 커널의 서비스 또는 장치 및 드라이버가 얼마나 효율적으로 수행하는지 나타냅니다. 페일오버가 감소하면 전력 소비가 줄어듭니다. 구성 요소는 전력 사용량을 최적화할 수 있는 정도에 따라 정렬됩니다.

카테고리

구성 요소 분류(예: 프로세스, 장치 또는 타이머).

설명

구성 요소에 대한 설명입니다.

올바르게 측정된 경우 첫 번째 열에 나열된 모든 항목에 대한 전력 소비 추정도 표시됩니다.

그 외에도 **Overview**(개요) 탭에는 다음과 같은 요약 통계가 포함된 줄이 포함되어 있습니다.

- 총 전력 소비
- 나머지 배터리 수명 (해당되는 경우에만)
- 초당 총 가동 시간 요약, 초당 **GPU** 작업, 초당 가상 파일 시스템 작업 요약

16.3.2. Idle 통계 탭

Idle 통계 탭에는 모든 프로세서 및 코어에 대한 **C-**상태가 표시되는 반면, **Frequency** 통계 탭에는 모든 프로세서 및 코어에 **Turbo** 모드를 포함한 **P-state** 사용이 표시됩니다. **C-** 또는 **P-states** 기간은 **CPU** 사용량이 최적화되었는지를 나타냅니다. 높은 **C-** 또는 **P** 상태(예: **C4**가 **C3**보다 높음)에 **CPU**가 유지될수록 **CPU** 사용량 최적화가 좋습니다. 이상적으로, 시스템이 유휴 상태일 때 가장 높은 **C-** 또는 **P** 상태의 **90%** 이상입니다.

16.3.3. 장치 통계 탭

Device stats(장치 통계) 탭은 **Overview**(개요) 탭과 비슷한 정보를 제공하지만 장치의 경우에만 사용됩니다.

16.3.4. 튜닝 가능 항목 탭

Tunables 탭에는 낮은 전력 소비를 위해 시스템을 최적화하기 위한 **PowerTOP**의 제안 사항이 포함되어 있습니다.

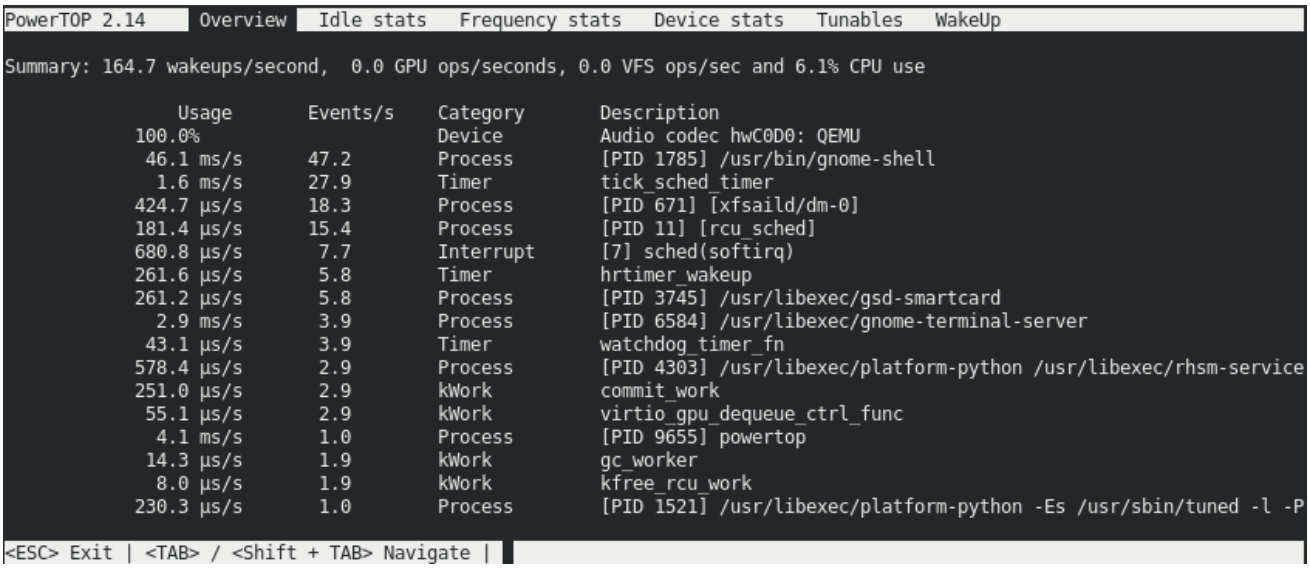
up 및 **down** 키를 사용하여 제안을 통해 이동하고 **Enter** 키를 사용하여 제안을 켜거나 끕니다.

16.3.5. WakeUp 탭

WakeUp 탭에는 사용자가 필요에 따라 변경할 수 있는 장치 시작 설정이 표시됩니다.

위쪽 및 아래쪽 키를 사용하여 사용 가능한 설정을 통해 이동하고 **Enter** 키를 사용하여 설정을 활성화하거나 비활성화합니다.

그림 16.1. PowerTOP 출력



추가 리소스

PowerTOP 에 대한 자세한 내용은 PowerTOP의 홈페이지를 참조하십시오.

16.4. POWERTOP에서 일부 인스턴스에서 FREQUENCY 통계 값을 표시하지 않는 이유

Intel P-State 드라이버를 사용하는 동안 드라이버가 패시브 모드인 경우 **PowerTOP**는 **Frequency Stats** 탭에만 값을 표시합니다. 그러나 이 경우에도 값이 불완전할 수 있습니다.

합계에는 **Intel P-State** 드라이버의 세 가지 가능한 모드가 있습니다.

- HWP(하드웨어 P-States)를 사용하는 활성 모드
- HWP 없는 활성 모드
- 패시브 모드

ACPI CPUfreq 드라이버로 전환하면 전체 정보가 PowerTOP에 표시됩니다. 그러나 시스템을 기본 설정으로 유지하는 것이 좋습니다.

로드되고 어떤 모드에서 드라이버를 보려면 다음을 실행합니다.

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_driver
```

- Intel P-State 드라이버가 로드되고 활성 모드로 전환되면 intel_pstate 가 반환됩니다.
- Intel P-State 드라이버가 로드되고 패시브 모드로 전환되면 intel_cpufreq 가 반환됩니다.
- ACPI CPU freq 드라이버가 로드되면 ACPI-cpu freq가 반환됩니다.

Intel P-State 드라이버를 사용하는 동안 다음 인수를 커널 부트 명령줄에 추가하여 드라이버를 패시브 모드로 강제 실행합니다.

```
intel_pstate=passive
```

Intel P-State 드라이버를 비활성화하고 를 사용하려면 대신 ACPI CPUfreq 드라이버를 사용하려면 커널 부팅 명령줄에 다음 인수를 추가합니다.

```
intel_pstate=disable
```

16.5. HTML 출력 생성

터미널의 powertop 출력 외에 HTML 보고서를 생성할 수도 있습니다.

절차

- **--html** 옵션을 사용하여 **powertop** 명령을 실행합니다.

```
# powertop --html=htmlfile.html
```

htmlfile.html 매개 변수를 출력 파일에 필요한 이름으로 바꿉니다.

16.6. 전력 소비 최적화

전원 소비를 최적화하려면 **power top** 서비스 또는 **powertop 2tuned** 유틸리티를 사용할 수 있습니다.

16.6.1. powertop 서비스를 사용하여 전력 소비 최적화

powertop 서비스를 사용하여 부팅 시 **Tunables** 탭에서 모든 **PowerTOP**의 제안 사항을 자동으로 활성화할 수 있습니다.

절차

- **powertop** 서비스를 활성화합니다.

```
# systemctl enable powertop
```

16.6.2. powertop2tuned 유틸리티

powertop2tuned 유틸리티를 사용하면 **PowerTOP** 제안 사항에서 사용자 지정 **TuneD** 프로필을 만들 수 있습니다.

기본적으로 **powertop2tuned** 는 **/etc/tuned/** 디렉터리에 프로필을 생성하고 현재 선택한 **TuneD** 프로필의 사용자 지정 프로필을 기반으로 합니다. 안전상의 이유로 새 프로필에서 모든 **PowerTOP** 튜닝이 초기에 비활성화됩니다.

튜닝을 활성화하려면 다음을 수행할 수 있습니다.

- **/etc/tuned/profile_name/tuned.conf** 파일에서 주석 처리를 해제합니다.

- **enable** 또는 **- e** 옵션을 사용하여 **PowerTOP** 에서 제안하는 대부분의 튜닝을 활성화하는 새 프로필을 생성합니다.

  USB 자동 일시 중지와 같은 특정 잠재적으로 문제가 있는 튜닝은 기본적으로 비활성화되어 있으며 수동으로 주석 처리를 해제해야 합니다.

16.6.3. powertop2tuned 유틸리티를 사용하여 전력 소비 최적화

사전 요구 사항

- **powertop2tuned** 유틸리티는 시스템에 설치됩니다.

```
# yum install tuned-utils
```

절차

1. 사용자 정의 프로필을 생성합니다.

```
# powertop2tuned new_profile_name
```

2. 새 프로필을 활성화합니다.

```
# tuned-adm profile new_profile_name
```

추가 정보

- **powertop2tuned** 에서 지원하는 전체 옵션 목록은 다음을 사용합니다.

```
$ powertop2tuned --help
```

16.6.4. powertop.service 및 powertop2tuned 비교

다음과 같은 이유로 **powertop2tuned** 를 사용하여 전원 소비를 최적화하는 것이 좋습니다.

- **powertop2tuned** 유틸리티는 **PowerTOP** 를 **TuneD** 에 통합하여 두 도구의 이점을 활용할 수 있습니다.

- **powertop2tuned** 유틸리티를 사용하면 활성화된 튜닝을 세부적으로 제어할 수 있습니다.
- **powertop2tuned** 를 사용하면 잠재적으로 위험한 튜닝이 자동으로 활성화되지 않습니다.
- **powertop2tuned** 를 사용하면 재부팅하지 않으면 롤백이 가능합니다.

17장. 에너지 사용량을 최적화하기 위한 CPU 빈도 튜닝

이 섹션에서는 필요한 **CPUfreq governor**를 설정한 후 요구 사항에 따라 시스템의 **CPU** 속도를 설정하기 위해 사용 가능한 **cpupower** 명령을 사용하여 시스템의 전력 소비를 최적화하는 방법을 설명합니다.

17.1. 지원되는 CPUPOWER 툴 명령

cpupower 툴은 프로세서의 전력 절약 관련 기능을 검사하고 조정하는 툴 모음입니다.

cpupower 툴은 다음 명령을 지원합니다.

idle-info

cpupower idle-info 명령을 사용하여 **CPU** 유휴 드라이버에 사용 가능한 유휴 상태 및 기타 통계를 표시합니다. 자세한 내용은 **CPU Idle State**를 참조하십시오.

idle-set

cpupower idle-set 명령을 **root**로 사용하여 특정 **CPU** 유휴 상태를 활성화하거나 비활성화합니다. **d** 를 사용하여 특정 **CPU** 유휴 상태를 비활성화하려면 **-e** 를 사용합니다.

frequency-info

cpu power frequency-info 명령을 사용하여 현재 **cpufreq** 드라이버와 사용 가능한 **cpu freq governor**를 표시합니다. 자세한 내용은 **CPUfreq** 드라이버,**코어 CPUfreq Governors** 및 **Intel P-state CPUfreq governors**를 참조하십시오.

frequency-set

cpu power frequency-set 명령을 **root**로 사용하여 **cpu freq** 및 **governor**를 설정합니다. 자세한 내용은 **CPUfreq governor** 설정을 참조하십시오.

set

cpupower set 명령을 **root**로 사용하여 프로세서 절전 정책을 설정합니다.

--perf-bias 옵션을 사용하면 지원되는 **Intel** 프로세서에서 소프트웨어를 활성화하여 최적의 성능과 전력 절약 사이의 균형을 결정할 수 있습니다. 할당된 값은 **0** 에서 **15** 까지입니다. 여기서 **0** 은 최적의 성능이며 **15** 는 최적의 전력 효율성입니다. 기본적으로 **-perf-bias** 옵션은 모든 코어에 적용됩니다. 개별 코어에만 적용하려면 **--cpu cpulist** 옵션을 추가합니다.

info

cpupower set 명령을 사용하여 활성화한 프로세서 전원 관련 및 하드웨어 구성을 표시합니다. 예를 들어 **--perf-bias** 값을 **5** 로 할당하는 경우

```
# cpupower set --perf-bias 5
# cpupower info
analyzing CPU 0:
perf-bias: 5
```

모니터

cpupower monitor 명령을 사용하여 유휴 통계 및 **CPU** 요구 사항을 표시합니다.

```
# cpupower monitor
 | Nehalem       || Mperf    ||Idle_Stats
 CPU| C3   | C6   | PC3  | PC6  || C0   | Cx   | Freq || POLL | C1   | C1E  | C3   | C6   | C7s  | C8   |
C9   | C10
   0|  1.95| 55.12|  0.00|  0.00||  4.21| 95.79|  3875||  0.00|  0.68|  2.07|  3.39| 88.77|  0.00|  0.00|
0.00| 0.00
[...]
```

I 옵션을 사용하면 시스템에서 사용 가능한 모든 모니터와 **-m** 옵션을 사용하여 특정 모니터와 관련된 정보를 표시할 수 있습니다. 예를 들어, **Mperf** 모니터와 관련된 정보를 모니터링하려면 **cpupower monitor -m Mperf** 명령을 **root**로 사용합니다.

추가 리소스

- **cpupower(1), cpu power-idle-info(1), cpu power-idle-set(1), cpu power-frequency-set(1), cpu power-frequency -info (1), cpu power-info(1)** 및 **cpupower-monitor(1)** 도움말 페이지

17.2. CPU ID 상태

x86 아키텍처가 있는 **CPU**는 비활성화되거나 더 낮은 성능 설정**(C-states)**을 사용하는 등 다양한 상태를 지원합니다.

이 상태를 사용하면 사용하지 않는 **CPU**를 부분적으로 비활성화하여 전력을 절약할 수 있습니다. 바람직하지 않은 전력이나 성능 문제를 방지하기 위해 **governor**가 필요한 **P-state**와 잠재적으로 설정된 **P-state**와 달리 **C** 상태를 구성할 필요가 없습니다. **C** 상태는 **C0** 이후부터 번호가 매겨지며 **CPU** 기능 감소와 전력 절약을 나타내는 숫자가 높습니다. 주어진 수의 **C** 상태는 프로세서 간에 광범위하게 유사하지만 상태의 특정 기능 세트의 정확한 세부 사항은 프로세서 제품군마다 다를 수 있습니다. **C-states 0-3**은 다음과 같이 정의됩니다.

C0

이 상태에서는 **CPU**가 작동 중이고 유휴 상태가 아닙니다.

C1, Halt

이 상태에서 프로세서는 명령을 실행하지 않지만 일반적으로 하위 전원 상태에 있지 않습니다. **CPU**는 지연 없이 계속 처리할 수 있습니다. **C-state**를 제공하는 모든 프로세서는 이 상태를 지원해야 합니다. **Pentium 4** 프로세서는 **C1E**라는 향상된 **C1** 상태를 지원하므로, 실제로 전력 소비를 줄이는 상태입니다.

C2, Stop-Clock

이 상태에서는 이 프로세서에 대한 클록이 있지만 레지스터와 캐시의 전체 상태를 유지하므로 시계를 다시 시작한 후 즉시 처리를 시작할 수 있습니다. 이는 선택적 상태입니다.

C3, Sleep

이 상태에서 프로세서는 유휴 상태로 전환되며 캐시를 최신 상태로 유지할 필요가 없습니다. 이러한 이유로 인해 이러한 상태에서 벗어나려면 **C2** 상태에서보다 훨씬 더 많은 시간이 필요합니다. 이는 선택적 상태입니다.

다음 명령을 사용하여 **CPUidle** 드라이버의 사용 가능한 유휴 상태 및 기타 통계를 볼 수 있습니다.

```
$ cpupower idle-info
CPUidle governor: menu
analyzing CPU 0:

Number of idle states: 9
Available idle states: POLL C1 C1E C3 C6 C7s C8 C9 C10
[...]
```

"Nehalem" 마이크로 아키텍처가 있는 **Intel CPU**에는 **C6** 상태가 있어 **CPU**의 공급을 **0**으로 줄일 수 있지만 일반적으로 전력 소비를 **80%**에서 **90%**까지 줄입니다. **Red Hat Enterprise Linux 8**의 커널에는 이 새로운 **C** 상태에 대한 최적화가 포함되어 있습니다.

추가 리소스

- **cpupower(1)** 및 **cpupower-idle(1)** 도움말 페이지

17.3. CPUFREQ 개요

시스템의 전력 소비 및 열 출력을 줄이는 가장 효과적인 방법 중 하나는 **Red Hat Enterprise Linux 8**의 **x86** 및 **ARM64** 아키텍처에서 지원하는 **CPUfreq**입니다. **CPU** 속도 확장이라고도 하는 **CPUfreq**는 전력을 절약하기 위해 **CPU** 빈도를 확장할 수 있는 **Linux** 커널의 인프라입니다.

CPU 확장은 고급 구성 및 전원 인터페이스(ACPI) 이벤트에 응답하여 시스템 부하에 따라 자동으로 수행하거나 사용자 공간 프로그램에 의해 수동으로 수행할 수 있으며, 프로세서의 클록 속도를 즉각적으로 조정할 수 있습니다. 이렇게 하면 시스템이 짧은 클럭 속도로 실행되어 전력을 절약할 수 있습니다. 더 빠르거나 느린 클록 속도, 빈도를 변경할 때 등 빈도를 변경하는 규칙은 CPUfreq governor에 의해 정의됩니다.

cpu power frequency-info 명령을 root로 사용하여 cpu freq 정보를 볼 수 있습니다.

17.3.1. CPUfreq 드라이버

cpupower frequency-info --driver 명령을 root로 사용하여 현재 CPUfreq 드라이버를 볼 수 있습니다.

다음은 CPUfreq에 사용할 수 있는 두 가지 드라이버입니다.

ACPI CPUfreq

ACPI(Advanced Configuration and Power Interface) CPUfreq 드라이버는 ACPI를 통해 특정 CPU의 빈도를 제어하는 커널 드라이버로, 커널과 하드웨어 간의 통신을 보장합니다.

Intel P-state

Red Hat Enterprise Linux 8에서는 Intel P-state 드라이버가 지원됩니다. 드라이버는 Intel Xeon E 시리즈 아키텍처 또는 최신 아키텍처를 기반으로 프로세서에서 P-상태 선택을 제어하기 위한 인터페이스를 제공합니다.

현재 지원되는 CPU에 기본적으로 Intel P-state가 사용됩니다. intel_pstate=disable 명령을 커널 명령줄에 추가하여 ACPI CPUfreq를 사용하도록 전환할 수 있습니다.

Intel P-state는 setpolicy() 콜백을 구현합니다. 드라이버는 cpufreq 코어에서 요청된 정책에 따라 사용할 P-state를 결정합니다. 프로세서가 내부적으로 다음 P-상태를 선택할 수 있는 경우 드라이버는 이 책임을 프로세서에 오프로드합니다. 그렇지 않은 경우 드라이버는 알고리즘을 구현하여 다음 P-상태를 선택합니다.

Intel P-state는 자체 sysfs 파일을 제공하여 P 상태 선택을 제어합니다. 이러한 파일은 /sys/devices/system/cpu/intel_pstate/ 디렉터리에 있습니다. 파일의 변경 사항은 모든 CPU에 적용할 수 있습니다.

이 디렉터리에는 P-state 매개변수 설정에 사용되는 다음 파일이 포함되어 있습니다.

- **max_perf_pct** 는 드라이버가 사용 가능한 성능의 백분율로 표현한 최대 **P-state**를 제한합니다. 사용 가능한 **P-state** 성능은 **no_turbo** 설정으로 줄일 수 있습니다.

- **min_perf_pct** 는 드라이버에서 요청한 최소 **P-**상태를 제한하며, 이는 최대 **no-turbo** 성능 수준의 백분율로 표시됩니다.

- **no_turbo** 는 **turbo** 빈도 범위 아래에 있는 **P-state**를 선택하도록 드라이버를 제한합니다.

- **turbo_pct** 는 **turbo** 범위에 있는 하드웨어에서 지원하는 총 성능의 백분율을 표시합니다. 이 숫자는 **turbo**가 비활성화 되었는지 여부와 독립적입니다.

- **num_p states**는 하드웨어에서 지원하는 **P-state** 수를 표시합니다. 이 숫자는 **turbo**가 비활성화되었는지 여부와 독립적입니다.

추가 리소스

- **cpupower-frequency-info(1)** 도움말 페이지

17.3.2. 코어 CPUfreq governor

CPUfreq governor는 시스템 **CPU**의 전원 특성을 정의하며, 이 특성은 **CPU** 성능에 영향을 미칩니다. 각 **governor**는 워크로드 측면에서 고유한 동작, 목적 및 적합성을 갖습니다. **cpupower frequency-info --governor** 명령을 **root**로 사용하여 사용 가능한 **CPUfreq governor**를 볼 수 있습니다.

Red Hat Enterprise Linux 8에는 여러 코어 **CPUfreq governor**가 포함되어 있습니다.

cpufreq_performance

CPU가 가능한 가장 높은 클록 빈도를 사용하도록 강제 적용합니다. 이 빈도는 정적으로 설정되며 변경되지 않습니다. 따라서 이 특정 **governor**는 전력 절약 효과를 제공하지 않습니다. 워크로드가 많은 시간에만 적합하며 **CPU**가 거의 또는 유휴 상태가 되지 않는 경우에만 적합합니다.

cpufreq_powersave

CPU가 가능한 가장 낮은 클록 빈도를 사용하도록 강제 적용합니다. 이 빈도는 정적으로 설정되며 변경되지 않습니다. 이 **governor**는 최대 전력 절감을 제공하지만 **CPU** 성능이 가장 낮습니다. 기본적으로 전체 부하의 느린 **CPU**가 로드되지 않은 빠른 **CPU**보다 더 많은 전력을 소비하므로 "전원 저

장"이라는 용어는 비활성화되는 경우가 있습니다. 따라서 예상 낮은 활동이 낮은 시간대에 절전 조정기를 사용하도록 CPU를 설정하는 것이 좋지만, 그 기간 동안 예기치 않은 높은 부하로 인해 시스템이 실제로 더 많은 전력을 소비할 수 있습니다. 절전 governor는 CPU의 속도 제한기보다 절전기입니다. 과열이 문제가 될 수 있는 시스템과 환경에서 가장 유용합니다.

cpufreq_ondemand

CPU가 시스템 로드가 높은 경우 최대 클록 빈도와 시스템이 유휴 상태일 때 최소 클록 빈도를 달성할 수 있는 동적 governor입니다. 따라서 시스템은 시스템 부하와 관련하여 전력 소비를 적절하게 조정할 수 있지만 빈도 전환 사이의 대기 시간이 소요되었습니다. 따라서 시스템이 유휴 워크로드와 작업량이 너무 자주 전환되는 경우 ondemand governor가 제공하는 모든 성능 또는 절전 이점을 오프셋할 수 있습니다. 대부분의 시스템에서 온 디맨드 governor는 난방 제거, 전력 소비, 성능 및 관리 효율성 사이에서 최상의 절충안을 제공할 수 있습니다. 시스템이 특정 시간에만 사용 중일 때 ondemand governor는 추가 개입 없이 부하에 따라 최대 빈도와 최소 빈도 간에 자동으로 전환합니다.

cpufreq_userspace

이를 통해 사용자 공간 프로그램 또는 root로 실행되는 모든 프로세스가 빈도를 설정할 수 있습니다. 모든 governor에서 사용자 공간은 가장 사용자 지정 가능한 공간 이며 구성 방법에 따라 시스템의 성능과 소비 간에 최적의 균형을 유지할 수 있습니다.

cpufreq_conservative

온 디맨드 governor와 마찬가지로, 보수적인 governor도 사용량에 따라 클럭 빈도를 조정합니다. 그러나 보수적인 governor는 빈도 사이에서 점점 더 점진적으로 전환됩니다. 즉, 보수적인 governor는 최대 및 최소 중에서 선택하는 대신 부하에 가장 적합한 클럭 빈도에 맞게 조정됩니다. 이는 전력 소비를 대폭 절감할 수 있지만, 온 디맨드 governor보다 더 많은 대기 시간을 확보할 수 있습니다.

참고

cron 작업을 사용하여 governor를 활성화할 수 있습니다. 이를 통해 하루 중 특정 시간에 특정 governor를 자동으로 설정할 수 있습니다. 따라서 유휴 시간(예: 근무 시간 후) 동안 낮은 빈도 governor를 지정하고 워크로드가 많은 시간 동안 더 높은 빈도의 governor로 돌아갈 수 있습니다.

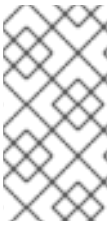
특정 governor를 활성화하는 방법에 대한 지침은 CPUfreq governor 설정을 참조하십시오.

17.3.3. Intel P-state CPUfreq governors

기본적으로 Intel P-state 드라이버는 CPU가 HWP를 지원하는지 여부에 따라 HWP(하드웨어 p-state)를 사용하거나 사용하지 않고 활성 모드로 작동합니다.

cpupower frequency-info --governor 명령을 root로 사용하여 사용 가능한 CPUfreq governor를 볼

수 있습니다.



참고

성능 및 절전 **Intel P-state CPUfreq governors** 기능은 동일한 이름의 코어 **CPUfreq governor**와 다릅니다.

Intel P-state 드라이버는 다음 세 가지 모드에서 작동할 수 있습니다.

하드웨어 관리 **P-states**를 사용하는 활성 모드

HWP를 사용한 활성 모드를 사용하는 경우 **Intel P-state** 드라이버는 **CPU**에 **P** 상태 선택을 수행하도록 지시합니다. 드라이버는 빈도 힌트를 제공할 수 있습니다. 그러나 최종 선택은 **CPU** 내부 논리에 따라 다릅니다. **HWP**를 사용한 활성 모드에서 **Intel P-state** 드라이버는 다음 두 가지 **P** 상태 선택 알고리즘을 제공합니다.

- **performance: performance governor**를 사용하면 이 드라이버는 내부 **CPU** 로직을 성능 지향적으로 지시합니다. 허용되는 **P-**상태의 범위는 드라이버에서 사용할 수 있는 범위의 위쪽 경계로 제한됩니다.

- **powersave: powersave governor**를 사용하면 이 드라이버는 내부 **CPU** 논리를 절전 지향으로 지시합니다.

하드웨어 관리 **P-state**가 없는 활성 모드

HWP가 없는 활성 모드를 사용하는 경우 **Intel P-state** 드라이버는 다음 두 가지 **P** 상태 선택 알고리즘을 제공합니다.

- **performance: performance governor**를 사용하면 드라이버에서 사용할 수 있는 최대 **P-state**를 선택합니다.

- **powersave: powersave governor**를 사용하면 드라이버에서 현재 **CPU** 사용률에 비례하여 **P-**상태를 선택합니다. 이 동작은 온디맨드 **CPUfreq core governor**와 유사합니다.

패시브 모드

passive 모드를 사용하면 **Intel P-state** 드라이버가 기존 **CPUfreq** 확장 드라이버와 동일하게 작동합니다. 사용 가능한 모든 일반 **CPUFreq** 코어 **governor**를 사용할 수 있습니다.

17.3.4. CPUfreq governor 설정

모든 **CPUfreq** 드라이버는 **kernel-tools** 패키지의 일부로 구축되며 자동으로 선택됩니다. **CPUfreq**를 설정하려면 **governor**를 선택해야 합니다.

사전 요구 사항

- **cpupower** 를 사용하려면 **kernel-tools** 패키지를 설치합니다.

```
# yum install kernel-tools
```

절차

1. 특정 **CPU**에 사용할 수 있는 **governor**를 확인합니다.

```
# cpupower frequency-info --governors
analyzing CPU 0:
 available cpufreq governors: performance powersave
```

2. 모든 **CPU**에서 **governor** 중 하나를 활성화합니다.

```
# cpupower frequency-set --governor performance
```

성능 **governor**를 요구 사항에 따라 **cpufreq governor** 이름으로 바꿉니다.

특정 코어에서만 **governor**를 활성화하려면 범위 또는 쉼표로 구분된 **CPU** 번호와 함께 **-c** 를 사용합니다. 예를 들어 **CPU 1-3** 및 **5**의 **userspace governor**를 활성화하려면 다음을 사용합니다.

```
# cpupower -c 1-3,5 frequency-set --governor cpufreq_userspace
```

참고

kernel-tools 패키지가 설치되지 않은 경우 **CPUfreq** 설정은 **/sys/devices/system/cpu/cpuid/cpufreq/** 디렉토리에서 볼 수 있습니다. 이러한 튜닝 가능 항목에 작성하여 설정 및 값을 변경할 수 있습니다. 예를 들어 **cpu0**의 최소 클록 속도를 **360 MHz**로 설정하려면 다음을 사용합니다.

```
# echo 360000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

검증

- **governor**가 활성화되었는지 확인합니다.

```
# cpupower frequency-info
analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency:  Cannot determine or is not supported.
  hardware limits: 400 MHz - 4.20 GHz
  available cpufreq governors: performance powersave
  current policy: frequency should be within 400 MHz and 4.20 GHz.
      The governor "performance" may decide which speed to use within this range.
  current CPU frequency: Unable to call hardware
  current CPU frequency: 3.88 GHz (asserted by call to kernel)
  boost state support:
    Supported: yes
    Active: yes
```

현재 정책에는 최근 활성화된 **cpufreq governor**가 표시됩니다. 이 경우 *성능이* 우수합니다.

추가 리소스

- **cpupower-frequency-info(1)** 및 **cpupower-frequency-set(1)** 도움말 페이지

18장. PERF 시작하기

시스템 관리자는 perf 툴을 사용하여 시스템의 성능 데이터를 수집 및 분석할 수 있습니다.

18.1. PERF 소개

PCL(커널 기반 하위 시스템 *성능 카운터)과 perf 사용자 공간 툴 인터페이스입니다. perf* 는 *PMU(* 성능 모니터링 장치)를 사용하여 다양한 하드웨어 및 소프트웨어 이벤트를 측정, 기록 및 모니터링하는 강력한 도구입니다. perf 는 추적 지점, kprobes 및 uprobes도 지원합니다.

18.2. PERF 설치

이 절차에서는 perf 사용자 공간 툴을 설치합니다.

절차

- perf 툴을 설치합니다.

```
# yum install perf
```

18.3. 일반적인 PERF 명령

이 섹션에서는 일반적으로 사용되는 perf 명령에 대한 개요를 제공합니다.

일반적으로 사용되는 perf 명령

perf stat

이 명령은 실행된 명령과 사용된 클럭 사이클을 포함하여 일반적인 성능 이벤트에 대한 전반적인 통계를 제공합니다. 옵션을 사용하면 기본 측정 이벤트 이외의 이벤트를 선택할 수 있습니다.

perf 레코드

이 명령은 성능 데이터를 파일에 기록합니다 . perf.data 는 나중에 perf report 명령을 사용하여 분석할 수 있습니다.

perf 보고서

이 명령은 perf 레코드로 만든 perf.data 파일에서 성능 데이터를 읽고 표시합니다.

perf list

이 명령은 특정 시스템에서 사용할 수 있는 이벤트를 나열합니다. 이러한 이벤트는 시스템의 성능 모니터링 하드웨어 및 소프트웨어 구성에 따라 달라집니다.

perf top

이 명령은 **top** 유틸리티와 유사한 기능을 수행합니다. 성능 카운터 프로필을 실시간으로 생성하고 표시합니다.

perf 추적

이 명령은 **strace** 툴과 유사한 기능을 수행합니다. 지정된 스레드 또는 프로세스와 해당 애플리케이션에서 수신한 모든 신호에서 사용하는 시스템 호출을 모니터링합니다.

perf 도움말

이 명령은 전체 권한 명령 목록을 표시합니다.

추가 리소스

- 하위 명령에 **--help** 옵션을 추가하여 도움말 페이지를 엽니다.

19장. PERF TOP을 사용하여 실시간으로 CPU 사용량 프로파일링

perf top 명령을 사용하여 다양한 기능의 CPU 사용량을 실시간으로 측정할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다 .

19.1. PERF TOP의 목적

perf top 명령은 실시간 시스템 프로파일링에 사용되며 **top** 유틸리티와 유사하게 작동합니다. 그러나 **top** 유틸리티에서 일반적으로 지정된 프로세스 또는 스레드가 사용하는 CPU 시간 양을 보여 주는 경우 **perf top** 은 각 특정 함수에서 사용하는 CPU 시간을 보여 줍니다. 기본 상태에서 **perf top** 은 사용자 공간 및 커널 공간의 모든 CPU에서 사용되는 기능에 대해 알려줍니다. **perf top** 을 사용하려면 루트 액세스가 필요합니다.

19.2. PERF TOP을 사용하여 CPU 사용량 프로파일링

이 절차에서는 최상위 및 프로파일 CPU 사용량을 실시간으로 활성화합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- 루트 액세스 권한이 있습니다.

절차

- **perf** 최상위 모니터링 인터페이스를 시작합니다.

```
# perf top
```

모니터링 인터페이스는 다음과 유사합니다.

```
Samples: 8K of event 'cycles', 2000 Hz, Event count (approx.): 4579432780 lost: 0/0 drop:
0/0
Overhead  Shared Object       Symbol
```

```
  2.20%  [kernel]           [k] do_syscall_64
  2.17%  [kernel]           [k] module_get_kallsym
  1.49%  [kernel]           [k] copy_user_enhanced_fast_string
  1.37%  libpthread-2.29.so  [.] pthread_mutex_lock 1.31% [unknown] [.] 0000000000000000
1.07% [kernel] [k] psi_task_change 1.04% [kernel] [k] switch_mm_irqs_off 0.94% [kernel] [k]
fget
  0.74%  [kernel]           [k] entry_SYSCALL_64
  0.69%  [kernel]           [k] syscall_return_via_sysret
  0.69%  libxul.so          [.] 0x000000000113f9b0
  0.67%  [kernel]           [k] kallsyms_expand_symbol.constprop.0
  0.65%  firefox            [.] moz_xmalloc
  0.65%  libpthread-2.29.so  [.] __pthread_mutex_unlock_usercnt
  0.60%  firefox            [.] free
  0.60%  libxul.so          [.] 0x000000000241d1cd
  0.60%  [kernel]           [k] do_sys_poll
  0.58%  [kernel]           [k] menu_select
  0.56%  [kernel]           [k] _raw_spin_lock_irqsave
  0.55%  perf               [.] 0x00000000002ae0f3
```

이 예에서 커널 함수 **do_syscall_64** 는 대부분의 **CPU** 시간을 사용하고 있습니다.

추가 리소스

- **perf-top(1)** 도움말 페이지

19.3. PERF 상위 출력 해석

perf top 모니터링 인터페이스는 여러 열에 데이터를 표시합니다.

"Overhead" 열

는 지정된 함수에서 사용 중인 **CPU**의 백분율을 표시합니다.

"Shared Object" 열

함수를 사용하고 있는 프로그램 또는 라이브러리의 이름을 표시합니다.

"Symbol" 열

함수 이름 또는 기호를 표시합니다. 커널 공간에서 실행되는 함수는 **[k]** 로 식별되며 사용자 공간에서 실행되는 함수는 **[.]** 로 식별됩니다.

19.4. PERF가 일부 기능 이름을 원시 기능 주소로 표시하는 이유

커널 함수의 경우 **perf** 는 **/proc/kallsyms** 파일의 정보를 사용하여 해당 함수 이름 또는 기호에 샘플을

매핑합니다. 그러나 사용자 공간에서 실행되는 함수의 경우 바이너리가 제거되므로 원시 함수 주소가 표시될 수 있습니다.

실행 파일의 **The debuginfo** 패키지를 설치해야 하거나 실행 파일이 로컬에서 개발된 애플리케이션인 경우 이러한 상황에서 기능 이름 또는 기호를 표시하려면 디버깅 정보를 사용하여 애플리케이션을 컴파일해야 합니다.

참고

실행 파일과 연결된 **the debuginfo** 를 설치한 후에는 **perf record** 명령을 다시 실행할 필요가 없습니다. **perf report** 명령을 간단히 다시 실행합니다.

추가 리소스

- 디버깅 정보를 사용하여 디버깅 활성화

19.5. 디버그 및 소스 리포지토리 활성화

Red Hat Enterprise Linux의 표준 설치에서는 디버그 및 소스 리포지토리를 활성화하지 않습니다. 이러한 리포지토리에는 시스템 구성 요소를 디버깅하고 성능을 측정하는 데 필요한 정보가 포함되어 있습니다.

절차

- 소스 및 디버그 정보 패키지 채널을 활성화합니다.

```
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms
```

$(uname -i) 부분은 자동으로 시스템 아키텍처의 일치하는 값으로 교체됩니다.

아키텍처 이름	현재의
64비트 Intel 및 AMD	x86_64
64비트 ARM	aarch64

아키텍처 이름	현재의
IBM POWER	ppc64le
64-bit IBM Z	s390x

19.6. GDB를 사용하여 애플리케이션 또는 라이브러리용 DEBUGINFO 패키지 가져오기

코드를 디버깅하려면 디버깅 정보가 필요합니다. 패키지에서 설치된 코드의 경우 **GDB(GNU Debugger)**는 누락된 디버그 정보를 자동으로 인식하고 패키지 이름을 해석하며 패키지를 가져오는 방법에 대한 구체적인 조언을 제공합니다.

사전 요구 사항

- 디버깅할 애플리케이션 또는 라이브러리가 시스템에 설치되어 있어야 합니다.
- **GDB** 및 **the debuginfo-install** 툴은 시스템에 설치해야 합니다. 자세한 내용은 애플리케이션 디버그 설정을 참조하십시오.
- 시스템에서 채널 **Provide debuginfo** 및 **debugsource** 패키지를 구성하고 활성화해야 합니다.

절차

1. 디버깅할 애플리케이션 또는 라이브러리에 연결된 **GDB**를 시작합니다. **GDB**는 누락된 디버깅 정보를 자동으로 인식하고 실행할 명령을 제안합니다.

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. **GDB**를 종료합니다. **q** 를 입력하고 **Enter** 를 사용하여 확인합니다.

```
(gdb) q
```

3. GDB에서 제안하는 명령을 실행하여 **required debuginfo** 패키지를 설치합니다.

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

dnf 패키지 관리 도구는 변경 사항에 대한 요약을 제공하고 확인 후 필요한 모든 파일을 다운로드 및 설치합니다.

4. GDB가 **debuginfo** 패키지를 제안할 수 없는 경우 애플리케이션 또는 라이브러리의 **debuginfo** 패키지 가져오기에 설명된 절차를 수동으로 수행합니다.

추가 리소스

- **Red Hat Developer Toolset** 사용 설명서, 디버깅 정보 설치 섹션
- **RHEL** 시스템용 **debuginfo** 패키지를 다운로드하거나 설치하는 방법은 무엇입니까? - **Red Hat Knowledgebase** 솔루션

20장. PERF 통계를 사용하여 프로세스 실행 중 이벤트 수

perf stat 명령을 사용하여 프로세스 실행 중에 하드웨어 및 소프트웨어 이벤트를 계산할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다 .

20.1. PERF STAT의 목적

perf stat 명령은 지정된 명령을 실행하고 명령을 실행하는 동안 하드웨어 및 소프트웨어 이벤트 발생의 실행 수를 유지하고 이러한 개수의 통계를 생성합니다. 이벤트를 지정하지 않으면 통계가 공통 하드웨어 및 소프트웨어 이벤트 집합을 계산합니다.

20.2. PERF 통계가 있는 이벤트 수

perf 통계를 사용하여 명령 실행 중에 하드웨어 및 소프트웨어 이벤트 발생 횟수를 계산하고 이러한 개수의 통계를 생성할 수 있습니다. 기본적으로 **perf** 통계는 스레드당 모드로 작동합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다 .

절차

- 이벤트 수를 계산합니다.
  - 루트 액세스 없이 **perf stat** 명령을 실행하면 사용자 공간에서 발생하는 이벤트만 계산됩니다.

```
$ perf stat ls
```

예 20.1. 루트 액세스없이 실행된 perf stat의 출력

```
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

 Performance counter stats for 'ls':

              1.28 msec task-clock:u              #    0.165 CPUs utilized
```

```
          0      context-switches:u        #    0.000 M/sec
          0      cpu-migrations:u          #    0.000 K/sec
        104      page-faults:u             #    0.081 M/sec
  1,054,302      cycles:u                  #    0.823 GHz
  1,136,989      instructions:u            #    1.08  insn per cycle
    228,531      branches:u                #  178.447 M/sec
     11,331      branch-misses:u           #    4.96% of all branches

0.007754312 seconds time elapsed

0.000000000 seconds user
0.007717000 seconds sys
```

이전 예에서 볼 수 있듯이, perf stat 이 root 액세스 없이 실행될 때 이벤트 이름 뒤에는 해당 이벤트가 사용자 공간에서만 계산되었음을 나타냅니다.

- **사용자 공간 및 커널 공간 이벤트를 모두 계산하려면 통계에 따라 root 액세스 권한이 있어야 합니다.**

```
# perf stat ls
```

예 20.2. 루트 액세스 권한으로 실행된 perf stat의 출력

```
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

 Performance counter stats for 'ls':

          3.09 msec task-clock              #    0.119 CPUs utilized
            18      context-switches        #    0.006 M/sec
             3      cpu-migrations          #    0.969 K/sec
           108      page-faults             #    0.035 M/sec
     6,576,004      cycles                  #    2.125 GHz
     5,694,223      instructions            #    0.87  insn per cycle
     1,092,372      branches                #  352.960 M/sec
        31,515      branch-misses           #    2.89% of all branches

      0.026020043 seconds time elapsed

      0.000000000 seconds user
      0.014061000 seconds sys
```

- **기본적으로 perf 통계는 스레드당 모드로 작동합니다. CPU 전체 이벤트 개수로 변경하려면 -a 옵션을 perf stat 에 전달합니다. CPU 전체 이벤트를 계산하려면 루트 액세스가 필요합니다.**

```
# perf stat -a ls
```

추가 리소스

- **perf-stat(1)** 도움말 페이지

20.3. PERF STAT 출력 해석

Perf stat 은 지정된 명령을 실행하고 명령 실행 중에 이벤트 발생 횟수를 계산하며 이러한 개수의 통계를 세 열로 표시합니다.

1. 특정 이벤트에 대해 계산된 발생 수
2. 계산된 이벤트의 이름
3. 관련 지표를 사용할 수 있게 되면 오른쪽 끝에 있는 해시 기호(#) 뒤에 비율 또는 백분율이 표시됩니다.

예를 들어 기본 모드에서 실행하는 경우 **perf stat** 은 사이클과 지침을 모두 계산하므로 오른쪽 끝에 있는 사이클당 명령을 계산하고 표시합니다. 기본적으로 두 이벤트가 모두 계산되므로 분기 누락과 관련하여 유사한 동작을 모든 분기의 백분율로 확인할 수 있습니다.

20.4. 실행 중인 프로세스에 PERF 통계 연결

실행 중인 프로세스에 **perf** 통계를 연결할 수 있습니다. 이렇게 하면 명령을 실행하는 동안 지정된 프로세스에서만 이벤트 발생 수를 계산하도록 **perf stat** 에 지시합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- 실행 중인 프로세스에 **perf** 통계를 연결합니다.

```
$ perf stat -p ID1,ID2 sleep seconds
```

이전 예제에서는 **sleep** 명령을 사용하여 지정한 시간(초) 동안 ***ID1 및 ID2*** 를 사용하여 프로

세스의 이벤트를 계산합니다.

추가 리소스

- **perf-stat(1)** 도움말 페이지

21장. PERF를 사용하여 성능 프로파일 기록 및 분석

perf 툴을 사용하면 성능 데이터를 기록하고 나중에 분석할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다 .

21.1. PERF 레코드의 목적

perf record 명령은 성능 데이터를 샘플링하고 이를 다른 perf 명령으로 읽고 시각화할 수 있는 perf.data 파일에 저장합니다 . perf.data 는 현재 디렉터리에서 생성되며 나중에 다른 시스템에서 액세스할 수 있습니다.

동안 기록할 perf 레코드에 대한 명령을 지정하지 않으면 Ctrl+C 를 눌러 프로세스를 수동으로 중지할 때까지 기록됩니다. -p 옵션 다음에 하나 이상의 프로세스 ID를 전달하여 perf 레코드 를 특정 프로세스에 연결할 수 있습니다. 루트 액세스 없이 perf 레코드 를 실행할 수 있지만 사용자 공간에서만 샘플 성능 데이터만 실행할 수 있습니다. 기본 모드에서 perf 레코드 는 CPU 주기를 샘플링 이벤트로 사용하며 상속 모드가 활성화된 스레드별 모드에서 작동합니다.

21.2. 루트 액세스없이 성능 프로필 기록

샘플에 대한 루트 액세스 권한 없이 perf 레코드 를 사용하고 사용자 공간에만 성능 데이터를 기록할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다 .

절차

- 성능 데이터를 샘플링하고 기록합니다.

```
$ perf record command
```

데이터를 샘플링하려는 *명령으로* 명령을 바꿉니다. 명령을 지정하지 않으면 Ctrl+C 를 눌러 수동으로 중지할 때까지 perf 레코드에서 데이터를 샘플링합니다.

추가 리소스

- **perf-record(1)** 도움말 페이지

21.3. 루트 액세스 권한으로 성능 프로필 기록

샘플에 대한 루트 액세스 권한과 함께 **perf** 레코드 를 사용하고 사용자 공간과 커널 공간에 동시에 성능 데이터를 기록할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다 .
- 루트 액세스 권한이 있습니다.

절차

- 성능 데이터를 샘플링하고 기록합니다.

```
# perf record command
```

데이터를 샘플링하려는 *명령으로* 명령을 바꿉니다. 명령을 지정하지 않으면 **Ctrl+C** 를 눌러 수동으로 중지할 때까지 **perf** 레코드에서 데이터를 샘플링합니다.

추가 리소스

- **perf-record(1)** 도움말 페이지

21.4. CPU당 모드에서 성능 프로필 기록

CPU당 perf 레코드 를 사용하여 모니터링된 **CPU**의 모든 스레드에서 동시에 및 사용자 공간 및 커널 공간에 성능 데이터를 샘플링하고 기록할 수 있습니다. 기본적으로 **CPU**당 모드는 모든 온라인 **CPU**를 모니터링합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다 .

절차

- 성능 데이터를 샘플링하고 기록합니다.

```
# perf record -a command
```

데이터를 샘플링하려는 *명령으로* 명령을 바꿉니다. 명령을 지정하지 않으면 **Ctrl+C** 를 눌러 수동으로 중지할 때까지 **perf** 레코드에서 데이터를 샘플링합니다.

추가 리소스

- **perf-record(1)** 도움말 페이지

21.5. PERF 레코드를 사용하여 호출 그래프 데이터 캡처

성능 프로필의 다른 기능을 호출하는 함수를 기록하도록 **perf** 레코드 툴을 구성할 수 있습니다. 여러 프로세스가 동일한 함수를 호출하는 경우 성능 장애를 식별하는 데 도움이 됩니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- **call-graph** 옵션을 사용하여 성능 데이터를 샘플링 하고 기록합니다.

```
$ perf record --call-graph method command
```

  - 데이터를 샘플링하려는 *명령으로* 명령을 바꿉니다. 명령을 지정하지 않으면 **Ctrl+C** 를 눌러 수동으로 중지할 때까지 **perf** 레코드에서 데이터를 샘플링합니다.

  - *방법을* 다음과 같은 팝업 방법 중 하나로 바꿉니다.

    fp

프레임 포인터 방법을 사용합니다. 컴파일러 최적화(예: GCC 옵션 **--fomit-frame-pointer** 로 빌드된 바이너리)에 따라 스택이 풀링되지 않을 수 있습니다.

dwarf

DWARF 호출 프레임 정보를 사용하여 스택을 제거합니다.

lbr

Intel 프로세서에서 마지막 분기 레코드 하드웨어를 사용합니다.

추가 리소스

- **perf-record(1)** 도움말 페이지

21.6. PERF 보고서를 사용하여 PERF.DATA 분석

perf 보고서를 사용하여 **perf .data** 파일을 표시하고 분석할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- 현재 디렉터리에는 **perf.data** 파일이 있습니다.
- 루트 액세스 권한으로 **perf.data** 파일을 만든 경우 루트 액세스 권한으로 **perf** 보고서 도 실행해야 합니다.

절차

- 추가 분석을 위해 **perf.data** 파일의 내용을 표시합니다.

```
# perf report
```

이 명령은 다음과 유사한 출력을 표시합니다.

```
Samples: 2K of event 'cycles', Event count (approx.): 235462960
```

```
Overhead  Command          Shared Object                     Symbol
   2.36%  kswapd0          [kernel.kallsyms]                 [k] page_vma_mapped_walk
   2.13%  sssd_kcm         libc-2.28.so                      [.] memset_avx2_erms 2.13% perf
[kernel.kallsyms] [k] smp_call_function_single 1.53% gnome-shell libc-2.28.so [.]
strcmp_avx2
   1.17%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_hash_table_lookup
   0.93%  Xorg             libc-2.28.so                      [.] memmove_avx_unaligned_erms 0.89%
gnome-shell libgobject-2.0.so.0.5600.4 [.] g_object_unref 0.87% kswapd0 [kernel.kallsyms]
[k] page_referenced_one 0.86% gnome-shell libc-2.28.so [.] memmove_avx_unaligned_erms
   0.83%  Xorg             [kernel.kallsyms]                 [k] alloc_vmap_area
   0.63%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_slice_alloc
   0.53%  gnome-shell      libgirepository-1.0.so.1.0.0      [.] g_base_info_unref
   0.53%  gnome-shell      ld-2.28.so                        [.] _dl_find_dso_for_object
   0.49%  kswapd0          [kernel.kallsyms]                 [k] vma_interval_tree_iter_next
   0.48%  gnome-shell      libpthread-2.28.so                [.] pthread_getspecific 0.47% gnome-
shell libgirepository-1.0.so.1.0.0 [.] 0x0000000000013b1d 0.45% gnome-shell libglib-
2.0.so.0.5600.4 [.] g_slice_free1 0.45% gnome-shell libgobject-2.0.so.0.5600.4 [.]
g_type_check_instance_is_fundamentally_a 0.44% gnome-shell libc-2.28.so [.] malloc 0.41%
swapper [kernel.kallsyms] [k] apic_timer_interrupt 0.40% gnome-shell ld-2.28.so [.]
_dl_lookup_symbol_x 0.39% kswapd0 [kernel.kallsyms] [k]
raw_callee_save___pv_queued_spin_unlock
```

추가 리소스

- **perf-report(1)** 도움말 페이지

21.7. PERF 보고서 출력 해석

perf report 명령을 실행하여 표시된 테이블에서 데이터를 여러 열로 정렬합니다.

'Overhead' 열

이 특정 기능에서 수집된 전체 샘플의 백분율을 나타냅니다.

'Command' 열

은 샘플이 수집한 프로세스를 알려줍니다.

'Shared Object' 열

는 샘플이 제공된 **ELF** 이미지의 이름을 표시합니다(커널.**kallsyms** 이름이 커널에서 온 경우 이름).

'Symbol' 열

함수 이름 또는 기호를 표시합니다.

기본 모드에서는 함수가 가장 높은 오버헤드가 가장 높은 함수가 먼저 내림차순으로 정렬됩니다.

21.8. 다른 장치에서 읽을 수 있는 PERF.DATA 파일 생성

perf 도구를 사용하여 성능 데이터를 perf .data 파일에 기록하여 다른 장치에서 분석할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다 .
- kernel debuginfo 패키지가 설치되어 있습니다. 자세한 내용은 GDB를 사용하여 애플리케이션 또는 라이브러리의 debuginfo 패키지 가져오기를 참조하십시오.

절차

1. 조사에 관심이 있는 성능 데이터를 캡처합니다.

```
# perf record -a --call-graph fp sleep seconds
```

이 예제에서는 sleep 명령을 사용하여 지정한 시간 동안 *전체* 시스템에 perf.data 를 생성합니다. 또한 프레임 포인터 방법을 사용하여 호출 그래프 데이터를 캡처합니다.

2. 기록된 데이터의 디버그 기호가 포함된 아카이브 파일을 생성합니다.

```
# perf archive
```

검증 단계

- 아카이브 파일이 현재 활성 디렉터리에 생성되었는지 확인합니다.

```
# ls perf.data*
```

출력에는 현재 디렉터리에 있는 perf.data 로 시작하는 모든 파일이 표시됩니다. 아카이브 파일의 이름은 다음과 같습니다.

```
perf.data.tar.gz
```

-

또는

```
perf.data.tar.bz2
```

추가 리소스

- perf를 사용하여 성능 프로파일 기록 및 분석
- perf 레코드를 사용하여 호출 그래프 데이터 캡처

21.9. 다른 장치에서 생성된 PERF.DATA 파일 분석

perf 툴을 사용하여 다른 장치에서 생성된 perf.data 파일을 분석할 수 있습니다.

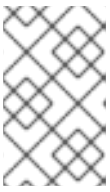
사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.
- 다른 장치에 생성된 perf.data 파일 및 관련 아카이브 파일이 현재 사용 중인 장치에 있습니다.

절차

1. perf.data 파일과 아카이브 파일을 현재 활성 디렉터리에 복사합니다.
2. 아카이브 파일을 ~/.debug:에 추출합니다.

```
# mkdir -p ~/.debug
# tar xf perf.data.tar.bz2 -C ~/.debug
```



참고

아카이브 파일의 이름은 ***perf.data.tar.gz*** 로 지정될 수도 있습니다.

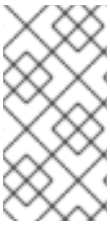
3. 추가 분석을 위해 **perf.data** 파일을 엽니다.

```
# perf report
```

21.10. PERF가 일부 기능 이름을 원시 기능 주소로 표시하는 이유

커널 함수의 경우 **perf** 는 **/proc/kallsyms** 파일의 정보를 사용하여 해당 함수 이름 또는 기호에 샘플을 매핑합니다. 그러나 사용자 공간에서 실행되는 함수의 경우 바이너리가 제거되므로 원시 함수 주소가 표시될 수 있습니다.

실행 파일의 **The debuginfo** 패키지를 설치해야 하거나 실행 파일이 로컬에서 개발된 애플리케이션인 경우 이러한 상황에서 기능 이름 또는 기호를 표시하려면 디버깅 정보를 사용하여 애플리케이션을 컴파일해야 합니다.



참고

실행 파일과 연결된 **the debuginfo** 를 설치한 후에는 **perf record** 명령을 다시 실행할 필요가 없습니다. **perf report** 명령을 간단히 다시 실행합니다.

추가 리소스

- 디버깅 정보를 사용하여 디버깅 활성화

21.11. 디버그 및 소스 리포지토리 활성화

Red Hat Enterprise Linux의 표준 설치에서는 디버그 및 소스 리포지토리를 활성화하지 않습니다. 이러한 리포지토리에는 시스템 구성 요소를 디버깅하고 성능을 측정하는 데 필요한 정보가 포함되어 있습니다.

절차

- 소스 및 디버그 정보 패키지 채널을 활성화합니다.

```
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms
```

$(uname -i) 부분은 자동으로 시스템 아키텍처의 일치하는 값으로 교체됩니다.

아키텍처 이름	현재의
64비트 Intel 및 AMD	x86_64
64비트 ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

21.12. GDB를 사용하여 애플리케이션 또는 라이브러리용 DEBUGINFO 패키지 가져오기

코드를 디버깅하려면 디버깅 정보가 필요합니다. 패키지에서 설치된 코드의 경우 **GDB(GNU Debugger)**는 누락된 디버그 정보를 자동으로 인식하고 패키지 이름을 해석하며 패키지를 가져오는 방법에 대한 구체적인 조언을 제공합니다.

사전 요구 사항

- 디버깅할 애플리케이션 또는 라이브러리가 시스템에 설치되어 있어야 합니다.
- **GDB** 및 **the debuginfo-install** 툴은 시스템에 설치해야 합니다. 자세한 내용은 애플리케이션 디버그 설정을 참조하십시오.
- 시스템에서 채널 **Provide debuginfo** 및 **debugsource** 패키지를 구성하고 활성화해야 합니다.

절차

1. 디버깅할 애플리케이션 또는 라이브러리에 연결된 **GDB**를 시작합니다. **GDB**는 누락된 디버깅 정보를 자동으로 인식하고 실행할 명령을 제안합니다.

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no
debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. **GDB**를 종료합니다. **q** 를 입력하고 **Enter** 를 사용하여 확인합니다.

```
(gdb) q
```

3. **GDB**에서 제안하는 명령을 실행하여 **required debuginfo** 패키지를 설치합니다.

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

dnf 패키지 관리 도구는 변경 사항에 대한 요약을 제공하고 확인 후 필요한 모든 파일을 다운로드 및 설치합니다.

4. **GDB**가 **debuginfo** 패키지를 제안할 수 없는 경우 애플리케이션 또는 라이브러리의 **debuginfo** 패키지 가져오기에 설명된 절차를 수동으로 수행합니다.

추가 리소스

- **Red Hat Developer Toolset** 사용 설명서, 디버깅 정보 설치 섹션
- **RHEL** 시스템용 **debuginfo** 패키지를 다운로드하거나 설치하는 방법은 무엇입니까? **- Red Hat Knowledgebase** 솔루션

22장. PERF를 사용하여 사용 중인 CPU 조사

시스템에서 성능 문제를 조사할 때 perf 툴을 사용하여 가장 많은 CPU를 식별하고 모니터링하여 작업에 집중할 수 있습니다.

22.1. PERF 통계를 사용하여 계산된 CPU 이벤트 표시

CPU 개수 집계를 비활성화하여 perf stat 을 사용하여 에서 계산된 CPU 이벤트를 표시할 수 있습니다. 이 기능을 사용하려면 -a 플래그를 사용하여 시스템 전체 모드에서 이벤트를 계산해야 합니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

- CPU 개수 집계가 비활성화된 이벤트를 계산합니다.

```
# perf stat -a -A sleep seconds
```

이전 예제는 CPU 0 부터 시작하여 각 개별 CPU에 대해 절전 명령을 사용하여 지시한 *대로* 시간 동안 기록된 일반적인 하드웨어 및 소프트웨어 이벤트 집합의 기본 집합을 표시합니다. 따라서 사이클과 같은 이벤트를 지정하는 것이 유용할 수 있습니다.

```
# perf stat -a -A -e cycles sleep seconds
```

22.2. PERF 보고서를 사용하여 수행한 CPU 샘플 표시

perf record 명령은 성능 데이터를 샘플링하고 perf report 명령을 사용하여 읽을 수 있는 perf.data 파일에 저장합니다. perf record 명령은 수행한 CPU 샘플을 항상 기록합니다. 이 정보를 표시하도록 perf 보고서 를 구성할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

- 현재 디렉터리에는 perf 레코드를 사용하여 생성된 perf.data 파일이 있습니다. 루트 액세스 권한으로 perf.data 파일을 만든 경우 루트 액세스 권한으로 perf 보고서 도 실행해야 합니다.

절차

- CPU별로 정렬하는 동안 추가 분석을 위해 perf.data 파일의 내용을 표시합니다.

```
# perf report --sort cpu
```

  - CPU 및 명령으로 정렬하여 CPU 시간이 사용되는 위치에 대한 자세한 정보를 표시할 수 있습니다.

```
# perf report --sort cpu,comm
```

    이 예제에서는 모든 모니터링된 CPU에서 오버헤드 사용량의 내림차순으로 총 오버헤드별 명령을 나열하고 명령이 실행된 CPU를 식별합니다.

추가 리소스

- perf를 사용하여 성능 프로파일 기록 및 분석

22.3. PERF TOP을 사용하여 프로파일링하는 동안 특정 CPU 표시

시스템을 실시간으로 프로파일링하는 동안 특정 CPU 및 상대적 사용량을 표시하도록 perf top 을 구성할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

- CPU별로 정렬하는 동안 perf 최상위 인터페이스를 시작합니다.

```
# perf top --sort cpu
```

  이 예제에서는 CPU 및 각 오버헤드를 실시간으로 오버헤드 사용량의 내림차순으로 나열합니다.

- CPU 및 명령별로 정렬하여 CPU 시간이 사용되는 위치에 대한 자세한 정보를 확인할 수 있습니다.

```
# perf top --sort cpu,comm
```

이 예제에서는 총 오버헤드별 명령을 오버헤드 사용량의 내림차순으로 나열하고 명령이 실시간으로 실행된 CPU를 식별합니다.

22.4. PERF 레코드 및 PERF 보고서를 사용하여 특정 CPU 모니터링

관심 있는 특정 CPU만 샘플하도록 perf 레코드 를 구성하고 추가 분석을 위해 perf 보고서 를 사용하여 생성된 perf.data 파일을 분석할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

1. 특정 CPU의 성능 데이터를 샘플링 및 기록하여 perf.data 파일을 생성합니다.

   - 쉼표로 구분된 CPU 목록 사용:

   ```
   # perf record -C 0,1 sleep seconds
   ```

   이전 예제는 sleep 명령을 사용하여 지정한 시간(초) 동안 CPU 0과 1의 데이터를 샘플링 및 기록합니다.

   - CPU 범위 사용:

   ```
   # perf record -C 0-2 sleep seconds
   ```

   이전 예제는 sleep 명령을 사용하여 지정한 시간(초) 동안 CPU 0에서 2로 모든 CPU의 데이터를 샘플링 및 기록합니다.

2. 추가 분석을 위해 **perf.data** 파일의 내용을 표시합니다.

```
# perf report
```

이 예에서는 **perf.data** 의 내용을 표시합니다. 여러 **CPU**를 모니터링하고 샘플한 **CPU** 데이터를 알고자 하는 경우, **perf** 보고서에서 가져온 **CPU** 샘플 표시를 참조하십시오.

23장. PERF를 사용하여 애플리케이션 성능 모니터링

이 섹션에서는 **perf** 툴을 사용하여 애플리케이션 성능을 모니터링하는 방법에 대해 설명합니다.

23.1. 실행 중인 프로세스에 PERF 레코드 연결

실행 중인 프로세스에 **perf** 레코드 를 연결할 수 있습니다. 이렇게 하면 지정된 프로세스의 성능 데이터만 샘플링하고 기록하도록 **perf** 레코드에 지시합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- 실행 중인 프로세스에 **perf** 레코드 를 연결합니다.

```
$ perf record -p ID1,ID2 sleep seconds
```

이전 예제에서는 **sleep** 명령을 사용하여 지정된 시간(초) 동안 프로세스 ***ID1*** 및 ***ID2*** 를 사용하여 프로세스의 성능 데이터를 샘플링하고 기록합니다. 특정 스레드의 이벤트를 기록하도록 **perf** 를 구성할 수도 있습니다.

```
$ perf record -t ID1,ID2 sleep seconds
```

참고

t 플래그를 사용하고 스레드 **ID**를 지정하는 경우 **perf** 는 기본적으로 상속을 비활성화합니다. **--inherit** 옵션을 추가하여 상속을 활성화할 수 있습니다.

23.2. PERF 레코드를 사용하여 호출 그래프 데이터 캡처

성능 프로필의 다른 기능을 호출하는 함수를 기록하도록 **perf** 레코드 툴을 구성할 수 있습니다. 여러 프로세스가 동일한 함수를 호출하는 경우 성능 장애를 식별하는 데 도움이 됩니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- **call-graph** 옵션을 사용하여 성능 데이터를 샘플링 하고 기록합니다.

```
$ perf record --call-graph method command
```

  - 데이터를 샘플링하려는 *명령으로* 명령을 바꿉니다. 명령을 지정하지 않으면 **Ctrl+C** 를 눌러 수동으로 중지할 때까지 **perf** 레코드에서 데이터를 샘플링합니다.

  - *방법을* 다음과 같은 팝업 방법 중 하나로 바꿉니다.

    fp

    프레임 포인터 방법을 사용합니다. 컴파일러 최적화(예: GCC 옵션 **--fomit-frame-pointer** 로 빌드된 바이너리)에 따라 스택이 풀링되지 않을 수 있습니다.

    dwarf

    DWARF 호출 프레임 정보를 사용하여 스택을 제거합니다.

    lbr

    Intel 프로세서에서 마지막 분기 레코드 하드웨어를 사용합니다.

추가 리소스

- **perf-record(1)** 도움말 페이지

23.3. PERF 보고서를 사용하여 PERF.DATA 분석

perf 보고서를 사용하여 **perf .data** 파일을 표시하고 분석할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

- 현재 디렉터리에는 **perf.data** 파일이 있습니다.

- 루트 액세스 권한으로 **perf.data** 파일을 만든 경우 루트 액세스 권한으로 **perf** 보고서 도 실행해야 합니다.

절차

- 추가 분석을 위해 **perf.data** 파일의 내용을 표시합니다.

```
# perf report
```

이 명령은 다음과 유사한 출력을 표시합니다.

```
Samples: 2K of event 'cycles', Event count (approx.): 235462960
Overhead  Command          Shared Object                     Symbol
  2.36%  kswapd0          [kernel.kallsyms]                 [k] page_vma_mapped_walk
  2.13%  sssd_kcm         libc-2.28.so                      [.] memset_avx2_erms 2.13% perf
[kernel.kallsyms] [k] smp_call_function_single 1.53% gnome-shell libc-2.28.so [.]
strcmp_avx2
  1.17%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_hash_table_lookup
  0.93%  Xorg             libc-2.28.so                      [.] memmove_avx_unaligned_erms 0.89%
gnome-shell libgobject-2.0.so.0.5600.4 [.] g_object_unref 0.87% kswapd0 [kernel.kallsyms]
[k] page_referenced_one 0.86% gnome-shell libc-2.28.so [.] memmove_avx_unaligned_erms
  0.83%  Xorg             [kernel.kallsyms]                 [k] alloc_vmap_area
  0.63%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_slice_alloc
  0.53%  gnome-shell      libgirepository-1.0.so.1.0.0      [.] g_base_info_unref
  0.53%  gnome-shell      ld-2.28.so                        [.] _dl_find_dso_for_object
  0.49%  kswapd0          [kernel.kallsyms]                 [k] vma_interval_tree_iter_next
  0.48%  gnome-shell      libpthread-2.28.so                [.] pthread_getspecific 0.47% gnome-
shell libgirepository-1.0.so.1.0.0 [.] 0x0000000000013b1d 0.45% gnome-shell libglib-
2.0.so.0.5600.4 [.] g_slice_free1 0.45% gnome-shell libgobject-2.0.so.0.5600.4 [.]
g_type_check_instance_is_fundamentally_a 0.44% gnome-shell libc-2.28.so [.] malloc 0.41%
swapper [kernel.kallsyms] [k] apic_timer_interrupt 0.40% gnome-shell ld-2.28.so [.]
_dl_lookup_symbol_x 0.39% kswapd0 [kernel.kallsyms] [k]
raw_callee_save___pv_queued_spin_unlock
```

추가 리소스

- **perf-report(1)** 도움말 페이지

24장. PERF를 사용하여 UPROBES 생성

24.1. PERF를 사용하여 함수 수준에서 UPROBES 생성

perf 툴을 사용하여 프로세스 또는 애플리케이션의 임의 지점에서 동적 추적 포인트를 생성할 수 있습니다. 그런 다음 이러한 추적 포인트를 perf 통계 및 perf 레코드 와 같은 다른 perf 툴과 함께 사용하여 프로세스 또는 애플리케이션 동작을 더 잘 이해할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

1. 프로세스 또는 애플리케이션에서 관심 있는 위치를 모니터링하는 데 관심이 있는 프로세스 또는 애플리케이션에서 uprobe를 생성합니다.

```
# perf probe -x /path/to/executable -a function
Added new event:
  probe_executable:function   (on function in /path/to/executable)

You can now use it in all perf tools, such as:

        perf record -e probe_executable:function -aR sleep 1
```

추가 리소스

- perf-probe 도움말 페이지
- perf를 사용하여 성능 프로파일 기록 및 분석
- perf stat를 사용하여 프로세스 실행 중 이벤트 계산

24.2. PERF를 사용하여 함수 내의 행에 UPROBES 생성

그런 다음 이러한 추적 포인트를 perf 통계 및 perf 레코드 와 같은 다른 perf 툴과 함께 사용하여 프로세스 또는 애플리케이션 동작을 더 잘 이해할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.
- 실행 파일에 대한 디버깅 기호를 표시했습니다.

```
# objdump -t ./your_executable | head
```

참고

이렇게 하려면 실행 파일의 **debuginfo** 패키지를 설치해야 하며, 실행 파일이 로컬에서 개발된 애플리케이션인 경우 **GCC**의 **-g** 옵션인 디버깅 정보를 사용하여 애플리케이션을 컴파일해야 합니다.

절차

1. uprobe를 배치할 수 있는 함수 행을 확인합니다.

```
$ perf probe -x ./your_executable -L main
```

이 명령의 출력은 다음과 유사합니다.

```
<main@/home/user/my_executable:0>
          0  int main(int argc, const char **argv)
          1  {
                 int err;
                 const char *cmd;
                 char sbuf[STRERR_BUFSIZE];

                 /* libsubcmd init */
          7      exec_cmd_init("perf", PREFIX, PERF_EXEC_PATH,
EXEC_PATH_ENVIRONMENT);
          8      pager_init(PERF_PAGER_ENVIRONMENT);
```

2. 원하는 함수 행에 대한 uprobe를 생성합니다.

```
# perf probe -x ./my_executable main:8
Added new event:
      probe_my_executable:main_L8   (on main:8 in /home/user/my_executable)
```

```
You can now use it in all perf tools, such as:

    perf record -e probe_my_executable:main_L8 -aR sleep 1
```

24.3. UPROBES에서 기록된 데이터의 PERF 스크립트 출력

uprobes를 사용하여 수집된 데이터를 분석하는 일반적인 방법은 perf 스크립트 명령을 사용하여 perf.data 파일을 읽고 기록된 워크로드의 상세 추적을 표시하는 것입니다.

perf 스크립트 출력 예에서 * uprobe는 my_prog * 라는 프로그램에서 함수 isprime() 에 추가됩니다. 또는 uprobe를 추가하는 위치의 코드 범위에 임의 변수가 표시될 수 있습니다.

```
# perf script
   my_prog  1367 [007] 10802159.906593: probe_my_prog:isprime: (400551) a=2
   my_prog  1367 [007] 10802159.906623: probe_my_prog:isprime: (400551) a=3
   my_prog  1367 [007] 10802159.906625: probe_my_prog:isprime: (400551) a=4
   my_prog  1367 [007] 10802159.906627: probe_my_prog:isprime: (400551) a=5
   my_prog  1367 [007] 10802159.906629: probe_my_prog:isprime: (400551) a=6
   my_prog  1367 [007] 10802159.906631: probe_my_prog:isprime: (400551) a=7
   my_prog  1367 [007] 10802159.906633: probe_my_prog:isprime: (400551) a=13
   my_prog  1367 [007] 10802159.906635: probe_my_prog:isprime: (400551) a=17
   my_prog  1367 [007] 10802159.906637: probe_my_prog:isprime: (400551) a=19
```

25장. PERF MEM을 사용하여 메모리 액세스 프로파일링

perf mem 명령을 사용하여 시스템의 메모리 액세스를 샘플링할 수 있습니다.

25.1. PERF MEM의 목적

perf 툴의 **mem** 하위 명령을 사용하면 메모리 액세스(로드 및 저장소)를 샘플링할 수 있습니다. **perf mem** 명령은 메모리 대기 시간, 메모리 액세스 유형, 캐시 적중 및 누락을 유발하는 함수, 데이터 기호를 기록하여 이러한 적중 및 누락이 발생하는 메모리 위치에 대한 정보를 제공합니다.

25.2. PERF MEM을 사용한 메모리 액세스 샘플링

다음 절차에서는 **perf mem** 명령을 사용하여 시스템의 메모리 액세스를 샘플하는 방법을 설명합니다. 명령은 **perf** 레코드 및 **perf** 보고서 와 동일한 옵션은 물론 **mem** 하위 명령에 배타적인 일부 옵션을 사용합니다. 기록된 데이터는 나중에 분석을 위해 현재 디렉터리의 **perf.data** 파일에 저장됩니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

1. 메모리 액세스 샘플:

```
# perf mem record -a sleep seconds
```

이 예제에서는 **sleep** 명령으로 지정된 *시간* 동안 모든 **CPU**에서 메모리 액세스를 샘플링합니다. 메모리 액세스 데이터를 샘플링하려는 모든 명령에 대해 **sleep** 명령을 교체할 수 있습니다. 기본적으로 **perf mem** 은 메모리 로드 및 저장소를 모두 샘플링합니다. **t** 옵션을 사용하고 **mem** 과 레코드 간에 "로드" 또는 "저장소"를 지정하여 하나의 메모리 작업만 선택할 수 있습니다. 로드의 경우 메모리 계층 구조 수준, **TLB** 메모리 액세스, 버스 **snoops** 및 메모리 잠금에 대한 정보가 캡처됩니다.

2. 분석을 위해 **perf.data** 파일을 엽니다.

```
# perf mem report
```

예제 명령을 사용한 경우 출력은 다음과 같습니다.

```
Available samples
35k cpu/mem-loads,ldlat=30/P
54k cpu/mem-stores/P
```

cpu/mem-loads,ldlat=30/P 행은 메모리 로드를 통해 수집된 데이터를 나타내며 **cpu/mem-stores/P** 행은 메모리 저장소를 통해 수집된 데이터를 나타냅니다. 관심 범주를 강조 표시하고 **Enter** 키를 눌러 데이터를 확인합니다.

```
Samples: 35K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 4067062
Overhead       Samples  Local Weight  Memory access            Symbol
Shared Object               Data Symbol                                  Data Object
Snoop       TLB access           Locked
  0.07%           29  98          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.06%           26  97          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.06%           25  96          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.06%           1  2325        Uncached or N/A hit      [k] pci_azx_readl
[kernel.kallsyms]           [k] 0xffffb092c06e9084
[kernel.kallsyms]                None        L1 or L2 hit          No
  0.06%           1  2247        Uncached or N/A hit      [k] pci_azx_readl
[kernel.kallsyms]           [k] 0xffffb092c06e8164
[kernel.kallsyms]                None        L1 or L2 hit          No
  0.05%           1  2166        L1 or L1 hit            [.] 0x00000000038140d6
libxul.so                 [.] 0x00007ffd7b84b4a8                       [stack]
None        L1 or L2 hit          No
  0.05%           1  2117        Uncached or N/A hit      [k] check_for_unclaimed_mmio
[kernel.kallsyms]           [k] 0xffffb092c1842300
[kernel.kallsyms]                None        L1 or L2 hit          No
  0.05%           22  95          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.05%           1  1898        L1 or L1 hit            [.] 0x0000000002a30e07
libxul.so                 [.] 0x00007f610422e0e0                       anon
None        L1 or L2 hit          No
  0.05%           1  1878        Uncached or N/A hit      [k] pci_azx_readl
[kernel.kallsyms]           [k] 0xffffb092c06e8164
[kernel.kallsyms]                None        L2 miss               No
  0.04%           18  94          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0        [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit          No
  0.04%           1  1593        Local RAM or RAM hit     [.] 0x00000000026f907d
libxul.so                 [.] 0x00007f3336d50a80                       anon
Hit         L2 miss               No
  0.03%           1  1399        L1 or L1 hit            [.] 0x00000000037cb5f1
libxul.so                 [.] 0x00007fbe81ef5d78                       libxul.so
```

```
None       L1 or L2 hit        No
  0.03%          1  1229       LFB or LFB hit         [.] 0x0000000002962aad
libxul.so                [.] 0x00007fb6f1be2b28                          anon
None       L2 miss             No
  0.03%          1  1202       LFB or LFB hit         [.] __pthread_mutex_lock
libpthread-2.29.so       [.] 0x00007fb75583ef20                            anon
None       L1 or L2 hit        No
  0.03%          1  1193       Uncached or N/A hit    [k] pci_azx_readl
[kernel.kallsyms]        [k] 0xffffb092c06e9164
[kernel.kallsyms]             None       L2 miss             No
  0.03%          1  1191       L1 or L1 hit           [k] azx_get_delay_from_lpib
[kernel.kallsyms]        [k] 0xffffb092ca7efcf0
[kernel.kallsyms]             None       L1 or L2 hit        No
```

또는 데이터를 표시할 때 관심 있는 다양한 측면을 조사하기 위해 결과를 정렬할 수 있습니다. 예를 들어 샘플링 기간 중 발생하는 메모리 액세스 유형별로 메모리 로드를 통한 데이터를 고려하는 오버헤드 순서의 내림차순으로 정렬하려면 다음을 수행합니다.

```
# perf mem -t load report --sort=mem
```

예를 들어 출력은 다음과 같습니다.

```
Samples: 35K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 40670
Overhead       Samples  Memory access
  31.53%          9725  LFB or LFB hit
  29.70%         12201  L1 or L1 hit
  23.03%          9725  L3 or L3 hit
  12.91%          2316  Local RAM or RAM hit
   2.37%           743  L2 or L2 hit
   0.34%             9  Uncached or N/A hit
   0.10%            69  I/O or N/A hit
   0.02%           825  L3 miss
```

추가 리소스

- **perf-mem(1)** 도움말 페이지.

25.3. PERF MEM 보고서 출력 해석

수정자가 데이터를 여러 열로 정렬하지 않고 **perf mem report** 명령을 실행하여 표시되는 테이블입니다.

'Overhead' 열

이 특정 기능에서 수집한 전체 샘플의 백분율을 나타냅니다.

'Samples' 열

는 해당 행에 의해 계정된 샘플 수를 표시합니다.

'Local Weight' 열

프로세서 코어 사이클의 액세스 대기 시간을 표시합니다.

'Memory Access' 열

발생한 메모리 액세스 유형을 표시합니다.

'Symbol' 열

함수 이름 또는 기호를 표시합니다.

'Shared Object' 열

는 샘플이 제공된 **ELF** 이미지의 이름을 표시합니다(커널.**kallsyms** 이름이 커널에서 온 경우 이름).

'Data Symbol' 열

행이 대상으로 지정된 메모리 위치의 주소를 표시합니다.

중요

액세스 중인 메모리 또는 스택 메모리의 동적 할당으로 인해 종종 **'Data Symbol'** 열에 원시 주소가 표시됩니다.

"Snoop" 열

버스 트랜잭션 표시.

'TLB 액세스**'** 열

TLB 메모리 액세스를 표시합니다.

'잠김**'** 열

함수가 메모리가 잠겨 있지 않은지 여부를 나타냅니다.

기본 모드에서는 함수가 가장 높은 오버헤드가 가장 높은 함수가 먼저 내림차순으로 정렬됩니다.

26장. 잘못된 공유 감지

거짓 공유는 SMP(Simmetric Multi Processing) 시스템의 프로세서 코어가 다른 프로세서에서 사용 중인 동일한 캐시 라인의 데이터 항목을 수정하여 프로세서 간에 공유되지 않는 다른 데이터 항목에 액세스할 때 발생합니다.

이러한 초기 수정을 위해서는 캐시 라인을 사용하는 다른 프로세서에서 복사본을 무효화하고 프로세서가 필요하지 않거나 수정된 데이터 항목의 업데이트된 버전에 액세스해야 합니다.

perf c2c 명령을 사용하여 잘못된 공유를 탐지할 수 있습니다.

26.1. PERF C2C의 목적

perf 툴의 c2c 하위 명령은 Shared Data Cache-to-Cache(C2C) 분석을 활성화합니다. perf c2c 명령을 사용하여 캐시 라인 경합을 검사하여 true 및 false 공유를 탐지할 수 있습니다.

SMP(Symetric Multi Processing) 시스템의 프로세서 코어가 다른 프로세서에서 사용 중인 동일한 캐시 라인의 데이터 항목을 수정할 때 캐시 라인이 발생합니다. 이 캐시 라인을 사용하는 다른 모든 프로세서는 해당 사본을 무효화하고 업데이트된 프로세서를 요청해야 합니다. 이로 인해 성능이 저하될 수 있습니다.

perf c2c 명령은 다음과 같은 정보를 제공합니다.

- 경합이 감지된 캐시 라인
- 데이터 읽기 및 작성 프로세스
- 경합을 유발하는 명령
- 경합과 관련된 NUMA(Non-Uniform Memory Access) 노드

26.2. PERF C2C를 사용하여 캐시 라인 경합 감지

perf c2c 명령을 사용하여 시스템의 캐시 라인 경합을 탐지합니다.

perf c2c 명령은 **perf** 레코드와 동일한 옵션과 **c2c** 하위 명령에 배타적인 일부 옵션을 지원합니다. 기록된 데이터는 나중에 분석을 위해 현재 디렉터리의 **perf.data** 파일에 저장됩니다.

사전 요구 사항

- **perf** 사용자 공간 도구가 설치됩니다. 자세한 내용은 perf 설치를 참조하십시오.

절차

- **perf c2c** 를 사용하여 캐시 라인 경합을 감지합니다.

```
# perf c2c record -a sleep seconds
```

이 예제에서는 *절전* 명령으로 지정된 기간 동안 모든 **CPU**에서 캐시 라인 경합 데이터를 샘플링하고 기록합니다. **sleep** 명령을 를 통해 캐시 라인 경합 데이터를 수집하려는 모든 명령으로 교체할 수 있습니다.

추가 리소스

- **perf-c2c(1)** 도움말 페이지

26.3. PERF C2C 레코드로 기록된 PERF.DATA 파일 시각화

이 절차에서는 **perf c2c** 명령을 사용하여 기록된 **perf.data** 파일을 시각화하는 방법을 설명합니다.

사전 요구 사항

- **perf** 사용자 공간 도구가 설치됩니다. 자세한 내용은 Installing perf 를 참조하십시오.

- **perf c2c** 명령을 사용하여 기록된 **perf.data** 파일은 현재 디렉터리에서 사용할 수 있습니다. 자세한 내용은 perf c2c를 사용하여 캐시 라인 경합 탐지를 참조하십시오.

절차

1. 추가 분석을 위해 **perf.data** 파일을 엽니다.

```
# perf c2c report --stdio
```

이 명령은 **perf.data** 파일을 터미널 내의 여러 그래프로 시각화합니다.

```
=================================================
        Trace Event Information
=================================================
 Total records                     :     329219
 Locked Load/Store Operations      :      14654
 Load Operations                   :      69679
 Loads - uncacheable               :          0
 Loads - IO                        :          0
 Loads - Miss                      :       3972
 Loads - no mapping                :          0
 Load Fill Buffer Hit              :      11958
 Load L1D hit                      :      17235
 Load L2D hit                      :         21
 Load LLC hit                      :      14219
 Load Local HITM                   :       3402
 Load Remote HITM                  :      12757
 Load Remote HIT                   :       5295
 Load Local DRAM                   :        976
 Load Remote DRAM                  :       3246
 Load MESI State Exclusive         :       4222
 Load MESI State Shared            :          0
 Load LLC Misses                   :      22274
 LLC Misses to Local DRAM          :       4.4%
 LLC Misses to Remote DRAM         :      14.6%
 LLC Misses to Remote cache (HIT)  :      23.8%
 LLC Misses to Remote cache (HITM) :      57.3%
 Store Operations                  :     259539
 Store - uncacheable               :          0
 Store - no mapping                :         11
 Store L1D Hit                     :     256696
 Store L1D Miss                    :       2832
 No Page Map Rejects               :       2376
 Unable to parse data source       :          1

=================================================
  Global Shared Cache Line Event Information
=================================================
 Total Shared Cache Lines          :         55
 Load HITs on shared lines         :      55454
 Fill Buffer Hits on shared lines  :      10635
 L1D hits on shared lines          :      16415
 L2D hits on shared lines          :          0
 LLC hits on shared lines          :       8501
 Locked Access on shared lines     :      14351
 Store HITs on shared lines        :     109953
 Store L1D hits on shared lines    :     109449
```

```
 Total Merged records            :     126112

=================================================
            c2c details
=================================================
 Events                         : cpu/mem-loads,ldlat=30/P
                                : cpu/mem-stores/P
 Cachelines sort on             : Remote HITMs
 Cacheline data groupping       : offset,pid,iaddr

=================================================
   Shared Data Cache Line Table
=================================================
#
#                        Total      Rmt  ----- LLC Load Hitm -----  ---- Store Reference ----  ---
Load Dram ----      LLC    Total  ----- Core Load Hit -----  -- LLC Load Hit --
# Index          Cacheline  records     Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss
Lcl       Rmt  Ld Miss    Loads       FB       L1       L2       Llc       Rmt
# .....  ..................  .......  .......  .......  .......  .......  .......  .......  .......  ........  ........  .......  .......  .......
.......  .......  ........  ........
#
    0          0x602180   149904   77.09%    12103     2269     9834   109504   109036      468
727      2657    13747    40400     5355    16154        0     2875      529
    1          0x602100    12128   22.20%     3951     1119     2832        0        0        0       65
200     3749    12128     5096      108        0     2056      652
    2  0xffff883ffb6a7e80      260    0.09%       15        3       12      161      161        0        1
1       15       99       25       50        0        6        1
    3  0xffffffff81aec000      157    0.07%        9        0        9        1        0        1        0        7
20      156       50       59        0       27        4
    4  0xffffffff81e3f540      179    0.06%        9        1        8      117       97       20        0
10       25       62       11        1        0       24        7

=================================================
    Shared Cache Line Distribution Pareto
=================================================
#
#       ----- HITM -----  -- Store Refs --        Data address                          ---------- cycles
----------       cpu                                Shared
#   Num      Rmt      Lcl   L1 Hit  L1 Miss              Offset      Pid        Code address  rmt
hitm  lcl hitm      load       cnt             Symbol              Object                Source:Line
Node{cpu list}
# .....  .......  .......  .......  .......  ...................  .......  ...................  ........  ........  ........  ........
.....................  .....................  .............................  ....
#
  ------------------------------------------------------------
    0     9834     2269   109036      468           0x602180
  ------------------------------------------------------------
       65.51%   55.88%   75.20%    0.00%                0x0    14604          0x400b4f     27161
26039     26017        9  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:144   0{0-1,4}  1{24-25,120}  2{48,54}  3{169}
   0.41%    0.35%    0.00%    0.00%                0x0    14604          0x400b56     18088
12601     26671        9  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:145   0{0-1,4}  1{24-25,120}  2{48,54}  3{169}
   0.00%    0.00%   24.80%  100.00%                0x0    14604          0x400b61        0        0
0        9  [.] read_write_func  no_false_sharing.exe  false_sharing_example.c:145   0{0-
1,4}  1{24-25,120}  2{48,54}  3{169}
```

```
  7.50%  9.92%  0.00%  0.00%           0x20  14604        0x400ba7     2470
1729     1897       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:154  1{122}  2{144}
 17.61%  20.89%  0.00%  0.00%           0x28  14604        0x400bc1     2294
1575     1649       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:158  2{53}  3{170}
  8.97%  12.96%  0.00%  0.00%           0x30  14604        0x400bdb     2325
1897     1828       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:162  0{96}  3{171}

-------------------------------------------------------------
   1    2832     1119       0       0          0x602100
-------------------------------------------------------------
 29.13%  36.19%  0.00%  0.00%           0x20  14604        0x400bb3     1964
1230     1788       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:155  1{122}  2{144}
 43.68%  34.41%  0.00%  0.00%           0x28  14604        0x400bcd     2274
1566     1793       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:159  2{53}  3{170}
 27.19%  29.40%  0.00%  0.00%           0x30  14604        0x400be7     2045
1247     2011       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:163  0{96}  3{171}
```

26.4. PERF C2C 보고서 출력 해석

이 섹션에서는 **perf c2c report** 명령의 출력을 해석하는 방법을 설명합니다.

perf c2c report --stdio 명령을 실행하여 표시되는 시각화에서 데이터를 여러 테이블로 정렬합니다.

이벤트 정보 추적

이 표는 **perf c2c record** 명령으로 수집한 모든 부하 및 저장소 샘플에 대한 간략한 요약을 제공합니다.

글로벌 공유 캐시 라인 이벤트 정보

이 표는 공유 캐시 라인에 대한 통계를 제공합니다.

C2c 세부 정보

이 표에서는 샘플링된 이벤트와 **perf c2c** 보고서 데이터가 시각화 내에서 구성하는 방법에 대한 정보를 제공합니다.

공유 데이터 캐시 라인 테이블

이 표에서는 **false** 공유가 감지되고 기본적으로 캐시 행당 감지된 원격 **Hitm** 의 양만큼 내림차순으로 정렬되는 가장 빠른 캐시 행에 대한 한 줄 요약을 제공합니다.

공유 캐시 라인 배포 **Pareto**

이 표는 경합이 발생하는 각 캐시 라인에 대한 다양한 정보를 제공합니다.

- 캐시 행은 **0** 부터 시작하여 **NUM** 열에 번호가 지정됩니다.
- 각 캐시 라인의 가상 주소는 데이터 주소 오프셋 열에 포함되고 그 뒤에 다른 액세스가 발생한 캐시 라인의 오프셋이 옵니다.
- **Pid** 열에는 프로세스 **ID**가 포함되어 있습니다.
- **Code Address**(코드 주소) 열에는 명령 포인터 코드 주소가 포함되어 있습니다.
- **cycle** 레이블 아래의 열 에는 평균 부하 대기 시간이 표시됩니다.
- **cpu cnt** 열에는 에서 가져온 다양한 **CPU** 샘플 수가 표시됩니다(기본적으로 해당 위치에 인덱싱된 데이터에 대해 대기 중인 **CPU** 수는 몇 개입니까).
- 심볼릭 열 에는 함수 이름 또는 기호가 표시됩니다.
- **Shared Object** 열에는 샘플이 제공된 **ELF** 이미지의 이름이 표시됩니다(커널에서 샘플이 올 때 **[kernel.kallsyms]** 이름).
- **Source:line** 열에는 소스 파일 및 행 번호가 표시됩니다.
- **Node{cpu list}** 열에는 각 노드에 대해 제공된 특정 **CPU** 샘플이 표시됩니다.

26.5. PERF C2C를 사용하여 FALSE 공유 감지

다음 절차에서는 **perf c2c** 명령을 사용하여 **false** 공유를 감지하는 방법을 설명합니다.

사전 요구 사항

- perf 사용자 공간 도구가 설치됩니다. 자세한 내용은 perf 설치를 참조하십시오.

- perf c2c 명령을 사용하여 기록된 perf.data 파일은 현재 디렉터리에서 사용할 수 있습니다. 자세한 내용은 perf c2c를 사용하여 캐시 라인 경합 탐지를 참조하십시오.

절차

1. 추가 분석을 위해 perf.data 파일을 엽니다.

```
# perf c2c report --stdio
```

그러면 터미널에서 perf.data 파일이 열립니다.

2. "Trace Event Information(Trace Event Information)" 표에서 HITM(Remote Cache)의 값이 포함된 행을 찾습니다.

HI TM(Remote Caches to Remote Cache) 행의 value 열에 있는 백분율은 수정된 캐시 라인에서 NUMA 노드에서 발생한 LLC 누락 비율을 나타내며 핵심 표시자 false 공유가 발생한 것입니다.

```
=================================================
         Trace Event Information
=================================================
  Total records                     :     329219
  Locked Load/Store Operations      :      14654
  Load Operations                   :      69679
  Loads - uncacheable               :          0
  Loads - IO                        :          0
  Loads - Miss                      :       3972
  Loads - no mapping                :          0
  Load Fill Buffer Hit              :      11958
  Load L1D hit                      :      17235
  Load L2D hit                      :         21
  Load LLC hit                      :      14219
  Load Local HITM                   :       3402
  Load Remote HITM                  :      12757
  Load Remote HIT                   :       5295
  Load Local DRAM                   :        976
  Load Remote DRAM                  :       3246
  Load MESI State Exclusive         :       4222
  Load MESI State Shared            :          0
  Load LLC Misses                   :      22274
  LLC Misses to Local DRAM          :       4.4%
  LLC Misses to Remote DRAM         :      14.6%
```

```
LLC Misses to Remote cache (HIT)  :      23.8%
LLC Misses to Remote cache (HITM) : 57.3%
Store Operations                  :    259539
Store - uncacheable               :         0
Store - no mapping                :        11
Store L1D Hit                     :    256696
Store L1D Miss                    :      2832
No Page Map Rejects               :      2376
Unable to parse data source       :         1
```

3. **Shared Data Cache Line** 테이블 의 **LLC Load Hitm** 필드의 **Rmt** 열 검사 :

```
=================================================
        Shared Data Cache Line Table
=================================================
#
#                        Total      Rmt  ----- LLC Load Hitm -----  ---- Store Reference ----  ---
Load Dram ----      LLC    Total  ----- Core Load Hit -----  -- LLC Load Hit --
# Index          Cacheline  records     Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss
Lcl       Rmt  Ld Miss    Loads       FB       L1       L2       Llc       Rmt
# ..... ..................  .......  .......  .......  .......  .......  .......  .......  .......  ........  ........  .......  .......  .......
.......  .......  ........  ........
#
     0          0x602180   149904   77.09%    12103     2269     9834   109504   109036
468      727      2657    13747    40400     5355    16154        0     2875      529
     1          0x602100    12128   22.20%     3951     1119     2832        0        0        0       65
200     3749    12128     5096      108        0     2056      652
     2  0xffff883ffb6a7e80      260    0.09%       15        3       12      161      161        0        1
1       15       99       25       50        0        6        1
     3  0xffffffff81aec000      157    0.07%        9        0        9        1        0        1        0        7
20      156       50       59        0       27        4
     4  0xffffffff81e3f540      179    0.06%        9        1        8      117       97       20        0
10       25       62       11        1        0       24        7
```

이 테이블은 캐시 라인당 감지된 원격 **Hitm** 의 양만큼 내림차순으로 정렬됩니다. **LLC Load Hitm** 섹션의 **Rmt** 열에 있는 높은 숫자는 **false** 공유를 나타내며 **false** 공유 활동을 디버깅하는 데 발생한 캐시 행을 추가로 검사해야 합니다.

27장. 사진 사용 시작

시스템 관리자는 그래프를 사용하여 **perf** 도구로 기록된 시스템 성능 데이터의 시각화를 생성할 수 있습니다. 소프트웨어 개발자는 그래프 를 사용하여 **perf** 툴로 기록된 애플리케이션 성능 데이터의 시각화를 생성할 수 있습니다.

샘플링 스택 추적은 **perf** 툴을 사용하여 **CPU** 성능을 프로파일링하는 일반적인 기술입니다. 안타깝게도 **perf** 를 사용한 스택 추적을 프로파일링하면 분석에 매우 자세하고 노동 집약적일 수 있습니다. 사진 에는 **perf** 로 기록된 데이터에서 생성된 시각화로 핫 코드 경로를 더 빠르고 쉽게 식별할 수 있습니다.

27.1. 사진 설치

표시 사용을 시작하려면 필수 패키지를 설치합니다.

절차

- **pilot graphs** 패키지를 설치합니다.

```
# yum install js-d3-flame-graph
```

27.2. 전체 시스템에 대한 사진 만들기

이 절차에서는 사진사를 사용하여 전체 시스템에서 기록된 성능 데이터를 시각화하는 방법을 설명합니다.

사전 요구 사항

- 갤러리 설치에 설명된 대로 설치됩니다 .
- **perf** 툴은 perf 설치에 설명된 대로 설치됩니다.

절차

- 데이터를 기록하고 시각화를 생성합니다.

```
# perf script flamegraph -a -F 99 sleep 60
```

이 명령은 **sleep** 명령을 사용하여 규정된 대로 전체 시스템에서 **60**초 동안 성능 데이터를 샘플링 및 기록한 다음, 현재 활성 디렉터리에 저장될 시각화를 **block graph.html** 로 구성합니다. 명령은 기본적으로 호출 그래프 데이터를 샘플링하고 **perf** 툴과 동일한 인수를 사용합니다. 이 경우 다음과 같습니다.

-a

전체 시스템에 대한 데이터를 기록하도록 규정합니다.

-F

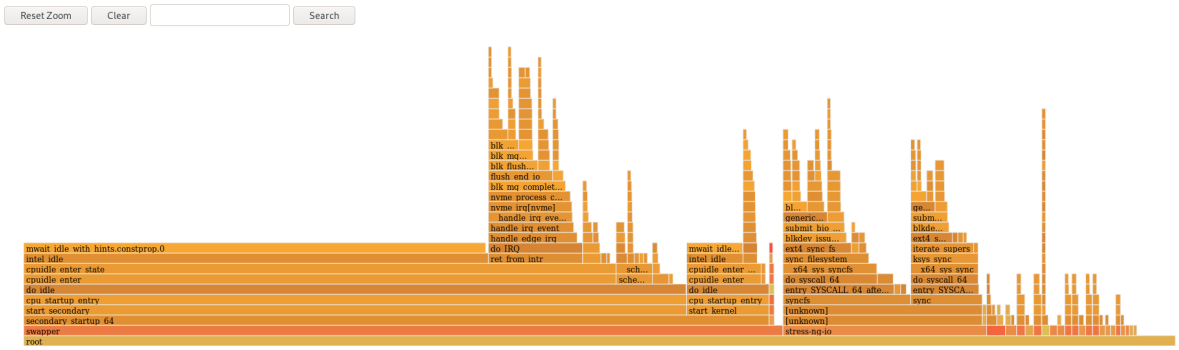
초당 샘플링 빈도를 설정하려면 다음을 수행합니다.

검증 단계

- 분석의 경우 생성된 시각화를 확인하십시오.

```
# xdg-open flamegraph.html
```

이 명령은 기본 브라우저에서 시각화를 엽니다.



27.3. 특정 프로세스에 대한 사진 생성

차트를 사용하여 실행 중인 특정 프로세스에 기록된 성능 데이터를 시각화할 수 있습니다.

사전 요구 사항

- 갤러리 설치에 설명된 대로 설치됩니다 .

- **perf** 툴은 **perf 설치에** 설명된 대로 설치됩니다.

절차

- 데이터를 기록하고 시각화를 생성합니다.

```
# perf script flamegraph -a -F 99 -p ID1,ID2 sleep 60
```

이 명령은 **sleep** 명령을 사용하여 지정된 대로 프로세스 **ID**의 ***ID1*** 및 ***ID2*** 를 **60**초 동안 샘플링하고 프로세스의 성능 데이터를 기록한 다음 현재 활성 디렉터리에 저장될 시각화를 **region graph.html** 로 구성합니다. 명령은 기본적으로 호출 그래프 데이터를 샘플링하고 **perf** 툴과 동일한 인수를 사용합니다. 이 경우 다음과 같습니다.

-a

전체 시스템에 대한 데이터를 기록하도록 규정합니다.

-F

초당 샘플링 빈도를 설정하려면 다음을 수행합니다.

-p

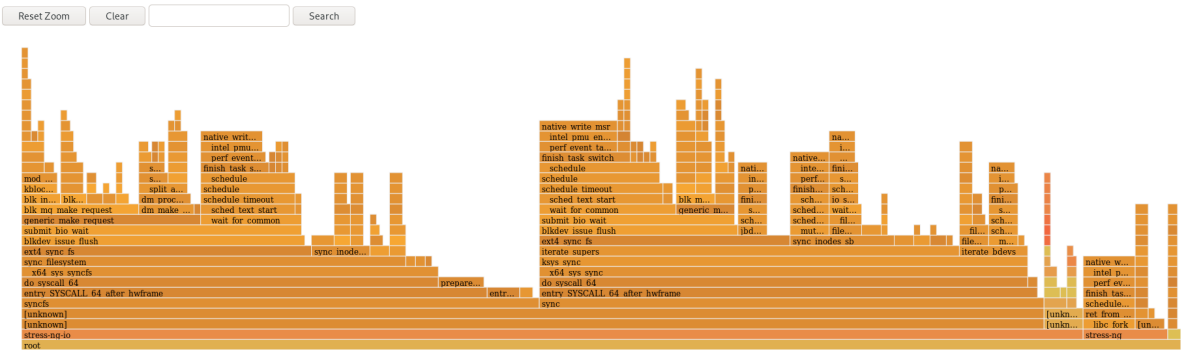
특정 프로세스 **ID**를 지정하여 에서 데이터를 샘플링하고 기록합니다.

검증 단계

- 분석의 경우 생성된 시각화를 확인하십시오.

```
# xdg-open flamegraph.html
```

이 명령은 기본 브라우저에서 시각화를 엽니다.



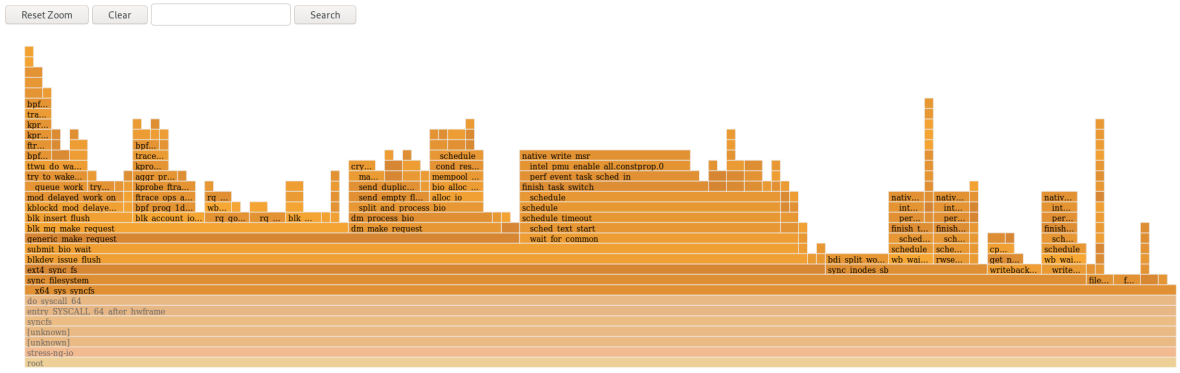
27.4. 사진사 해석

표시의 각 상자는 스택의 다른 기능을 나타냅니다. **y**축은 각 스택에 가장 높은 상자가 있는 스택의 깊이를 보여줍니다. **x**축은 샘플링된 호출 그래프 데이터의 채우기(**population**)를 표시합니다.

지정된 행에 있는 스택의 자식은 **x**축에 따라 내림차순으로 각 함수에서 가져온 샘플 수에 따라 표시됩니다. **x**축은 시간이 지난 후를 나타내지 않습니다. 개별 상자가 폭이 넓을수록 데이터가 샘플링될 때 **CPU** 상의 경우 또는 온-**CPU** 상위의 일부가 더 빈번했습니다.

절차

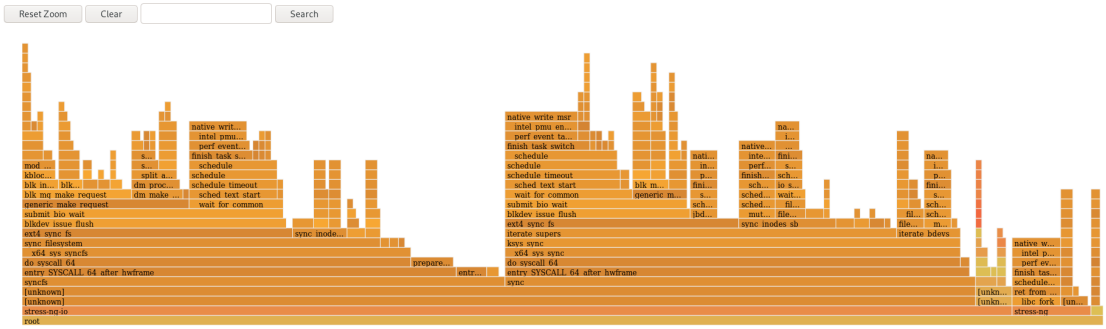
- 이전에 표시되지 않았을 수 있는 함수의 이름을 표시하여 지정된 위치에 있는 스택을 확대하기 위해 사진사 내의 상자를 클릭하여 데이터를 자세히 조사합니다.



- 표시의 기본 보기로 돌아가려면 **redhatm** 재설정을 클릭합니다.

중요

사용자 공간 함수를 나타내는 상자는 함수의 바이너리가 제거되기 때문에 아카이브에서 알 수 없음으로 레이블이 지정될 수 있습니다. 실행 파일의 **The debuginfo** 패키지를 설치해야 합니다. 또는 실행 파일이 로컬에서 개발된 애플리케이션인 경우 디버깅 정보를 사용하여 애플리케이션을 컴파일해야 합니다. **GCC**에서 **-g** 옵션을 사용하여 이러한 상황에서 함수 이름 또는 기호를 표시합니다.



추가 리소스

- perf가 일부 기능 이름을 원시 함수 주소로 표시하는 이유
- 디버깅 정보를 사용하여 디버깅 활성화

28장. PERF 원형 버퍼를 사용하여 성능 병목 현상 모니터링

perf 툴을 사용하여 이벤트별 데이터 스냅샷을 찍는 순환 버퍼를 생성하여 시스템에서 실행되는 애플리케이션의 성능 병목을 모니터링할 수 있습니다. 이러한 경우 perf 는 지정된 이벤트가 탐지되는 경우 나중에 분석을 위해 perf.data 파일에만 데이터를 씁니다.

28.1. PERF가 있는 원형 버퍼 및 이벤트 특정 스냅샷

프로세스 또는 perf 를 사용하는 애플리케이션에서 성능 문제를 조사하는 경우 관심 있는 특정 이벤트가 발생하기 전 몇 시간 동안 데이터를 기록하는 것이 경제적이거나 적절하지 않을 수 있습니다. 이러한 경우 perf 레코드 를 사용하여 특정 이벤트 후에 스냅샷을 가져오는 사용자 지정 순환 버퍼를 생성할 수 있습니다.

Overwrite 옵션은 perf 레코드 저장소를 덮어쓸 수 있는 순환 버퍼에 있는 모든 데이터를 만듭니다. 버퍼가 가득 차면 Perf 레코드가 가장 오래된 레코드를 자동으로 덮어쓰므로 perf.data 파일에 작성되지 않습니다.

overwrite 및 -- switch-output-event 옵션을 사용하면 --switch-output-event 트리거 이벤트를 탐지할 때까지 지속적으로 데이터를 기록하고 덤프하는 순환 버퍼를 구성합니다. 트리거 이벤트 신호는 사용자에게 관심이 있음을 perf 레코드에 전달하고 원형 버퍼의 데이터를 perf.data 파일에 기록합니다. 이렇게 하면 관심 있는 특정 데이터를 수집하고 perf. data 파일에 원하지 않는 데이터를 작성하지 않고 실행 중인 perf 프로세스의 오버헤드를 동시에 줄입니다.

28.2. PERF 원형 버퍼를 사용하여 성능 병목 현상 모니터링을 위한 특정 데이터 수집

perf 툴을 사용하면 관심 있는 데이터만 수집하기 위해 지정한 이벤트에서 트리거하는 원형 버퍼를 생성할 수 있습니다. 이벤트별 데이터를 수집하는 원형 버퍼를 생성하려면 perf 에 --overwrite 및 --switch-output-event 옵션을 사용합니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.
- 프로세스 또는 애플리케이션 내의 관심 위치에서 모니터링하는 데 관심이 있는 프로세스 또는 애플리케이션에 uprobe를 배치했습니다.

```
# perf probe -x /path/to/executable -a function
Added new event:
  probe_executable:function   (on function in /path/to/executable)
```

You can now use it in all perf tools, such as:

perf record -e *probe_executable:function* -aR sleep 1

절차

- **uprobe**를 트리거 이벤트로 사용하여 원형 버퍼를 생성합니다.

```
# perf record --overwrite -e cycles --switch-output-event probe_executable:function
./executable
[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012231959 ]
[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012232008 ]
^C[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012232082 ]
[ perf record: Captured and wrote 5.621 MB perf.data.<timestamp> ]
```

이 예제에서는 실행 파일을 시작하고 perf 가 --switch-output-event 옵션 다음에 지정된 트리거 이벤트인 uprobe를 감지할 때까지 -e 옵션 다음에 지정된 cpu 사이클을 수집합니다. 이 시점에서 perf 는 원형 버퍼에 있는 모든 데이터의 스냅샷을 가져와 타임스탬프로 식별된 고유한 perf.data 파일에 저장합니다. 이 예제에서는 모두 2개의 스냅샷을 생성했으며 마지막 perf.data 파일은 Ctrl+c를 눌러 강제 적용되었습니다.

29장. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에서 추적 지점 추가 및 제거

제어 파이프 인터페이스를 사용하여 실행 중인 **perf** 수집기에서 다양한 추적 포인트를 활성화하고 비활성화하면 **perf** 를 중지하거나 다시 시작하지 않고도 수집할 데이터를 동적으로 조정할 수 있습니다 . 이렇게 하면 프로세스를 중지하거나 다시 시작하는 동안 기록된 성능 데이터가 손실되지 않습니다.

29.1. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에 추적 지점 추가

제어 파이프 인터페이스를 사용하여 실행 중인 **perf** 수집기에 추적 포인트를 추가하여 **perf** 를 중지하고 성능 데이터 손실 없이 기록하는 데이터를 조정합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

1. 제어 파이프 인터페이스를 구성합니다.

```
# mkfifo control ack perf.pipe
```

2. 활성화하려는 제어 파일 설정 및 이벤트를 사용하여 **perf** 레코드 를 실행합니다.

```
# perf record --control=fifo:control,ack -D -1 --no-buffering -e 'sched:*' -o - > perf.pipe
```

이 예에서 **-e** 옵션 다음에 **'ched:*'** 선언은 스케줄러 이벤트와 함께 **perf** 레코드 를 시작합니다.

3. 두 번째 터미널에서 제어 파이프의 읽기 측면을 시작합니다.

```
# cat perf.pipe | perf --no-pager script -i -
```

제어 파이프의 읽기 측면을 시작하면 첫 번째 터미널에서 다음 메시지가 트리거됩니다.

```
Events disabled
```

4. 세 번째 터미널에서 제어 파일을 사용하여 추적 지점을 활성화합니다.

```
# echo 'enable sched:sched_process_fork' > control
```

이 명령은 **perf** 를 트리거하여 선언된 이벤트에 대해 제어 파일의 현재 이벤트 목록을 스캔합니다. 이벤트가 있는 경우 추적 지점이 활성화되고 첫 번째 터미널에 다음 메시지가 표시됩니다.

```
event sched:sched_process_fork enabled
```

추적 지점이 활성화되면 두 번째 터미널은 추적 지점을 감지하는 **perf** 의 출력을 표시합니다.

```
bash 33349 [034] 149587.674295: sched:sched_process_fork: comm=bash pid=33349
child_comm=bash child_pid=34056
```

29.2. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에서 추적 지점 제거

제어 파이프 인터페이스를 사용하여 실행 중인 **perf** 수집기에서 추적 지점을 제거하여 **perf** 를 중지하고 성능 데이터 손실 없이 수집할 데이터 범위를 줄입니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- 제어 파이프 인터페이스를 통해 실행 중인 **perf** 수집기에 추적 포인트를 추가했습니다. 자세한 내용은 **perf**를 중지하거나 다시 시작하지 않고 실행 중인 **perf** 수집기에 추적 지점 추가를 참조하십시오.

절차

- 추적 지점을 제거합니다.

```
# echo 'disable sched:sched_process_fork' > control
```

참고

이 예제에서는 이전에 컨트롤 파일에 스케줄러 이벤트를 로드하고 추적 포인트 **sched_process_fork**를 활성화했다고 가정합니다.

이 명령은 **perf** 를 트리거하여 선언된 이벤트에 대해 제어 파일의 현재 이벤트 목록을 스캔합니다. 이벤트가 있으면 추적 지점이 비활성화되고 제어 파이프를 구성하는 데 사용되는 터미널에 다음 메시지가 표시됩니다.

```
event sched:sched_process_fork disabled
```

30장. NUMASTAT를 사용하여 메모리 할당 프로파일링

numastat 도구를 사용하면 시스템의 메모리 할당에 대한 통계를 표시할 수 있습니다.

numastat 도구는 각 **NUMA** 노드의 데이터를 별도로 표시합니다. 이 정보를 사용하여 시스템의 메모리 성능 또는 시스템의 다양한 메모리 정책의 효과를 조사할 수 있습니다.

30.1. 기본 NUMASTAT 통계

기본적으로 **numastat** 도구는 각 **NUMA** 노드에 대한 이러한 범주에 대한 통계를 표시합니다.

numa_hit

이 노드에 성공적으로 할당된 페이지 수입니다.

numa_miss

의도한 노드에서 메모리가 부족하여 이 노드에 할당된 페이지 수입니다. 각 **numa_miss** 이벤트는 다른 노드에 해당하는 **numa_foreign** 이벤트가 있습니다.

numa_foreign

대신 다른 노드에 할당된 이 노드에 대해 처음 설계된 페이지 수입니다. 각 **numa_foreign** 이벤트에는 다른 노드에 해당하는 **numa_miss** 이벤트가 있습니다.

interleave_hit

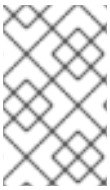
이 노드에 성공적으로 할당된 인터리브 정책 페이지 수입니다.

local_node

이 노드의 프로세스를 통해 이 노드에 성공적으로 할당된 페이지 수입니다.

other_node

다른 노드의 프로세스에 의해 이 노드에 할당된 페이지 수입니다.



참고

높은 **numa_hit** 값과 낮은 **numa_miss** 값(연간 상대적)은 최적의 성능을 나타냅니다.

30.2. NUMASTAT를 사용하여 메모리 할당 보기

numastat 도구를 사용하여 시스템의 메모리 할당을 볼 수 있습니다.

사전 요구 사항

- **numactl** 패키지를 설치합니다.

```
# yum install numactl
```

절차

- 시스템의 메모리 할당을 확인합니다.

```
$ numastat
                           node0           node1
numa_hit                76557759        92126519
numa_miss               30772308        30827638
numa_foreign            30827638        30772308
interleave_hit            106507          103832
local_node              76502227        92086995
other_node              30827840        30867162
```

추가 리소스

- **numastat(8)** 도움말 페이지

31장. CPU 사용률을 최적화하도록 운영 체제 구성

이 섹션에서는 워크로드 전반에서 CPU 사용률을 최적화하도록 운영 체제를 구성하는 방법에 대해 설명합니다.

31.1. 프로세서 문제 모니터링 및 진단 툴

다음은 Red Hat Enterprise Linux 8에서 프로세서 관련 성능 문제를 모니터링 및 진단하는 툴입니다.

- GRUBbostat 툴은 관리자가 과도한 전력 사용량, 심도 있는 절전 상태 입력 또는 불필요하게 생성되는 SMI(시스템 관리 인터럽트)와 같은 서버에서 예기치 않은 동작을 식별하는 데 도움이 되도록 카운터 결과를 출력합니다.
- numactl 유틸리티는 프로세서 및 메모리 선호도를 관리할 수 있는 다양한 옵션을 제공합니다. numactl 패키지에는 커널에서 지원하는 NUMA 정책에 대한 간단한 프로그래밍 인터페이스를 제공하는 libnuma 라이브러리가 포함되어 있으며 numactl 애플리케이션보다 더 세분화된 튜닝에 사용할 수 있습니다.
- numastat 도구는 운영 체제 및 해당 프로세스에 대한 NUMA 노드 메모리 통계를 표시하고, 관리자가 프로세스 메모리가 시스템 전체에 분산되어 있는지 또는 특정 노드에 중앙 집중식 상태인지를 보여줍니다. 이 도구는 numactl 패키지에서 제공합니다.
- numad 는 자동 NUMA 선호도 관리 데몬입니다. NUMA 리소스 할당 및 관리를 동적으로 개선하기 위해 시스템 내의 NUMA 토폴로지 및 리소스 사용량을 모니터링합니다.
- /proc/interrupts 파일은 인터럽트 요청(IRQ) 번호, 시스템의 각 프로세서가 처리하는 유사한 인터럽트 요청 수, 전송된 인터럽트 유형 및 나열된 인터럽트 요청에 응답하는 쉼표로 구분된 장치 목록을 표시합니다.
- pqos 유틸리티는 intel-cmt-cat 패키지에서 사용할 수 있습니다. 최근 Intel 프로세서에서 CPU 캐시 및 메모리 대역폭을 모니터링합니다. 모니터링:
  - 사이클당 지침(IPC).
  - 마지막 수준 캐시 MISSES의 수입니다.

- 프로그램이 SKU의 지정된 CPU 점유에서 실행되는 킬로바이트 단위 크기입니다.
- 로컬 메모리에 대한 대역폭(MBL).
- 원격 메모리에 대한 대역폭(MBR).

- x86__perf_policy 도구를 사용하면 관리자가 성능과 에너지 효율성의 상대적 중요성을 정의할 수 있습니다. 이 정보는 성능과 에너지 효율성 사이에서 전환할 옵션을 선택할 때 이 기능을 지원하는 프로세서에 영향을 주는 데 사용할 수 있습니다.
- 작업 집합 도구는 util-linux 패키지에서 제공합니다. 관리자는 실행 중인 프로세스의 프로세서 선호도를 검색 및 설정하거나 지정된 프로세서 선호도를 사용하여 프로세스를 시작할 수 있습니다.

추가 리소스

- turbostat(8), numactl(8), numastat(8), numa(7), numad(8), pqos(8), x86__perf_policy(8) 및 taskset(1) 도움말 페이지

31.2. 시스템 토폴로지 유형

오늘날의 컴퓨팅에서 대부분의 최신 시스템에는 여러 프로세서가 있기 때문에 CPU의 개념은 잘못된 개념입니다. 시스템의 토폴로지는 이러한 프로세서가 서로 그리고 다른 시스템 리소스에 연결하는 방식입니다. 이는 시스템과 애플리케이션 성능 및 시스템의 튜닝 고려 사항에 영향을 줄 수 있습니다.

다음은 최신 컴퓨팅에 사용되는 두 가지 기본 토폴로지 유형입니다.

SMP(대칭 멀티프로세서) 토폴로지

SMP 토폴로지를 사용하면 모든 프로세서가 동일한 시간 내에 메모리에 액세스할 수 있습니다. 그러나 공유 및 동일한 메모리 액세스는 본질적으로 모든 CPU에서 직렬화된 메모리 액세스를 강제하기 때문에 SMP 시스템 확장 제약 조건은 일반적으로 허용되지 않습니다. 이러한 이유로 사실상 모든 최신 서버 시스템은 NUMA 시스템입니다.

NUMA(Non-Uniform Memory Access) 토폴로지

NUMA 토폴로지는 SMP 토폴로지보다 최근에 개발되었습니다. NUMA 시스템에서 다중 프로세서는 소켓에 물리적으로 그룹화됩니다. 각 소켓에는 해당 메모리에 대한 로컬 액세스 권한이 있는 전용 메모리 및 프로세서 영역이 있으며, 이를 전체적으로 노드라고 합니다. 동일한 노드의 프로세서는 해당

노드의 메모리 은행에 대한 빠른 속도로 액세스하고, 해당 노드에 없는 메모리 은행에 대한 액세스 속도가 느려집니다.

따라서 비로컬 메모리에 액세스할 때 성능이 저하됩니다. 따라서 **NUMA** 토폴로지가 있는 시스템에서 성능에 민감한 애플리케이션은 애플리케이션을 실행하는 프로세서와 동일한 노드에 있는 메모리에 액세스해야 하며 가능한 경우 원격 메모리에 액세스하지 않아야 합니다.

성능에 민감한 다중 스레드 애플리케이션은 특정 프로세서가 아닌 특정 **NUMA** 노드에서 실행하도록 를 구성하는 것이 좋습니다. 이것이 적합한지 여부는 시스템과 애플리케이션의 요구 사항에 따라 달라집니다. 여러 애플리케이션 스레드가 동일한 캐시 데이터에 액세스하는 경우 동일한 프로세서에서 실행하도록 해당 스레드를 구성하는 것이 적합할 수 있습니다. 그러나 동일한 프로세서에서 액세스하고 캐시하는 여러 스레드가 동일한 프로세서에서 실행되는 경우 각 스레드는 이전 스레드에서 액세스하는 캐시된 데이터를 제거할 수 있습니다. 즉, 각 스레드가 캐시를 '누락'하고 메모리에서 데이터를 가져와 캐시에서 데이터를 교체하는 실행 시간이 낭비됩니다. **perf** 도구를 사용하여 과도한 수의 캐시 누락이 있는지 확인합니다.

31.2.1. 시스템 토폴로지 표시

시스템의 토폴로지를 이해하는 데 도움이 되는 여러 명령이 있습니다. 다음 절차에서는 시스템 토폴로지를 결정하는 방법을 설명합니다.

절차

- 시스템 토폴로지의 개요를 표시하려면 다음을 수행합니다.

```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
[...]
```

- **CPU** 수, 스레드, 코어, 소켓 및 **NUMA** 노드 수와 같은 **CPU** 아키텍처에 대한 정보를 수집하려면 다음을 수행합니다.

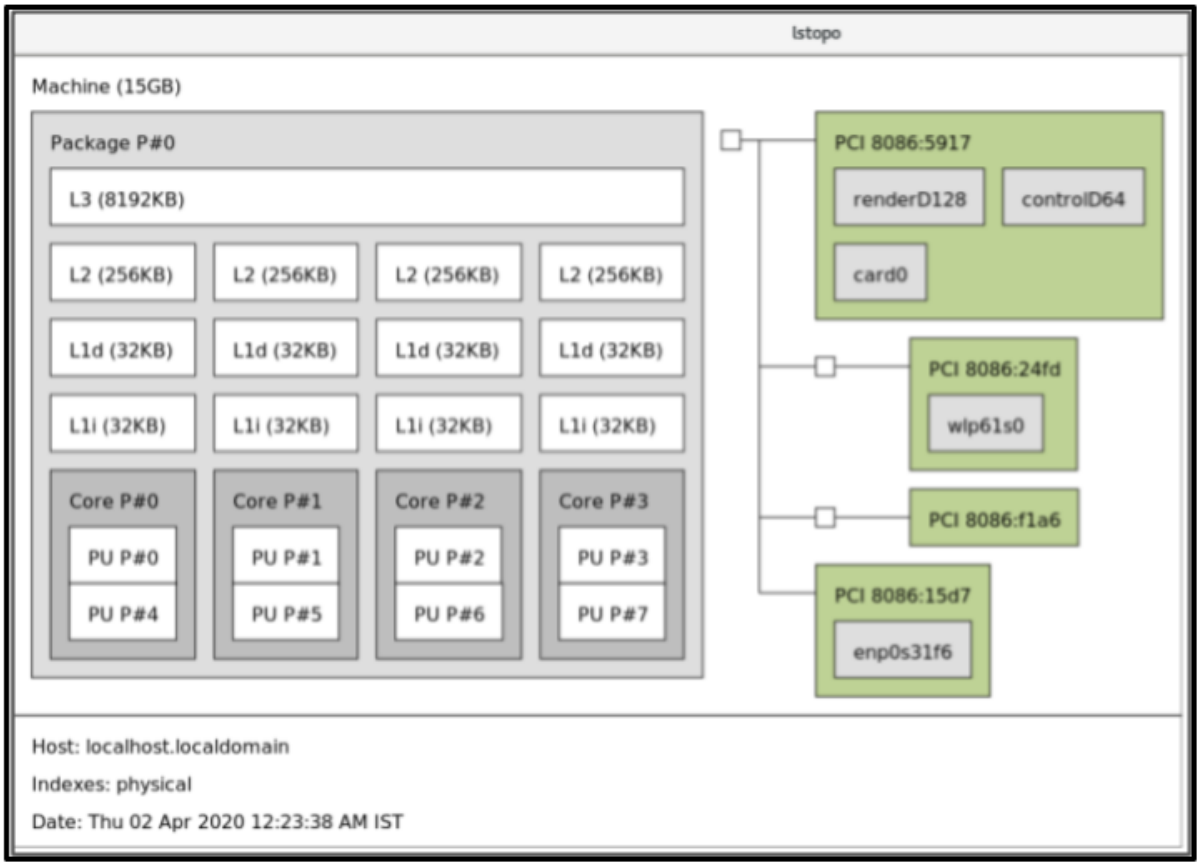
```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    10
Socket(s):             4
```

```
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 47
Model name:            Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:              2
CPU MHz:               2394.204
BogoMIPS:              4787.85
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0,4,8,12,16,20,24,28,32,36
NUMA node1 CPU(s):     2,6,10,14,18,22,26,30,34,38
NUMA node2 CPU(s):     1,5,9,13,17,21,25,29,33,37
NUMA node3 CPU(s):     3,7,11,15,19,23,27,31,35,39
```

- **시스템의 그래픽 표현을 보려면 다음을 수행합니다.**

```
# yum install hwloc-gui
# lstopo
```

그림 31.1. lstopo 출력



- **자세한 텍스트 출력을 보려면 다음을 수행합니다.**

```
# yum install hwloc
# lstopo-no-graphics
Machine (15GB)
  Package L#0 + L3 L#0 (8192KB)
    L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
      PU L#0 (P#0)
      PU L#1 (P#4)
     HostBridge L#0
    PCI 8086:5917
      GPU L#0 "renderD128"
      GPU L#1 "controlD64"
      GPU L#2 "card0"
    PCIBridge
      PCI 8086:24fd
        Net L#3 "wlp61s0"
    PCIBridge
      PCI 8086:f1a6
    PCI 8086:15d7
      Net L#4 "enp0s31f6"
```

추가 리소스

- **numactl(8), lscpu(1)** 및 **lstopo(1)** 도움말 페이지

31.3. 커널 틱 시간 설정

기본적으로 **Red Hat Enterprise Linux 8**은 전원 사용량을 줄이고 새 프로세서가 절전 상태를 활용할 수 있도록 유휴 **CPU**를 중단하지 않는 틱리스 커널을 사용합니다.

또한 **Red Hat Enterprise Linux 8**은 고성능 컴퓨팅 또는 실시간 컴퓨팅과 같은 대기 시간에 민감한 워크로드에 유용한 동적 틱리스 옵션을 제공합니다. 기본적으로 동적 틱리스 옵션은 비활성화되어 있습니다. **Red Hat**은 **cpu-partitioning TuneD** 프로필을 사용하여 **isolated_cores** 로 지정된 코어의 동적 틱리스 옵션을 활성화하는 것이 좋습니다.

이 절차에서는 동적 틱리스 동작을 수동으로 영구적으로 활성화하는 방법을 설명합니다.

절차

1. 특정 코어에서 동적 틱리스 동작을 사용하려면 **nohz_full** 매개 변수를 사용하여 커널 명령줄에서 해당 코어를 지정합니다. **16**코어 시스템에서 **/etc/default/grub** 파일의 **GRUB_CMDLINE_LINUX** 옵션에 이 매개변수를 추가합니다.

```
nohz_full=1-15
```

이를 통해 **1 ~15**코어에 대해 동적 틱리스 동작을 수행할 수 있어 모든 시간이 지정되지 않은 유일한 코어**(**코어 **0)**로 이동합니다**.**

2. 동적 틱리스 동작을 영구적으로 활성화하려면 편집한 기본 파일을 사용하여 **GRUB2** 설정을 다시 생성합니다**. BIOS** 펌웨어가 있는 시스템에서 다음 명령을 실행합니다**.**

```
# grub2-mkconfig -o /etc/grub2.cfg
```

UEFI 펌웨어가 있는 시스템에서 다음 명령을 실행합니다**.**

```
# grub2-mkconfig -o /etc/grub2-efi.cfg
```

3. 시스템이 부팅되면 **rcu** 스레드를 대기 시간이 중요하지 않은 코어로 직접 이동합니다**.**이 경우 코어 **0:**

```
# for i in `pgrep rcu[^c]` ; do taskset -pc 0 $i ; done
```

4. 선택 사항**:** 커널 명령줄에서 **isolcpus** 매개 변수를 사용하여 특정 코어를 사용자 공간 작업에서 격리합니다**.**

5. 선택 사항**:** 커널의 나중 **bdi-flush** 스레드의 **CPU** 선호도를 하우스키핑 코어로 설정합니다**.**

```
echo 1 > /sys/bus/workqueue/devices/writeback/cpumask
```

검증 단계

- 시스템이 재부팅되면 **if dynticks** 가 활성화되었는지 확인합니다**.**

```
# journalctl -xe | grep dynticks
Mar 15 18:34:54 rhel-server kernel: NO_HZ: Full dynticks CPUs: 1-15.
```

- 동적 틱리스 구성이 올바르게 작동하는지 확인합니다**.**

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
```

이 명령은 **CPU 1**에서 틱을 측정하고 **CPU 1**은 **3**초 동안 절전하도록 지시합니다**.**

- 기본 커널 타이머 구성은 일반 **CPU**에서 약 **3100**개의 틱을 보여줍니다.

```
# perf stat -C 0 -e irq_vectors:local_timer_entry taskset -c 0 sleep 3

 Performance counter stats for 'CPU(s) 0':

             3,107      irq_vectors:local_timer_entry

       3.001342790 seconds time elapsed
```

- 동적 틱리스 커널을 구성하면 대신 약 **4**개의 눈금이 표시됩니다.

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3

 Performance counter stats for 'CPU(s) 1':

                 4      irq_vectors:local_timer_entry

       3.001544078 seconds time elapsed
```

추가 리소스

- **perf(1)** 및 **cpuset(7)** 도움말 페이지
- 모두 **nohz_full** 커널 매개변수 **Red Hat Knowledgebase** 문서
- **sysfs**에서 **"isolated"** 및 **"nohz_full" CPU** 정보를 확인하는 방법은 무엇입니까**? Red Hat Knowledgebase** 문서

31.4. 인터럽트 요청 개요

인터럽트 요청 또는 **IRQ**는 하드웨어에서 프로세서로 전송되는 즉시 감시를 위한 신호입니다. 시스템의 각 장치에는 고유한 인터럽트를 보낼 수 있는 하나 이상의 **IRQ** 번호가 할당됩니다. 인터럽트가 활성화되면 인터럽트 요청을 수신하는 프로세서는 인터럽트 요청을 처리하기 위해 현재 애플리케이션 스레드의 실행을 즉시 일시 중지합니다.

인터럽트가 정상적인 작업을 중지하므로 높은 인터럽트 속도가 시스템 성능을 심각하게 저하시킬 수 있습니다. 인터럽트 선호도를 구성하거나 우선 순위가 낮은 인터럽트를 배치(여러 인터럽트 결합)로 전송하여 인터럽트에서 수행하는 시간을 줄일 수 있습니다.

인터럽트 요청에는 인터럽트 요청을 처리하는 프로세서를 정의하는 관련 선호도 속성인 **smp_affinity** 가 있습니다. 애플리케이션 성능을 개선하기 위해 동일한 프로세서 또는 동일한 코어의 프로세서에 인터럽트 선호도 및 프로세스 선호도를 할당합니다. 이렇게 하면 지정된 인터럽트 및 애플리케이션 스레드가 캐시 행을 공유할 수 있습니다.

인터럽트를 지원하는 시스템에서 인터럽트 요청의 **smp_affinity** 속성을 수정하여 특정 프로세서와 함께 인터럽트를 서비스하기로 한 결정은 커널의 개입 없이 하드웨어 수준에서 이루어집니다.

31.4.1. 인터럽트 수동 분산

BIOS에서 **NUMA** 토폴로지를 내보내는 경우 **irqbalance** 서비스는 하드웨어 요청 서비스로 로컬인 노드에서 인터럽트 요청을 자동으로 제공할 수 있습니다.

절차

1. 구성할 인터럽트 요청에 해당하는 장치를 확인합니다.

2. 플랫폼에 맞는 하드웨어 사양을 찾아보십시오. 시스템의 칩셋이 인터럽트 배포를 지원하는지 확인합니다.

   a. 이 경우 다음 단계에 설명된 대로 인터럽트 전달을 구성할 수 있습니다. 또한 칩셋에서 인터럽트의 균형을 조정하는 데 사용하는 알고리즘을 확인합니다. 일부 **BIOS**에는 인터럽트 전달을 구성하는 옵션이 있습니다.

   b. 그렇지 않으면 칩셋은 항상 모든 인터럽트를 하나의 정적 **CPU**로 라우팅합니다. 사용되는 **CPU**는 구성할 수 없습니다.

3. 시스템에서 사용 중인 고급 프로그래밍 가능한 인터럽트 컨트롤러 **(APIC)** 모드를 확인합니다.

```
$ journalctl --dmesg | grep APIC
```

   여기,

   - 시스템에서 **flat** 이외의 모드를 사용하는 경우 물리적 플랫 으로 **APIC** 라우팅 설정과 유사한 행을 확인할 수 있습니다.

- 이러한 메시지가 표시되지 않으면 시스템은 플랫 모드를 사용합니다.

  시스템이 **x2apic** 모드를 사용하는 경우 부트 로더 구성의 커널 명령줄에 **nox2apic** 옵션을 추가하여 비활성화할 수 있습니다.

  물리적이 아닌 플랫 모드(**flat**)만 여러 **CPU**에 인터럽트를 분산하는 기능을 지원합니다. 이 모드는 **CPU**가 최대 **8** 개인 시스템에만 사용할 수 있습니다.

4. **smp_affinity** 마스크 를 계산합니다. **smp_affinity** 마스크 를 계산하는 방법에 대한 자세한 내용은 **smp_affinity** 마스크 설정을 참조하십시오.

추가 리소스

- **journalctl(1)** 및 **taskset(1)** 도움말 페이지

31.4.2. smp_affinity 마스크 설정

smp_affinity 값은 시스템의 모든 프로세서를 나타내는 **16**진수 비트 마스크로 저장됩니다. 각 비트는 다른 **CPU**를 구성합니다. 가장 적은 유효 비트는 **CPU 0**입니다.

마스크의 기본값은 **f** 입니다. 즉, 시스템의 모든 프로세서에서 인터럽트 요청을 처리할 수 있습니다. 이 값을 **1**로 설정하면 프로세서 **0**만 인터럽트를 처리할 수 있습니다.

절차

1. 바이너리에서는 인터럽트를 처리하는 **CPU**에 대해 값 **1**을 사용합니다. 예를 들어 **CPU 0** 및 **CPU 7**을 인터럽트를 처리하도록 설정하려면 바이너리 코드로 **0000000010000001** 을 사용합니다.

   표 31.1. CPU의 바이너리 비트

CPU	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
바이너리	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

2. 바이너리 코드를 **16**진수로 변환합니다.

예를 들어 **Python**을 사용하여 바이너리 코드를 변환하려면 다음을 수행합니다.

```
>>> hex(int('0000000010000001', 2))

'0x81'
```

32개 프로세서가 넘는 시스템에서는 개별 **32**비트 그룹의 **smp_affinity** 값을 구분해야 합니다. 예를 들어 **64** 프로세서 시스템의 처음 **32** 프로세서만 인터럽트 요청을 서비스하려면 **0xffffff,00000000** 을 사용합니다.

3. 특정 인터럽트 요청의 인터럽트 선호도 값은 연결된 **/proc/irq/irq_number/smp_affinity** 파일에 저장됩니다. 이 파일에서 **smp_affinity** 마스크를 설정합니다.

```
# echo mask > /proc/irq/irq_number/smp_affinity
```

추가 리소스

- **journalctl(1), irqbalance(1)** 및 **taskset(1)** 도움말 페이지

32장. 스케줄링 정책 조정

Red Hat Enterprise Linux에서 프로세스 실행의 최소 단위는 스레드라고 합니다. 시스템 스케줄러는 스레드를 실행하는 프로세서와 스레드가 실행되는 시간을 결정합니다. 그러나 스케줄러의 주된 문제는 시스템을 계속 사용 상태로 유지하는 것이므로 애플리케이션 성능에 맞게 스레드를 최적으로 예약하지 못할 수 있습니다.

예를 들어 노드 B의 프로세서를 사용할 수 있게 되면 NUMA 시스템의 애플리케이션이 노드 A에서 실행 중이라고 가정합니다. 노드 B에서 프로세서를 계속 사용하려면 스케줄러가 애플리케이션 스레드 중 하나를 노드 B로 이동합니다. 그러나 애플리케이션 스레드는 여전히 노드 A의 메모리에 액세스해야 합니다. 그러나 이제 스레드가 Node B 및 Node A 메모리에서 실행 중이므로 이 메모리는 더 이상 스레드가 로컬에 있지 않으므로 이 메모리가 더 이상 액세스할 수 없습니다. 따라서 노드 B에서 실행을 완료하는 데 Node A의 프로세서가 사용 가능하게 될 때까지 기다린 다음 로컬 메모리 액세스 권한이 있는 원래 노드에서 스레드를 실행하는 데 시간이 오래 걸릴 수 있습니다.

32.1. 스케줄링 정책 범주

성능에 민감한 애플리케이션은 스레드가 실행되는 위치를 결정하는 디자이너나 관리자의 이점을 얻는 경우가 많습니다. Linux 스케줄러는 스레드 실행 시간을 결정하는 여러 스케줄링 정책을 구현합니다.

다음은 스케줄링 정책의 두 가지 주요 범주입니다.

일반 정책

일반 스레드는 일반 우선 순위 작업에 사용됩니다.

실시간 정책

실시간 정책은 중단 없이 완료해야 하는 시간에 민감한 작업에 사용됩니다. 실시간 스레드는 시간 분할의 대상이 아닙니다. 즉 스레드는 블록, 종료, 자발적으로 양보 또는 더 높은 우선 순위 스레드에 의해 선점될 때까지 실행됩니다.

가장 낮은 우선 순위 실시간 스레드는 일반 정책이 있는 스레드보다 먼저 예약됩니다. 자세한 내용은 SCHED_FIFO로 정적 우선 순위 스케줄링 및 SCHED_ RR로 라운드 로빈 우선 순위 스케줄링을 참조하십시오.

추가 리소스

- sched(7), sched_setaffinity(2), sched_getaffinity(2), sched_setscheduler(2) 및 sched_getscheduler(2) 도움말 페이지

32.2. SCHED_FIFO로 고정 우선 순위 스케줄링

SCHED_FIFO (정적 우선 순위 스케줄링이라고도 함)는 각 스레드에 대한 고정 우선 순위를 정의하는 실시간 정책입니다. 이 정책을 통해 관리자는 이벤트 응답 시간을 개선하고 대기 시간을 줄일 수 있습니다. 시간이 중요한 작업에 대해 이 정책을 장기간 실행하지 않는 것이 좋습니다.

SCHED_FIFO 가 사용 중인 경우 스케줄러는 우선 순위에 따라 모든 SCHED_FIFO 스레드 목록을 검사하고 실행할 준비가 된 가장 높은 우선 순위 스레드를 예약합니다. SCHED_FIFO 스레드의 우선 순위 수준은 1 에서 99 사이의 정수일 수 있습니다. 여기서 99 는 가장 높은 우선 순위로 간주됩니다. Red Hat 은 대기 시간 문제를 식별하는 경우에만 더 적은 수의 우선 순위로 시작하는 것이 좋습니다.

주의

실시간 스레드는 시간 분할 대상이 아니므로 Red Hat은 우선순위 99를 설정하는 것을 권장하지 않습니다. 이렇게 하면 프로세스가 마이그레이션 및 감시 스레드와 동일한 우선 순위 수준으로 유지됩니다. 스레드가 컴퓨팅 루프로 이동하고 이러한 스레드가 차단되면 실행할 수 없습니다. 단일 프로세서가 있는 시스템은 결국 이 상황이 중단됩니다.

관리자는 SCHED_FIFO 대역폭을 제한하여 실시간 애플리케이션 프로그래머가 프로세서를 독점하는 실시간 작업을 시작하지 못하도록 할 수 있습니다.

다음은 이 정책에서 사용되는 매개 변수의 일부입니다.

/proc/sys/kernel/sched_rt_period_us

이 매개 변수는 프로세서 대역폭의 100 %로 간주되는 기간을 마이크로 초 단위로 정의합니다. 기본값은 1000000s 또는 1초 입니다.

/proc/sys/kernel/sched_rt_runtime_us

이 매개 변수는 실시간 스레드 실행에 적합한 기간을 마이크로초 단위로 정의합니다. 기본값은 950000s 또는0.95 초 입니다.

32.3. SCHED_RR을 사용한 라운드 로빈 우선 순위 스케줄링

SCHED_RR 은 SCHED_FIFO 의 라운드 로빈 변형입니다. 이 정책은 여러 스레드를 동일한 우선 순위

수준에서 실행해야 하는 경우 유용합니다.

SCHED_FIFO 와 마찬가지로**SCHED_RR** 은 각 스레드의 고정 우선 순위를 정의하는 실시간 정책입니다. 스케줄러는 모든 **SCHED_RR** 스레드 목록을 우선 순위 순으로 스캔하고 실행할 준비가 된 가장 높은 우선 순위 스레드를 예약합니다. 그러나 **SCHED_FIFO** 와 달리 우선 순위가 동일한 스레드는 특정 시간 슬라이스 내에서 라운드 로빈 형식으로 예약됩니다.

/proc/sys/kernel/sched _rr_timeslice_ms 파일에서 **sched_rr _timeslice_ms** 커널 매개 변수를 사용하여 이 시간 슬라이스의 값을 밀리초 단위로 설정할 수 있습니다. 가장 낮은 값은 **1**밀리초입니다.

32.4. SCHED_OTHER로 일반 스케줄링

SCHED_OTHER 는 **Red Hat Enterprise Linux 8**의 기본 스케줄링 정책입니다. 이 정책은 **CFS(Completely Fair Scheduler)**를 사용하여 이 정책으로 예약된 모든 스레드에 대한 적절한 프로세서 액세스를 허용합니다. 이 정책은 많은 스레드가 있거나 시간 경과에 따른 스레드를 더 효율적으로 예약할 수 있기 때문에 데이터 처리량이 우선 순위인 경우 가장 유용합니다.

이 정책이 사용 중인 경우 스케줄러는 각 프로세스 스레드의 **niceness** 값에 부분적으로 따라 동적 우선 순위 목록을 생성합니다. 관리자는 프로세스의 **niceness** 값을 변경할 수 있지만 스케줄러의 동적 우선 순위 목록을 직접 변경할 수 없습니다.

32.5. 스케줄러 정책 설정

chrt 명령줄 툴을 사용하여 스케줄러 정책 및 우선 순위를 확인하고 조정합니다. 원하는 속성을 사용하여 새 프로세스를 시작하거나 실행 중인 프로세스의 속성을 변경할 수 있습니다. 런타임 시 정책을 설정하는 데도 사용할 수 있습니다.

절차

1. 활성 프로세스의 **PID(**프로세스 **ID)**를 확인합니다.

```
# ps
```

ps 명령과 함께 **--pid** 또는 **-p** 옵션을 사용하여 특정 **PID**의 세부 정보를 확인합니다.

2. 특정 프로세스의 스케줄링 정책**, PID** 및 우선 순위를 확인합니다.

```
# chrt -p 468
pid 468's current scheduling policy: SCHED_FIFO
pid 468's current scheduling priority: 85

# chrt -p 476
pid 476's current scheduling policy: SCHED_OTHER
pid 476's current scheduling priority: 0
```

여기서 ***468*** 및 ***476*** 은 프로세스의 **PID**입니다.

3. 프로세스의 스케줄링 정책을 설정합니다.

a. 예를 들어 **PID** ***1000*** 이 있는 프로세스를 우선 순위가 ***50*** 인 ***SCHED_FIFO*** 로 설정하려면 다음을 실행합니다.

```
# chrt -f -p 50 1000
```

b. 예를 들어 **PID**가 ***1000*** 인 프로세스를 우선 순위가 ***0*** 인 ***SCHED_OTHER*** 로 설정하려면 다음을 실행합니다.

```
# chrt -o -p 0 1000
```

c. 예를 들어 **PID** ***1000*** 이 있는 프로세스를 우선 순위가 ***10*** 인 ***SCHED_RR*** 로 설정하려면 다음을 수행합니다.

```
# chrt -r -p 10 1000
```

d. 특정 정책 및 우선 순위로 새 애플리케이션을 시작하려면 애플리케이션 이름을 지정합니다.

```
# chrt -f 36 /bin/my-app
```

추가 리소스

- **chrt(1)** 도움말 페이지
- **chrt** 명령의 정책 옵션

- 부팅 프로세스 중 서비스 우선 순위 변경

32.6. CHRT 명령의 정책 옵션

chrt 명령을 사용하여 프로세스의 스케줄링 정책을 보고 설정할 수 있습니다.

다음 표에서는 프로세스의 스케줄링 정책을 설정하는 데 사용할 수 있는 적절한 정책 옵션을 설명합니다.

표 32.1. chrt 명령의 정책 옵션

짧은 옵션	긴 옵션	설명
-f	**--fifo**	일정을 **SCHED_FIFO**로 설정
-o	**--other**	schedule을 **SCHED_OTHER**로 설정
-r	**--rr**	schedule을 **SCHED_RR**로 설정

32.7. 부팅 프로세스 중 서비스 우선 순위 변경

systemd 서비스를 사용하면 부팅 프로세스 중에 시작된 서비스에 대한 실시간 우선 순위를 설정할 수 있습니다. *장치 구성 지시문* 은 부팅 프로세스 중에 서비스의 우선 순위를 변경하는 데 사용됩니다.

부팅 프로세스 우선 순위 변경은 **service** 섹션에서 다음 지시문을 사용하여 수행합니다.

CPUSchedulingPolicy=

실행된 프로세스에 대한 **CPU** 스케줄링 정책을 설정합니다. 기타,**fifo** 및 **rr** 정책을 설정하는 데 사용됩니다.

CPUSchedulingPriority=

실행된 프로세스의 **CPU** 스케줄링 우선 순위를 설정합니다. 사용 가능한 우선 순위 범위는 선택한 **CPU** 스케줄링 정책에 따라 다릅니다. 실시간 스케줄링 정책의 경우 **1** (최저 우선 순위)에서 **99** (최고 우선 순위) 사이의 정수를 사용할 수 있습니다.

다음 절차에서는 **mcelog** 서비스를 사용하여 부팅 프로세스 중에 서비스의 우선 순위를 변경하는 방법을 설명합니다.

사전 요구 사항

1. **tuned** 패키지를 설치합니다.

```
# yum install tuned
```

2. **tuned** 서비스를 활성화하고 시작합니다.

```
# systemctl enable --now tuned
```

절차

1. 실행 중인 스레드의 스케줄링 우선 순위를 확인합니다.

```
# tuna --show_threads
                    thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
 1     OTHER    0    0xff      3181          292         systemd
 2     OTHER    0    0xff       254            0        kthreadd
 3     OTHER    0    0xff         2            0         rcu_gp
 4     OTHER    0    0xff         2            0      rcu_par_gp
 6     OTHER    0       0         9            0 kworker/0:0H-kblockd
 7     OTHER    0    0xff      1301            1 kworker/u16:0-events_unbound
 8     OTHER    0    0xff         2            0    mm_percpu_wq
 9     OTHER    0       0       266            0     ksoftirqd/0
[...]
```

2. 보조 **mcelog** 서비스 구성 디렉터리 파일을 생성하고 이 파일에 정책 이름 및 우선 순위를 삽입합니다.

```
# cat <<-EOF > /etc/systemd/system/mcelog.system.d/priority.conf

>
[SERVICE]
CPUSchedulingPolicy=_fifo_
CPUSchedulingPriority=_20_
EOF
```

3. **systemd** 스크립트 구성을 다시 로드합니다.

```
# systemctl daemon-reload
```

4. **mcelog** 서비스를 다시 시작하십시오.

```
# systemctl restart mcelog
```

검증 단계

- **systemd** 문제로 설정된 **mcelog** 우선 순위를 표시합니다.

```
# tuna -t mcelog -P
thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
826     FIFO    20  0,1,2,3        13             0          mcelog
```

추가 리소스

- **systemd(1)** 및 **tuna(8)** 도움말 페이지
- 우선순위 범위에 대한 설명

32.8. 우선순위 맵

우선순위는 그룹에 정의되며, 일부 그룹은 특정 커널 기능을 전용으로 합니다. 실시간 스케줄링 정책의 경우 **1** (최저 우선 순위)에서 **99** (최고 우선 순위) 사이의 정수를 사용할 수 있습니다.

다음 테이블에서는 프로세스의 스케줄링 정책을 설정하는 동안 사용할 수 있는 우선 순위 범위를 설명합니다.

표 **32.2.** 우선순위 범위에 대한 설명

우선 순위	스레드	설명
1	낮은 우선 순위 커널 스레드	일반적으로 이 우선 순위는 **SCHED_OTHER** 위에 있어야 하는 작업에 대해 예약됩니다.
2 - 49	사용 가능	일반적인 애플리케이션 우선순위에 사용되는 범위입니다.
50	기본 hard-IRQ 값	

우선 순위	스레드	설명
51 - 98	높은 우선 순위 스레드	주기적으로 실행하고 빠른 응답 시간이 있어야 하는 스레드에 대해 이 범위를 사용합니다. CPU 바인딩 스레드에 이 범위를 사용하지 마십시오. 인터럽트가 부족하기 때문입니다.
99	워치독 및 마이그레이션	가장 높은 우선 순위에서 실행해야 하는 시스템 스레드입니다.

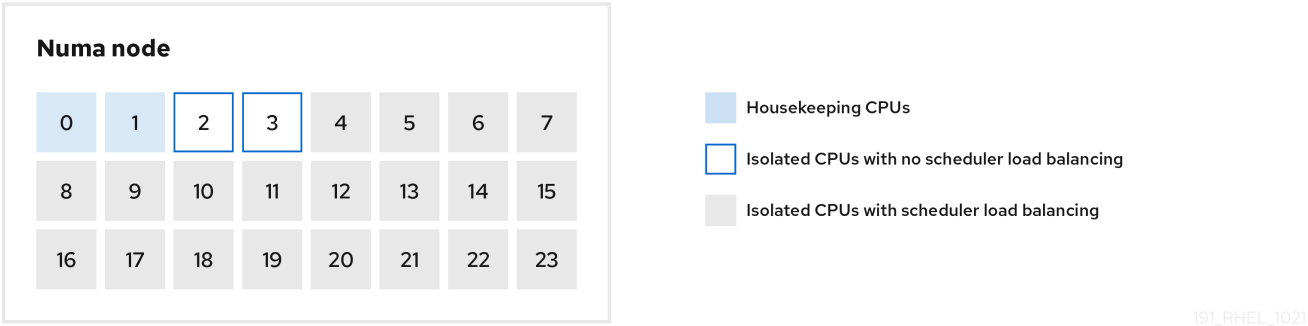
32.9. TUNED CPU-PARTITIONING 프로파일

대기 시간에 민감한 워크로드를 위해 **Red Hat Enterprise Linux 8**을 튜닝하려면 **cpu-partitioning TuneD** 프로필을 사용하는 것이 좋습니다.

Red Hat Enterprise Linux 8 이전에는 대기 시간이 짧은 **Red Hat** 설명서에서는 대기 시간이 짧은 튜닝을 달성하는 데 필요한 수많은 낮은 수준의 단계를 설명했습니다. **Red Hat Enterprise Linux 8**에서는 **cpu-partitioning TuneD** 프로필을 사용하여 대기 시간이 짧은 튜닝을 보다 효율적으로 수행할 수 있습니다. 이 프로필은 대기 시간이 짧은 개별 애플리케이션의 요구 사항에 따라 쉽게 사용자 지정할 수 있습니다.

다음 그림은 **cpu-partitioning** 프로필을 사용하는 방법을 보여주는 예입니다. 이 예에서는 **CPU** 및 노드 레이아웃을 사용합니다.

그림 **32.1.** 그림 **cpu-partitioning**



다음 설정 옵션을 사용하여 **/etc/tuned/cpu-partitioning-variables.conf** 파일에서 **cpu-partitioning** 프로필을 구성할 수 있습니다.

로드 밸런싱이 있는 분리된 **CPU**

cpu-partitioning 그림의 4에서 23으로 번호가 지정된 블록은 기본 분리된 CPU입니다. 커널 스케줄러의 프로세스 로드 밸런싱이 이러한 CPU에서 활성화됩니다. 커널 스케줄러 부하 분산이 필요한 여러 스레드가 있는 대기 시간이 짧은 프로세스를 위해 설계되었습니다.

isolated_cores=cpu-list 옵션을 사용하여 /etc/tuned/cpu-partitioning-variables.conf 파일에서 cpu- partitioning 프로필을 구성할 수 있습니다. 이 옵션은 커널 스케줄러 로드 밸런싱을 사용할 CPU를 격리합니다.

분리된 CPU 목록은 쉼표로 구분되거나 3-5 와 같은 대시를 사용하여 범위를 지정할 수 있습니다. 이 옵션은 필수입니다. 이 목록에서 누락된 CPU는 자동으로 하우스키핑 CPU로 간주됩니다.

로드 밸런싱이 없는 분리된 CPU

cpu-partitioning 그림의 경우 번호가 매겨진 2 및 3 블록은 추가 커널 스케줄러 프로세스 로드 밸런싱을 제공하지 않는 분리된 CPU입니다.

no_balance_cores=cpu-list 옵션을 사용하여 /etc/tuned/cpu-partitioning-variables.conf 파일에서 cpu- partitioning 프로필을 구성할 수 있습니다. 이 옵션은 커널 스케줄러 부하 분산을 사용하지 않는 CPU를 격리하도록 나열합니다.

no_balance_cores 옵션을 지정하는 것은 선택 사항이지만 이 목록의 모든 CPU는 isolated_cores 목록에 나열된 CPU의 서브 세트여야 합니다.

이러한 CPU를 사용하는 애플리케이션 스레드를 각 CPU에 개별적으로 고정해야 합니다.

하우스키핑 CPU

cpu-partitioning- Variabless.conf 파일에서 분리되지 않은 모든 CPU는 하우스키핑 CPU로 자동으로 간주됩니다. 하우스키핑 CPU에서는 모든 서비스, 데몬, 사용자 프로세스, 이동 가능한 커널 스레드, 인터럽트 핸들러 및 커널 타이머를 실행할 수 있습니다.

추가 리소스

- tuned-profiles-cpu-partitioning(7) 도움말 페이지

32.10. 대기 시간이 짧은 튜닝을 위해 TUNED CPU-PARTITIONING 프로필 사용

다음 절차에서는 TuneD의 cpu-partitioning 프로필을 사용하여 대기 시간이 짧은 시스템을 조정하는 방법을 설명합니다. cpu-partitioning 그림에 언급된 대로 cpu-partitioning 및 CPU 레이아웃을 사용할

수 있는 대기 시간이 짧은 애플리케이션의 예를 사용합니다.

이 경우 애플리케이션은 다음을 사용합니다.

- 네트워크에서 데이터를 읽는 전용 리더 스레드가 **CPU 2**에 고정됩니다.
- 이 네트워크 데이터를 처리하는 다수의 스레드가 **CPU 4-23**에 고정됩니다.
- 처리된 데이터를 네트워크에 쓰는 전용 작성기 스레드는 **CPU 3**에 고정됩니다.

사전 요구 사항

- **yum install tuned-profiles -cpu-partitioning** 명령을 **root**로 사용하여 **cpu -partitioning TuneD** 프로필을 설치했습니다.

절차

1. **/etc/tuned/cpu-partitioning-variables.conf** 파일을 편집하고 다음 정보를 추가합니다.

```
# Isolated CPUs with the kernel's scheduler load balancing:
isolated_cores=2-23
# Isolated CPUs without the kernel's scheduler load balancing:
no_balance_cores=2,3
```

2. **cpu-partitioning TuneD** 프로필을 설정합니다.

```
# tuned-adm profile cpu-partitioning
```

3. 재부팅

재부팅 후 **cpu-partitioning** 그림의 격리에 따라 시스템이 대기 시간이 짧아지도록 조정됩니다. 이 애플리케이션은 **taskset**을 사용하여 **reader** 및 **writer** 스레드를 **CPU 2** 및 **3**에 고정하고 나머지 애플리케이션 스레드는 **CPU 4-23**에 고정할 수 있습니다.

추가 리소스

- **tuned-profiles-cpu-partitioning(7)** 도움말 페이지

32.11. CPU-PARTITIONING TUNED 프로필 사용자 정의

TuneD 프로필을 확장하여 추가 튜닝을 변경할 수 있습니다.

예를 들어 **cpu-partitioning** 프로필은 **CPU**를 **cstate=1** 으로 설정합니다. **cpu-partitioning** 프로필을 사용하지만 **CPU cstate1**을 **cstate0**으로 추가로 변경하기 위해 다음 절차에서는 **cpu-partitioning** 프로필을 상속한 다음 **C state-0**을 설정하는 ***my_profile*** 이라는 새 **TuneD** 프로필을 설명합니다.

절차

1. **/etc/tuned/my_profile** 디렉토리를 생성합니다.

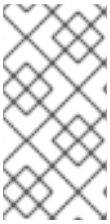
```
# mkdir /etc/tuned/my_profile
```

2. 이 디렉터리에 **tuned.conf** 파일을 생성하고 다음 내용을 추가합니다.

```
# vi /etc/tuned/my_profile/tuned.conf
[main]
summary=Customized tuning on top of cpu-partitioning
include=cpu-partitioning
[cpu]
force_latency=cstate.id:0|1
```

3. 새 프로필을 사용합니다.

```
# tuned-adm profile my_profile
```



참고

공유 예에서는 재부팅이 필요하지 않습니다. 그러나 ***my_profile*** 프로필의 변경 사항을 적용하려면 시스템을 재부팅해야 합니다.

추가 리소스

- **tuned-profiles-cpu-partitioning(7)** 도움말 페이지

33장. I/O 및 파일 시스템 성능에 영향을 주는 요인

스토리지 및 파일 시스템 성능에 대한 적절한 설정은 스토리지 목적에 따라 크게 달라집니다.

I/O 및 파일 시스템 성능은 다음 요인의 영향을 받을 수 있습니다.

- 데이터 쓰기 또는 읽기 패턴
- 순차적 또는 임의
- 버퍼링 또는 직접 IO
- 기본 표시와 데이터 정렬
- 블록 크기
- 파일 시스템 크기
- 저널 크기 및 위치
- 액세스 시간 기록
- 데이터 신뢰성 보장
- 데이터 사전 가져오기
- 디스크 공간 사전 할당

- 파일 조각화
- 리소스 경합

33.1. I/O 및 파일 시스템 문제 모니터링 및 진단 툴

시스템 성능을 모니터링하고 I/O, 파일 시스템 및 해당 구성과 관련된 성능 문제를 진단하기 위해 Red Hat Enterprise Linux 8에서 다음 도구를 사용할 수 있습니다.

- **vmstat** 툴은 전체 시스템에서 프로세스, 메모리, 페이징, 블록 I/O, 인터럽트 및 CPU 활동에 대해 보고합니다. 이는 관리자가 I/O 하위 시스템이 성능 문제를 책임지고 있는지 여부를 결정하는 데 도움이 될 수 있습니다. **vmstat** 를 사용한 분석에서 I/O 하위 시스템에서 성능이 저하된 것으로 표시되면 관리자가 **iostat** 도구를 사용하여 담당 I/O 장치를 확인할 수 있습니다.
- **iostat** 는 시스템의 I/O 장치 부하를 보고합니다. **sysstat** 패키지에서 제공합니다.
- **blktrace** 는 I/O 하위 시스템에서 시간을 소비하는 방법에 대한 자세한 정보를 제공합니다. 병행 유틸리티 **blkparse** 는 **blktrace**의 원시 출력을 읽고 **blktrace** 에서 기록한 입력 및 출력 작업의 요약을 사람이 읽을 수 있습니다 .
- **BTT**는 **blktrace** 출력을 분석하고 데이터가 I/O 스택의 각 영역에 소비되는 시간을 표시하여 I/O 하위 시스템에서 병목 현상을 더 쉽게 식별할 수 있도록 합니다. 이 유틸리티는 **blktrace** 패키지의 일부로 제공됩니다. **blktrace** 메커니즘에서 추적하고 **btt** 에 의해 분석되는 몇 가지 중요한 이벤트는 다음과 같습니다.
  - I/O 이벤트(Q)의 대기열
  - I/O를 드라이버 이벤트(D)로 디스패치
  - I/O 이벤트 완료(C)
- **iowatcher** 는 **blktrace** 출력을 사용하여 시간 경과에 따라 I/O를 그래프로 표시할 수 있습니다. 디스크 I/O의 논리 블록 주소(LBA), 초당 메가바이트 단위 처리량, 초당 검색 수, 초당 I/O 작업 수에 중점을 둡니다. 그러면 장치의 1초당 한도에 도달하는 시기를 식별하는 데 도움이 됩니다.

- BCC(BPF Compiler Collection)는eBPF(extended Berkeley Packet Filter) 프로그램을 쉽게 생성할 수 있는 라이브러리입니다. eBPF 프로그램은 디스크 I/O, TCP 연결 및 프로세스 생성과 같은 이벤트에서 트리거됩니다. BCC 툴은 /usr/share/bcc/tools/ 디렉토리에 설치됩니다. 다음 bcc-tools 는 성능을 분석하는 데 도움이 됩니다.

  - biolatency 는 히스토그램의 블록 장치 I/O(디스크 I/O)의 대기 시간을 요약합니다. 이를 통해 장치 캐시 적중 및 캐시 누락에 대한 두 가지 모드와 대기 시간 이전 모드를 포함하여 배포를 연구할 수 있습니다.

  - biosnoop 는 발급 프로세스 ID 및 I/O 대기 시간과 함께 각 I/O 이벤트를 표시하는 기본 블록 I/O 추적 툴입니다. 이 툴을 사용하여 디스크 I/O 성능 문제를 조사할 수 있습니다.

  - biotop 은 커널의 블록 i/o 작업에 사용됩니다.

  - filelife 도구는 stat() syscall을 추적합니다.

  - fileslower 는 느린 동기 파일 읽기 및 쓰기를 추적합니다.

  - filetop 는 파일 읽기 및 프로세스별 쓰기를 표시합니다.

  - ext4slower,nfsslower 및 xfsslower 는 파일 시스템 작업이 특정 임계값보다 느리게 표시하는 도구이며 기본값은 10ms 입니다.

    자세한 내용은 BPF Compiler Collection을 사용하여 시스템 성능 분석 에서 참조하십시오.

- bpftace 는 성능 문제를 분석하는 데 사용되는 eBPF 의 추적 언어입니다. 또한 시스템 관찰을 위해 BCC와 같은 추적 유틸리티를 제공하므로 I/O 성능 문제를 조사하는 데 유용합니다.

- 다음 SystemTap 스크립트는 스토리지 또는 파일 시스템 성능 문제를 진단하는 데 유용할 수 있습니다.

  - disktop.stp: 5초마다 디스크를 읽거나 쓰는 상태를 확인하고 해당 기간 동안 상위 10개 항목을 출력합니다.

- **iotime.stp:** 읽기 및 쓰기 작업에 사용된 시간과 읽기 및 작성된 바이트 수를 표시합니다.
- **traceio.stp:** 1초마다 관찰된 누적 I/O 트래픽을 기반으로 상위 10개의 실행 파일을 인쇄합니다.
- **traceio2.stp:** 실행 가능한 이름과 프로세스 식별자를 지정된 장치에 대한 읽기 및 쓰기로 인쇄합니다.
- **inodewatch.stp:** 지정된 주 또는 부 장치의 지정된 inode에 읽기 또는 쓰기가 발생할 때마다 실행 파일 이름 및 프로세스 식별자를 인쇄합니다.
- **inodewatch2.stp:** 지정된 주 또는 부 장치의 지정된 inode에서 속성이 변경될 때마다 실행 파일 이름, 프로세스 식별자 및 속성을 출력합니다.

추가 리소스

- **vmstat(8), iostat(1), blktrace(8), blkparse(1), btt(1), bpftrace, and iowatcher(1)** 도움말 페이지
- BPF Compiler Collection을 사용하여 시스템 성능 분석

33.2. 파일 시스템을 포맷하는 데 사용 가능한 튜닝 옵션

장치를 포맷한 후에는 일부 파일 시스템 구성 결정을 변경할 수 없습니다.

다음은 스토리지 장치를 포맷하기 전에 사용할 수 있는 옵션입니다.

크기

워크로드에 맞게 적절한 크기의 파일 시스템을 만듭니다. 작은 파일 시스템은 파일 시스템 점검에 필요한 시간과 메모리가 줄어듭니다. 그러나 파일 시스템이 너무 작으면 성능이 높은 조각화로 인한 영향을 받습니다.

블록 크기

블록은 파일 시스템의 작업 단위입니다. 블록 크기는 단일 블록에 저장할 수 있는 데이터 양을 결정하므로 한 번에 쓰거나 읽는 데이터 양을 결정합니다.

기본 블록 크기는 대부분의 사용 사례에 적합합니다. 그러나 파일 시스템은 블록 크기 또는 여러 블록의 크기가 한 번에 일반적으로 읽거나 쓰는 데이터의 양과 약간 큰 경우 데이터를 효율적으로 저장하고 데이터를 효율적으로 저장합니다. 작은 파일은 여전히 전체 블록을 사용합니다. 파일은 여러 블록에 분산될 수 있지만, 추가 런타임 오버헤드가 발생할 수 있습니다.

또한 일부 파일 시스템은 특정 블록 수로 제한되며, 이 경우 파일 시스템의 최대 크기를 제한합니다. 블록 크기는 **mkfs** 명령으로 장치를 포맷할 때 파일 시스템 옵션의 일부로 지정됩니다. 블록 크기를 지정하는 매개 변수는 파일 시스템에 따라 다릅니다.

마케도니아

파일 시스템은 파일 시스템 전체의 데이터 배포와 관련이 있습니다. 시스템이 **RAID**와 같은 스트라이핑된 스토리지를 사용하는 경우 장치를 포맷할 때 데이터와 메타데이터를 기본 스토리지에 정렬하여 성능을 향상시킬 수 있습니다.

많은 장치는 권장 사항을 내보내며, 이는 장치가 특정 파일 시스템을 사용하여 포맷될 때 자동으로 설정됩니다. 장치에서 이러한 권장 사항을 내보내지 않거나 권장 설정을 변경하려면 **mkfs** 명령을 사용하여 장치를 포맷할 때 수동으로 지정해야 합니다.

파일 시스템을 지정하는 매개 변수는 파일 시스템에 따라 다릅니다.

외부 저널

저널링 파일 시스템은 작업을 실행하기 전에 저널 파일의 쓰기 작업 중 수행할 변경 사항을 기록합니다. 이렇게 하면 시스템 충돌 또는 정전 시 스토리지 장치가 손상될 가능성이 줄어들고 복구 프로세스가 빨라집니다.

참고

외부 저널 옵션을 사용하지 않는 것이 좋습니다.

메타데이터를 많이 사용하는 워크로드에는 저널을 자주 업데이트해야 합니다. 더 큰 저널은 더 많은 메모리를 사용하지만 쓰기 작업의 빈도를 줄입니다. 또한 기본 스토리지보다 빠르거나 빠른 전용 스토리지에 저널을 배치하여 메타데이터를 많이 사용하는 워크로드가 있는 장치의 검색 시간을 개선할 수 있습니다.

주의

외부 저널이 신뢰할 수 있는지 확인합니다. 외부 저널 장치가 손실되면 파일 시스템이 손상됩니다. 외부 저널은 마운트 시 저널 장치를 지정하고 포맷 시 생성해야 합니다.

추가 리소스

- **mkfs(8)** 및 **mount(8)** 도움말 페이지
- 사용 가능한 파일 시스템 개요

33.3. 파일 시스템 마운트에 사용 가능한 튜닝 옵션

다음은 대부분의 파일 시스템에서 사용할 수 있는 옵션이며 장치가 마운트되어 있으므로 지정할 수 있습니다.

액세스 시간

파일을 읽을 때마다 해당 메타데이터는 액세스가 발생한 시간(시간)으로 업데이트됩니다. 여기에는 추가 쓰기 **I/O**가 포함됩니다. **relatime** 은 대부분의 파일 시스템의 기본 **A time** 설정입니다.

그러나 이 메타데이터를 업데이트하는 데 시간이 오래 걸리고 정확한 액세스 시간 데이터가 필요하지 않은 경우 **noatime** 마운트 옵션을 사용하여 파일 시스템을 마운트할 수 있습니다. 파일을 읽을 때 메타데이터 업데이트를 비활성화합니다. 또한 **nodiratime** 동작을 활성화하여 디렉터리를 읽을 때 메타데이터 업데이트를 비활성화합니다.

참고

noa time 마운트 옵션을 사용하여 **a time** 업데이트를 비활성화하면 백업 프로그램(예: 백업 프로그램)에 의존하는 애플리케이션이 중단될 수 있습니다.

읽기**-ahead**

읽기-헤드 동작은 곧 필요할 수 있는 사전 가져오기 데이터로 파일 액세스를 가속화하고 페이지 캐시에 로드할 수 있습니다. 이 경우 디스크에 있는 것보다 더 빠르게 검색할 수 있습니다. 읽기**-ahead**

값이 클수록 시스템에서 데이터를 미리 준비하는 것이 더 높습니다.

Red Hat Enterprise Linux는 파일 시스템에 대해 감지한 내용에 따라 적절한 읽기**-ahead** 값을 설정하려고 합니다. 그러나 정확한 탐지가 항상 가능한 것은 아닙니다. 예를 들어 스토리지 어레이가 단일 **LUN**으로 시스템에 자체적으로 제공되는 경우 시스템은 단일 **LUN**을 감지하고 배열에 적절한 읽기**-ahead** 값을 설정하지 않습니다.

순차적 **I/O** 스트리밍이 많은 업무 부하가 높은 읽기**-**면치 값을 활용하는 경우가 많습니다. **Red Hat Enterprise Linux**와 함께 제공되는 스토리지 관련 **tuned** 프로필은 **LVM** 스트라이핑을 사용하는 것과 마찬가지로 읽기 헤드 값을 높이지만 이러한 조정만으로는 모든 워크로드에 충분하지는 않습니다.

추가 리소스

- **mount(8), xfs(5)** 및 **ext4(5)** 도움말 페이지

33.4. 사용하지 않는 블록을 폐기하는 유형

파일 시스템에서 사용하지 않는 정기적으로 블록을 삭제하는 것은 솔리드 스테이트 디스크와 씬 프로비저닝 스토리지 모두에 권장되는 방법입니다.

다음은 사용되지 않는 블록을 폐기하는 두 가지 방법입니다.

배치 삭제

이 유형의 삭제는 **fstrim** 명령의 일부입니다. 관리자가 지정한 기준과 일치하는 파일 시스템에서 사용되지 않은 모든 블록을 폐기합니다. **Red Hat Enterprise Linux 8**은 물리적 삭제 작업을 지원하는 **XFS** 및 **ext4** 형식의 장치에서 배치 삭제를 지원합니다.

온라인 삭제

이러한 유형의 삭제 작업은 **discard** 옵션을 사용하여 마운트 시 구성되며 사용자 개입 없이 실시간으로 실행됩니다. 그러나 가용에서 무료로 전환되는 블록만 폐기합니다. **Red Hat Enterprise Linux 8**은 **XFS** 및 **ext4** 형식의 장치에서 온라인 삭제를 지원합니다.

Red Hat은 성능을 유지하기 위해 온라인 삭제가 필요한 위치 또는 시스템 워크로드에 배치 삭제가 불가능한 위치를 제외하고 배치 삭제를 권장합니다.

사전 할당은 해당 공간에 데이터를 쓰지 않고 디스크에 할당된 것으로 표시합니다. 이 기능은 데이터 조

각화 및 읽기 성능이 저하되는 데 유용할 수 있습니다. Red Hat Enterprise Linux 8은 XFS, ext4 및 GFS2 파일 시스템의 사전 할당 공간을 지원합니다. 또한 애플리케이션은 fallocate(2) glibc 호출을 사용하여 공간을 미리 할당하여 이점을 얻을 수 있습니다.

추가 리소스

- mount(8) 및 fallocate(2) 도움말 페이지

33.5. 솔리드 스테이트 디스크 튜닝 고려 사항

SSD(반도체 디스크)는 영구 데이터를 저장하기 위해 자기 플래터를 회전하는 대신 NAND 플래시 칩을 사용합니다. SSD는 전체 논리 블록 주소 범위에서 데이터에 대한 지속적인 액세스 시간을 제공하며 순환과 같은 비용을 측정할 수 없습니다. 스토리지 공간당 1기가바이트당 비용이 더 높으며 스토리지 밀도가 더 적지만 대기 시간이 짧고 HDD보다 처리량이 높습니다.

SSD에서 사용된 블록이 디스크 용량에 접근하므로 일반적으로 성능이 저하됩니다. 성능 저하 수준은 벤더에 따라 다르지만 모든 장치는 이러한 상황에서 성능이 저하됩니다. 삭제 동작을 활성화하면 이러한 저하를 완화하는 데 도움이 될 수 있습니다. 자세한 내용은 사용하지 않는 블록 삭제 유형을 참조하십시오.

기본 I/O 스케줄러와 가상 메모리 옵션은 SSD와 함께 사용하기에 적합합니다. SSD 성능에 영향을 줄 수 있는 설정을 구성할 때 다음 요소를 고려하십시오.

I/O 스케줄러

모든 I/O 스케줄러는 대부분의 SSD에서 제대로 수행되어야 합니다. 그러나 다른 스토리지 유형과 마찬가지로 Red Hat은 벤치마킹을 통해 지정된 워크로드에 대한 최적의 구성을 결정하는 것이 좋습니다. SSD를 사용하는 경우 Red Hat은 특정 워크로드 벤치마킹에 대해서만 I/O 스케줄러를 변경하는 것을 권장합니다. I/O 스케줄러 간 전환 방법에 대한 자세한 내용은 /usr/share/doc/kernel-version/Documentation/block/switching-sched.txt 파일을 참조하십시오.

단일 대기열 HBA의 경우 기본 I/O 스케줄러는 데드라인 입니다. 다중 대기열 HBA의 경우 기본 I/O 스케줄러는 none 입니다. I/O 스케줄러를 설정하는 방법에 대한 자세한 내용은 디스크 스케줄러 설정을 참조하십시오.

가상 메모리

I/O 스케줄러와 마찬가지로 VM(가상 메모리) 하위 시스템에는 특별한 튜닝이 필요하지 않습니다. SSD에서 I/O의 빠른 특성을 고려할 때 vm_dirty_background_ratio 및 vm_dirty_ratio 설정을 끄십시오. 증가한 쓰기 활동은 일반적으로 디스크에서 다른 작업 대기 시간에 부정적인 영향을 미치지 않습니다. 그러나 이 튜닝은 전반적인 I/O를 생성할 수 있으므로 일반적으로 워크로드별 테스트 없이는 권장되지 않습니다.

스왑

SSD는 스왑 장치로 사용할 수도 있으며 좋은 페이지 아웃 및 페이지 축소 성능을 생성할 수 있습니다.

33.6. 일반 블록 장치 튜닝 매개변수

이 섹션에 나열된 일반 튜닝 매개 변수는 /sys/block/sdX/queue/ 디렉터리에서 사용할 수 있습니다.

나열된 다음 튜닝 매개 변수는 I/O 스케줄러 튜닝과 별개이며 모든 I/O 스케줄러에 적용됩니다.

add_random

일부 I/O 이벤트는 /dev/random 의 엔트로피 풀에 기여합니다. 이러한 기여의 오버헤드를 측정할 수 있는 경우 이 매개변수를 0 으로 설정할 수 있습니다.

iostats

기본적으로 iostats 는 활성화되고 기본값은 1 입니다. iostats 값을 0 으로 설정하면 장치에 대한 I/O 통계 수집이 비활성화되어 I/O 경로가 있는 적은 양의 오버헤드가 제거됩니다. iostats 를 0 으로 설정하면 특정 NVMe 솔리드 스테이트 스토리지 장치와 같이 고성능 장치의 성능이 약간 향상될 수 있습니다. 벤더가 지정된 스토리지 모델에 달리 지정하지 않는 한 iostats 를 활성화하는 것이 좋습니다.

iostats 를 비활성화하면 /proc/diskstats 파일에 더 이상 장치의 I/ O 통계가 표시되지 않습니다. /sys/diskstats 파일의 내용은 sar 또는 iostats와 같은 I/O 툴 모니터링을 위한 I/O 정보의 소스입니다. 따라서 장치에 대한 iostats 매개변수를 비활성화하면 더 이상 장치가 I/O 모니터링 툴 출력에 표시되지 않습니다.

max_sectors_kb

I/O 요청의 최대 크기(KB)를 지정합니다. 기본값은 512 KB입니다. 이 매개 변수의 최소 값은 스토리지 장치의 논리적 블록 크기에 따라 결정됩니다. 이 매개변수의 최대 값은 max_hw_sectors_kb 값에 따라 결정됩니다.

max_sectors_kb 는 항상 최적의 I/O 크기 및 내부 삭제 블록 크기의 배수가 되도록 권장합니다. 두 매개 변수 중 하나가 스토리지 장치에서 지정되지 않은 경우 매개 변수에 logical_block_size 값을 사용합니다.

nomerges

대부분의 워크로드는 요청 병합으로 이점을 제공합니다. 그러나 병합을 비활성화하면 디버깅에 유용할 수 있습니다. 기본적으로 nomerges 매개 변수는 병합을 활성화하는 0 으로 설정됩니다. 간단

한 일회성 병합을 비활성화하려면 **nomerges** 를 **1** 로 설정합니다. 모든 유형의 병합을 비활성화하려면 **nomerges** 를 **2** 로 설정합니다.

nr_requests

대기 중인 **I/O**에서 허용되는 최대 수입니다. 현재 **I/O** 스케줄러가 없는 경우 이 수는 줄일 수 있습니다. 그렇지 않으면 숫자를 늘리거나 줄일 수 있습니다.

optimal_io_size

일부 스토리지 장치는 이 매개 변수를 통해 최적의 **I/O** 크기를 보고합니다. 이 값이 보고되면 **Red Hat**은 애플리케이션이 가능한 최적의 **I/O** 크기의 배수로 **I/O**를 실행할 것을 권장합니다.

read_ahead_kb

순차적 읽기 작업 중에 운영 체제가 미리 읽을 수 있는 최대 킬로바이트 수를 정의합니다. 결과적으로 다음 순차 읽기를 위해 커널 페이지 캐시 내에 필요한 정보가 이미 있으므로 읽기 **I/O** 성능이 향상됩니다.

장치 매퍼가 높은 **read_ahead_kb** 값을 활용하는 경우가 많습니다. 매핑할 각 장치에 대한 **128 KB**는 좋은 시작 지점이지만, 디스크의 **read_aheads_kb** 값을 요청하기 위해 최대 **128KB**의 용량이 큰 파일을 순차적으로 읽는 애플리케이션 환경에서 성능이 향상될 수 있습니다.

rotational

일부 솔리드 스테이트 디스크는 솔리드 상태를 올바르게 알리지 않으며 기존 회전 디스크로 마운트됩니다. 스케줄러에서 불필요한 **find-reducing** 논리를 비활성화하려면 수동으로 **rotational** 값을 **0** 으로 설정합니다.

rq_affinity

rq_affinity 의 기본값은 **1** 입니다. 발행된 **CPU** 코어의 동일한 **CPU** 그룹에 있는 하나의 **CPU** 코어에서 **I/O** 작업을 완료합니다. **I/O** 요청을 발행한 프로세서에서만 완료를 수행하려면 **rq_affinity** 를 **2** 로 설정합니다. 언급된 두 가지 기능을 비활성화하려면 **0** 으로 설정합니다.

scheduler

특정 스토리지 장치에 대한 스케줄러 또는 스케줄러 기본 설정 순서를 설정하려면 **/sys/block/*devname*/queue/scheduler** 파일을 편집합니다. 여기서 ***devname*** 은 구성하려는 장치의 이름입니다.

34장. 네트워크 리소스에 대한 액세스를 최적화하도록 운영 체제 구성

이 섹션에서는 워크로드 전반에서 네트워크 리소스에 대한 최적화된 액세스를 제공하도록 운영 체제를 구성하는 방법에 대해 설명합니다. 네트워크 성능 문제는 하드웨어 오작동 또는 결함이 있는 인프라의 원인이 되는 경우가 있습니다. 이러한 문제를 해결하는 것은 이 문서의 범위를 벗어납니다.

TuneD 서비스는 다양한 특정 사용 사례에서 성능을 향상시키는 다양한 프로필을 제공합니다.

- **latency-performance**
- **network-latency**
- **network-throughput**

34.1. 성능 문제 모니터링 및 진단 툴

다음은 시스템 성능을 모니터링하고 네트워킹 하위 시스템과 관련된 성능 문제를 진단하는 데 사용되는 **Red Hat Enterprise Linux 8**에서 사용 가능한 도구입니다.

- **SS** 유틸리티는 소켓에 대한 통계 정보를 인쇄하여 관리자가 시간 경과에 따른 장치 성능을 평가할 수 있도록 합니다. 기본적으로 **ss** 는 설정된 연결에서 수신되지 않은 **TCP** 소켓을 표시합니다. 관리자는 명령줄 옵션을 사용하여 특정 소켓에 대한 통계를 필터링할 수 있습니다. **Red Hat Enterprise Linux**에서 더 이상 사용되지 않는 **netstat** 에 대해 **ss** 를 권장합니다.
- **IP** 유틸리티를 사용하면 관리자가 경로, 장치, 라우팅 정책 및 터널을 관리하고 모니터링할 수 있습니다. **ip monitor** 명령은 장치, 주소 및 경로의 상태를 지속적으로 모니터링할 수 있습니다. **j** 옵션을 사용하여 출력을 **JSON** 형식으로 표시합니다. **JSON** 형식은 정보 처리를 자동화하기 위해 다른 유틸리티에 추가로 제공될 수 있습니다.
- **dropwatch** 는 **dropwatch** 패키지에서 제공하는 대화형 도구입니다. 커널에서 삭제한 패킷을 모니터링하고 기록합니다.
- **ethtool** 유틸리티를 사용하면 관리자가 네트워크 인터페이스 카드 설정을 보고 편집할 수 있습니다. 이 도구를 사용하여 특정 장치의 해당 장치에서 삭제한 패킷 수와 같은 통계를 관찰합니

다. **ethtool -S *device name*** 명령을 사용하여 모니터링할 장치의 지정된 장치의 카운터 상태를 확인합니다.

- **/proc/net/snmp** 파일은 **the snmp** 에이전트가 **IP, ICMP, TCP, UDP** 모니터링 및 관리에 사용하는 데이터를 표시합니다. 정기적으로 이 파일을 검사하면 관리자가 비정상적인 값을 식별하여 잠재적인 성능 문제를 식별할 수 있습니다. 예를 들어 **/proc/net/snmp** 파일의 **UDP** 입력 오류(오류)증가는 소켓 수신 큐의 성능 장애를 나타낼 수 있습니다.

- **nstat** 도구는 커널 **SNMP** 및 네트워크 인터페이스 통계를 모니터링합니다. 이 도구는 **/proc/net/snmp** 파일에서 데이터를 읽고 사람이 읽을 수 있는 형식으로 정보를 인쇄합니다.

- 기본적으로 **systemtap-client** 패키지에서 제공하는 **SystemTap** 스크립트는 **/usr/share/systemtap/examples/network** 디렉터리에 설치됩니다.

  - **nettop.stp: 5**초마다 스크립트는 전송 및 수신된 패킷 수와 해당 간격 동안 프로세스에서 보내고 받는 데이터 양과 프로세스에서 보낸 데이터(프로세스 식별자 및 명령) 목록을 표시합니다.

  - **socket-trace.stp: Linux** 커널의 **net/socket.c** 파일에서 각 기능을 결합하고 추적 데이터를 표시합니다.

  - **dropwatch.stp: 5**초마다 스크립트는 커널의 위치에서 사용 가능한 소켓 버퍼 수를 표시합니다. **all -modules** 옵션을 사용하여 심볼릭 이름을 확인합니다.

  - **latencytap.stp:** 이 스크립트는 하나 이상의 프로세스에 다양한 유형의 대기 시간이 미치는 영향을 기록합니다. 프로세스 또는 프로세스에서 대기하는 총 시간별로 **30**초마다 내림차순으로 정렬된 대기 시간 유형의 목록을 인쇄합니다. 이는 스토리지 및 네트워크 대기 시간의 원인을 식별하는 데 유용할 수 있습니다.

  대기 시간 이벤트의 매핑을 보다 효과적으로 지원하기 위해 이 스크립트와 함께 **--all-modules** 옵션을 사용하는 것이 좋습니다. 기본적으로 이 스크립트는 **/usr/share/systemtap/examples/profiling** 디렉터리에 설치됩니다.

- **BCC(BPF Compiler Collection)**는**eBPF(extended Berkeley Packet Filter)** 프로그램을 쉽게 생성할 수 있는 라이브러리입니다. **eBPF** 프로그램의 주요 유틸리티는 오버헤드나 보안 문제가 발생하지 않고 **OS** 성능 및 네트워크 성능을 분석하는 것입니다.

추가 리소스

- SS(8), ethtool(8), nettop(1), ip(8), dropwatch(1) 및 systemtap(8) 도움말 페이지
- /usr/share/systemtap/examples/network directory
- /usr/share/doc/bcc/README.md file
- ethtool 명령을 적용하기 위해 NetworkManager 디스패치 스크립트를 작성하는 방법은 무엇입니까? Red Hat Knowlegebase 솔루션
- ethtool 오프로드 기능 구성

34.2. 패킷 수신의 병목 현상

네트워크 스택은 대체로 자체적으로 최적화되지만 네트워크 패킷 처리 중에 병목 현상이 되고 성능을 저하시킬 수 있는 여러 가지 지점이 있습니다.

다음은 성능 장애를 일으킬 수 있는 문제입니다.

네트워크 카드의 버퍼 또는 링 버퍼

커널이 많은 수의 패킷을 삭제하는 경우 하드웨어 버퍼가 성능 장애일 수 있습니다. 삭제된 패킷에 대해 시스템을 모니터링하려면 ethtool 유틸리티를 사용합니다.

하드웨어 또는 소프트웨어 인터럽트 대기열

인터럽트를 사용하면 대기 시간 및 프로세서 경합이 증가할 수 있습니다. 프로세서에서 인터럽트를 처리하는 방법에 대한 자세한 내용은 인터럽트 요청 개요,수동으로 분산 인터럽트, smp_affinity 마스크 설정을 참조하십시오.

애플리케이션의 소켓 수신 대기열

/proc/net/snmp 파일의 UDP 입력 오류(오류)증가 또는 복사되지 않는 많은 패킷은 애플리케이션의 수신 대기열의 성능 장애를 나타냅니다.

하드웨어 버퍼가 많은 패킷을 삭제하는 경우 다음과 같은 몇 가지 가능한 솔루션이 있습니다.

입력 트래픽 속도 저하

들어오는 트래픽을 필터링하거나, 참여 멀티캐스트 그룹의 수를 줄이거나, 대기열이 채워지는 속도를 줄이기 위해 브로드캐스트 트래픽 양을 줄입니다.

하드웨어 버퍼 큐 크기 조정

하드웨어 버퍼 큐의 크기를 조정합니다. 쉽게 오버플로하지 않도록 대기열의 크기를 늘려 삭제되는 패킷 수를 줄입니다. **ethtool** 명령을 사용하여 네트워크 장치의 **rx/tx** 매개변수를 수정할 수 있습니다.

ethtool --set-ring ***device-name*** **value**

큐의 드레이닝 속도 변경

- 큐에 도달하기 전에 패킷을 필터링하거나 삭제하거나 장치의 가중치를 줄여 큐가 채우는 속도를 줄입니다. 들어오는 트래픽을 필터링하거나 네트워크 인터페이스 카드의 장치 가중치를 낮춰 들어오는 트래픽 속도가 느려집니다.

  장치 가중치는 단일 예약된 프로세서 액세스에서 한 번에 장치가 수신할 수 있는 패킷 수를 나타냅니다. **dev_weight** 커널 설정으로 제어되는 장치 가중치를 늘려 큐가 드레인되는 속도를 늘릴 수 있습니다. 이 매개 변수를 일시적으로 변경하려면 **/proc/sys/net/core/dev_weight** 파일의 내용을 변경하거나 영구적으로 변경하려면 **procps-ng** 패키지에서 제공하는 **sysctl** 명령을 사용합니다.

- 애플리케이션의 소켓 대기열의 길이를 늘립니다. 일반적으로 소켓 큐의 드레이닝 속도를 향상시키는 가장 쉬운 방법이지만 장기 솔루션이 아닐 수 있습니다. 소켓 큐가 버스트에서 제한된 트래픽 양을 수신하면 소켓 큐의 깊이를 늘려 트래픽 버스트 크기와 일치하도록 패킷이 삭제되는 것을 방지할 수 있습니다. 큐의 깊이를 늘리려면 다음 변경 사항 중 하나를 수행하여 소켓 수신 버퍼의 크기를 늘립니다.

  - **/proc/sys/net/core/rmem_default** 매개변수 값을 늘립니다. 이 매개변수는 소켓에 사용되는 수신 버퍼의 기본 크기를 제어합니다. 이 값은 **/proc/sys/net/core/rmem_max** 매개변수 값보다 작거나 같아야 합니다.

  - **setockopt**를 사용하여 더 큰 **SO_RCVBUF** 값을 구성합니다. 이 매개 변수는 소켓 수신 버퍼의 최대 바이트 크기를 제어합니다. **getockopt** 시스템 호출을 사용하여 현재 버퍼 값을 확인합니다.

일반적으로 큐의 드레이닝 속도를 변경하는 것이 좋지 않은 네트워크 성능을 완화하는 가장 간단한 방법입니다. 그러나 한 번에 장치를 수신할 수 있는 패킷 수를 늘리면 추가 프로세서 시간이 사용되므로 다른 프로세스를 예약할 수 없으므로 다른 성능 문제가 발생할 수 있습니다.

추가 리소스

- SS(8), socket(7) 및 ethtool(8) 도움말 페이지
- /proc/net/snmp 파일

34.3. 사용 중인 폴링

분석을 통해 대기 시간이 길어지면 시스템이 인터럽트 기반 패킷 수신이 아닌 폴링 기반 패킷의 이점을 얻을 수 있습니다.

사용량이 많은 폴링을 사용하면 소켓 계층 코드가 네트워크 장치의 수신 대기열을 폴링하고 네트워크 인터럽트를 비활성화하여 네트워크 수신 경로의 대기 시간을 줄일 수 있습니다. 이렇게 하면 인터럽트 및 결과 컨텍스트 스위치로 인한 지연이 제거됩니다. 그러나 CPU 사용률도 증가합니다. 사용량이 많은 폴링을 통해 CPU가 유휴 상태가 되지 않아 추가 전력 소비가 발생할 수 있습니다. 사용 중인 폴링 동작은 모든 장치 드라이버에서 지원됩니다.

추가 리소스

- 사용 중인 폴링 활성화

34.3.1. 사용 중인 폴링 활성화

기본적으로 사용 가능한 폴링은 비활성화되어 있습니다. 이 절차에서는 사용 중인 폴링을 활성화하는 방법을 설명합니다.

절차

1. CONFIG_NET_RX_BUSY_POLL 컴파일 옵션이 활성화되어 있는지 확인합니다.

```
# cat /boot/config-$(uname -r) | grep CONFIG_NET_RX_BUSY_POLL
CONFIG_NET_RX_BUSY_POLL=y
```

2. 사용 중인 폴링 활성화

   a. 특정 소켓에서 사용 중인 폴링을 활성화하려면 sysctl.net.core.busy_poll 커널 값을 0 이외의 값으로 설정합니다.

```
# echo "net.core.busy_poll=50" > /etc/sysctl.d/95-enable-busy-polling-for-sockets.conf
# sysctl -p /etc/sysctl.d/95-enable-busy-polling-for-sockets.conf
```

이 매개 변수는 소켓 폴링에서 패킷 대기할 마이크로초 수를 제어하고 **syscall**을 선택합니다. **Red Hat**은 **50** 개 이상의 가치를.

b. **SO_BUSY_POLL** 소켓 옵션을 소켓에 추가합니다.

c. 사용량이 많은 폴링을 활성화하려면 **sysctl.net.core.busy_read** 를 **0** 이외의 값으로 설정합니다.

```
# echo "net.core.busy_read=50" > /etc/sysctl.d/95-enable-busy-polling-globally.conf
# sysctl -p /etc/sysctl.d/95-enable-busy-polling-globally.conf
```

net.core.busy_read 매개 변수는 소켓 읽기를 위해 장치 큐에서 패킷을 대기할 마이크로초 수를 제어합니다. 또한 **SO_BUSY_POLL** 옵션의 기본값을 설정합니다. **Red Hat**은 소수의 소켓 수에 대해 **50** 의 값을, 다수의 소켓에 대해 값 **100** 을 권장합니다. 매우 많은 수의 소켓의 경우(예: 수백 개 이상) **epoll** 시스템 호출을 대신 사용합니다.

추가 리소스

- **ethtool(8), socket(7), sysctl(8)** 및 **sysctl.conf(5)** 도움말 페이지
- **ethtool** 오프로드 기능 구성

34.4. 수신 크기 조정

멀티 큐 수신이라고도 하는 수신 크기 조정(**RSS**)은 네트워크 수신 처리를 여러 하드웨어 기반 수신 대기열에 분산하여 인바운드 네트워크 트래픽을 여러 **CPU**에서 처리할 수 있도록 합니다. **RSS**는 단일 **CPU** 과부하로 인해 발생하는 수신 인터럽트 처리에서 병목 현상을 완화하고 네트워크 대기 시간을 줄이는 데 사용할 수 있습니다. 기본적으로 **RSS**가 활성화됩니다.

RSS의 네트워크 활동을 처리해야 하는 대기열 수 또는 **CPU**는 적절한 네트워크 장치 드라이버에 구성됩니다.

- **bnx2x** 드라이버의 경우 **num_queues** 매개변수로 구성됩니다.

- **sfc** 드라이버의 경우 **rss_cpus** 매개 변수에 구성됩니다.

그러나 일반적으로 **/sys/class/net/***장치***/queues/*rx-queue*/** 디렉터리에 구성됩니다. 여기서 ***device*** 는 네트워크 장치의 이름(예: **enp1s0**)이며 ***rx-queue*** 는 적절한 수신 대기열의 이름입니다.

irqbalance 데몬을 **RSS**와 함께 사용하여 교차 노드 메모리 전송 및 캐시 라인 변동 가능성을 줄일 수 있습니다. 이렇게 하면 네트워크 패킷을 처리하는 대기 시간이 줄어듭니다.

34.4.1. 인터럽트 요청 대기열 보기

RCS(Receive-Side Scaling)를 구성할 때는 대기열 수를 물리적 **CPU** 코어당 하나씩 제한하는 것이 좋습니다. 하이퍼 스레드는 종종 분석 도구에서 별도의 코어로 표현되지만 하이퍼 스레드와 같은 논리적 코어를 포함한 모든 코어에 대한 대기열은 네트워크 성능에 도움이 되지 않았습니다.

활성화된 경우 **RSS**는 각 **CPU**가 대기 중인 처리량에 따라 사용 가능한 **CPU** 간에 네트워크 처리를 동일하게 배포합니다. 그러나 **ethtool** 유틸리티의 **--show-rxfh-indir** 및 **--set-rxfh-indir** 매개 변수를 사용하여 **RHEL**에서 네트워크 활동을 배포하는 방법을 수정하고 특정 유형의 네트워크 활동을 다른 것보다 더 중요한 방식으로 조정합니다.

다음 절차에서는 인터럽트 요청 대기열을 보는 방법을 설명합니다.

절차

- 네트워크 인터페이스 카드가 **RSS**를 지원하는지 확인하려면 여러 인터럽트 요청 큐가 **/proc/interrupts** 의 인터페이스와 연결되어 있는지 확인합니다.

```
# egrep 'CPU|p1p1' /proc/interrupts
 CPU0    CPU1    CPU2    CPU3    CPU4    CPU5
89:  40187     0     0     0     0     0  IR-PCI-MSI-edge   p1p1-0
90:      0   790     0     0     0     0  IR-PCI-MSI-edge   p1p1-1
91:      0     0   959     0     0     0  IR-PCI-MSI-edge   p1p1-2
92:      0     0     0  3310     0     0  IR-PCI-MSI-edge   p1p1-3
93:      0     0     0     0   622     0  IR-PCI-MSI-edge   p1p1-4
94:      0     0     0     0     0  2475  IR-PCI-MSI-edge   p1p1-5
```

출력에는 **NIC** 드라이버에서 **p1p1** 인터페이스의 **6**개의 수신 대기열(**p1p 1-0 ~ p1p1-5 **)이 생성되었음을 보여줍니다. 또한 각 대기열에서 처리한 인터럽트 수와 인터럽트를 서비스한 **CPU**가 표시됩니다. 이 경우 기본적으로 이 특정 **NIC** 드라이버가 **CPU**당 하나의 큐를 생성하고 이 시스템

에는 6개의 CPU가 있으므로 6개의 대기열이 있습니다. 이것은 NIC 드라이버 간에 매우 일반적인 패턴입니다.

- 주소가 0000:01:00.0인 PCI 장치의 인터럽트 요청 대기열을 나열하려면 다음을 수행합니다.

```
# ls -1 /sys/devices/*/*/0000:01:00.0/msi_irqs
101
102
103
104
105
106
107
108
109
```

34.5. 패킷 제거를 수신

RPS(Receive-Side Scaling)는 처리를 위해 특정 CPU로 패킷을 보내는 데 사용된다는 점에서 RSP(Receive-Side Scaling)와 유사합니다. 그러나 RPS는 소프트웨어 수준에서 구현되며 단일 네트워크 인터페이스 카드의 하드웨어 대기열이 네트워크 트래픽에서 병목 현상이 되지 않도록 방지하는 데 도움이 됩니다.

RPS는 하드웨어 기반 RSS에 비해 몇 가지 장점이 있습니다.

- RPS는 모든 네트워크 인터페이스 카드와 함께 사용할 수 있습니다.
- RPS에 소프트웨어 필터를 추가하여 새 프로토콜을 처리하기 쉽습니다.
- RPS는 네트워크 장치의 하드웨어 인터럽트 속도를 늘리지 않습니다. 하지만 프로세서 간 인터럽트가 도입됩니다.

RPS는 /sys/class/net/ device /queues/*rx-queue*/rps_cpus 파일에서 네트워크 장치 및 수신 대기열별로 구성됩니다. 여기서 *device* 는 enp1s0 및 *rx-queue* 와 같은 네트워크 장치의 이름입니다(예: rx-0).

rps_cpus 파일의 기본값은 0입니다. 이렇게 하면 RPS를 비활성화하고 CPU는 네트워크 인터럽트를 처리하고 패킷을 처리합니다. RPS를 활성화하려면 지정된 네트워크 장치에서 패킷을 처리하고 큐를 수신해야 하는 CPU를 사용하여 적절한 rps_cpus 파일을 구성합니다.

rps_cpus 파일은 쉼표로 구분된 **CPU** 비트맵을 사용합니다. 따라서 **CPU**가 인터페이스에서 수신 대기열에 대한 인터럽트를 처리할 수 있도록 하려면 비트맵에서 위치 값을 **1**로 설정합니다. 예를 들어 **CPU 0,1,2, 3** 으로 인터럽트를 처리하려면 **rps_cpus** 의 값을 **15** 의 **16**진수 값인 **f** 로 설정합니다. 바이너리 표현에서 **15** 는 **00001111(1+2+4+8)** 입니다.

단일 전송 대기열이 있는 네트워크 장치의 경우 동일한 메모리 도메인에서 **CPU**를 사용하도록 **RPS**를 구성하여 최상의 성능을 얻을 수 있습니다. **NUMA**가 아닌 시스템에서는 사용 가능한 모든 **CPU**를 사용할 수 있습니다. 네트워크 인터럽트 속도가 매우 높은 경우 네트워크 인터럽트를 처리하는 **CPU**를 제외하고 성능도 향상될 수 있습니다.

여러 대기열이 있는 네트워크 장치의 경우 기본적으로 수신되는 각 대기열에 **CPU**를 매핑하도록 **RSS**를 구성하므로 일반적으로 **RPS** 및 **RSS**를 구성할 수 있는 이점이 없습니다. 그러나 **CPU**보다 하드웨어 대기열이 적고 **RPS**가 동일한 메모리 도메인에서 **CPU**를 사용하도록 구성된 경우에도 **RPS**는 여전히 유용할 수 있습니다.

34.6. 흐름 처리 수신

RFC(수동 패킷 제거)는 **CPU** 캐시 적중 속도를 높이고 네트워크 대기 시간을 줄이기 위해 **RCS**(수신 패킷 제거) 동작을 확장합니다. **RPS**가 대기열 길이를 기반으로 패킷을 전달하는 경우에는 **RPS** 백엔드를 사용하여 가장 적절한 **CPU**를 계산한 다음 패킷을 사용하는 애플리케이션 위치에 따라 패킷을 전달합니다. 따라서 **CPU** 캐시 효율성이 증가합니다.

단일 발신자에서 수신한 데이터는 둘 이상의 **CPU**로 전송되지 않습니다. 단일 발신자에서 수신한 데이터의 양이 단일 **CPU**에서 처리할 수 있는 것보다 크면, 인터럽트 수와 **CPU**의 처리 작업량을 줄이기 위해 더 큰 프레임 크기를 구성하십시오. 또는 **NIC** 오프로드 옵션 또는 더 빠른 **CPU**를 고려하십시오.

NUMAS와 함께 **numactl** 또는 **taskset** 을 사용하여 특정 코어, 소켓 또는 **NUMA** 노드에 애플리케이션을 고정하는 것이 좋습니다. 이렇게 하면 패킷이 주문에서 처리되지 않도록 방지할 수 있습니다.

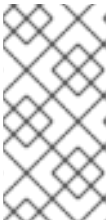
34.6.1. 수신 흐름 제거 활성화

기본적으로 **RFC**(수신 흐름 제거)는 비활성화되어 있습니다. 이 절차에서는 사용 방법에 대해 설명합니다.

절차

1. **net.core.rps_sock_flow_entries** 커널 값의 값을 예상되는 최대 동시 연결 수로 설정합니다.

```
# echo "net.core.rps_sock_flow_entries=32768" > /etc/sysctl.d/95-enable-rps.conf
```



참고

Red Hat은 보통 서버 부하를 위해 **32768** 의 값을 권장합니다. 입력한 모든 값은 실제로 **2** 의 가장 가까운 거듭제곱으로 반올림됩니다.

2. **net.core.rps_sock_flow_entries** 의 값을 영구적으로 설정합니다.

```
# sysctl -p /etc/sysctl.d/95-enable-rps.conf
```

3. **sys/class/net/*device*/queues/*rx-queue*/rps_flow_cnt** 파일의 값을 **(rps_sock_flow_entries/*N*)** 값으로 일시적으로 설정하려면 여기서 ***N*** 은 장치의 수신 대기열 수입니다.

```
# echo 2048 > /sys/class/net/device/queues/rx-queue/rps_flow_cnt
```

device 를 구성하려는 네트워크 장치의 이름으로 바꿉니다(예: **enp1s0**) 및 ***rx-queue*** 를 구성하려는 수신 대기열로 바꿉니다(예: ***rx-0***).

N 을 구성된 수신 대기열 수로 바꿉니다. 예를 들어 **rps_flow_entries** 가 **32768** 로 설정되어 있고 수신 대기열이 **16** 개 구성된 경우 **rps_flow_cnt = 32786/16= 2048** (즉, **rps_flow_cnt = rps_flow_enties/*N***).

단일 대기열 장치의 경우 **rps_flow_cnt** 값은 **rps_sock_flow_entries** 값과 동일합니다.

4. 모든 네트워크 장치에서 자동으로 활성화하고 **/etc/udev/rules.d/99-persistent-net.rules** 파일을 만들고 다음 내용을 추가합니다.

```
SUBSYSTEM=="net", ACTION=="add", RUN{program}+="/bin/bash -c 'for x in
/sys/$DEVPATH/queues/rx-*; do echo 2048 > $x/rps_flow_cnt;  done'"
```

5. 선택 사항: 특정 네트워크 장치에서 **RPS**를 활성화하려면 다음을 수행합니다.

```
SUBSYSTEM=="net", ACTION=="move", NAME="device name" RUN{program}+="/bin/bash
-c 'for x in /sys/$DEVPATH/queues/rx-*; do echo 2048 > $x/rps_flow_cnt; done'"
```

*장치 이름*을 실제 네트워크 장치 이름으로 바꿉니다.

검증 단계

- ✓S가 활성화되어 있는지 확인합니다.

```
# cat /proc/sys/net/core/rps_sock_flow_entries
32768

# cat /sys/class/net/device/queues/rx-queue/rps_flow_cnt
2048
```

추가 리소스

- **sysctl(8)** 도움말 페이지

34.7. 가속화됨

가속도로 하드웨어 지원을 추가하여 **RFC**(수신 흐름 제거)의 속도를 높일 수 있습니다. **1800S**와 마찬가지로 패킷을 사용하는 애플리케이션 위치에 따라 패킷이 전달됩니다.

그러나 기존 **ICS**와 달리 패킷은 데이터를 사용하는 스레드에 로컬인 **CPU**로 직접 전송됩니다.

- 애플리케이션을 실행 중인 **CPU** 중 하나
- 또는 캐시 계층 구조의 해당 **CPU**에 대한 로컬 **CPU**

가속화된**œS**는 다음 조건이 충족되는 경우에만 사용할 수 있습니다.

- **NIC**에서 가속화된 상자를 지원해야 합니다. 가속화된**œS**는 **the ndo_rx_flow_steer() net_ device** 함수를 내보내는 카드에서 지원됩니다. **NIC**의 데이터 시트를 확인하여 이 기능이 지원되는지 확인합니다.
- **ntuple** 필터링을 활성화해야 합니다. 이러한 필터를 활성화하는 방법에 대한 자세한 내용은 **ntuple** 필터 활성화를 참조하십시오.

이러한 조건이 충족되면 **CPU**와 대기열 간의 매핑이 기존 **MachineConfigS** 구성에 따라 자동으로 줄어듭니다. 즉, **CPU to queue** 매핑은 각 수신 대기열에 대해 드라이버가 구성한 **IRQ** 선호도를 기반으로 합

니다. 기존 ICS 활성화에 대한 자세한 내용은 수신 흐름 활성화를 참조하십시오.

34.7.1. ntuple 필터 활성화

ntuple 필터링을 활성화해야 합니다. ethtool -k 명령을 사용하여 ntuple 필터를 활성화합니다.

절차

1. ntuple 필터의 현재 상태를 표시합니다.

```
# ethtool -k enp1s0 | grep ntuple-filters

ntuple-filters: off
```

2. ntuple 필터를 활성화합니다.

```
# ethtool -k enp1s0 ntuple on
```

참고

출력이 ntuple-filters: off [fixed] 인 경우 ntuple 필터링이 비활성화되어 구성할 수 없습니다.

```
# ethtool -k enp1s0 | grep ntuple-filters
ntuple-filters: off [fixed]
```

검증 단계

- ntuple 필터가 활성화되어 있는지 확인합니다.

```
# ethtool -k enp1s0 | grep ntuple-filters
ntuple-filters: on
```

추가 리소스

- ethtool(8) 도움말 페이지

35장. 메모리 액세스를 최적화하도록 운영 체제 구성

이 섹션에서는 워크로드 전체에서 메모리 액세스를 최적화하도록 운영 체제를 구성하는 방법과 이를 수행하는 데 사용할 수 있는 툴에 대해 설명합니다.

35.1. 시스템 메모리 문제 모니터링 및 진단 툴

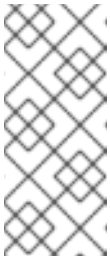
시스템 성능을 모니터링하고 시스템 메모리와 관련된 성능 문제를 진단하기 위해 **Red Hat Enterprise Linux 8**에서 다음 도구를 사용할 수 있습니다.

- **procps-ng** 패키지에서 제공하는 **vmstat** 도구는 시스템 프로세스, 메모리, 페이징, 블록 I/O, 트랩, 디스크 및 CPU 활동에 대한 보고서를 표시합니다. 시스템이 마지막으로 켜졌거나 이전 보고서 이후부터 이러한 이벤트의 평균 즉시 보고서를 제공합니다.

- **Valgrind** 프레임워크는 사용자 공간 바이너리에 대한 계측을 제공합니다. **yum install valgrind** 명령을 사용하여 이 툴을 설치합니다. 여기에는 다음과 같은 프로그램 성능을 프로파일링 및 분석하는 데 사용할 수 있는 여러 도구가 포함되어 있습니다.

  - **Memcheck** 옵션은 기본 **valgrind** 툴입니다. 다음과 같이 감지 및 진단하기 어려운 여러 메모리 오류를 감지하고 보고합니다.

    - 발생해서는 안 되는 메모리 액세스
    - 정의되지 않았거나 초기화되지 않은 값 사용
    - 잘못 사용 가능한 힙 메모리
    - 포인터 중복
    - 메모리 누수



참고

Memcheck는 이러한 오류만 보고할 수 있으며 발생을 방지할 수 없습니다. 그러나 **memcheck** 는 오류가 발생하기 직전에 오류 메시지를 기록합니다.

- **cachegrind** 옵션은 시스템의 캐시 계층 구조 및 분기 예측자와 애플리케이션 상호 작용을 시뮬레이션합니다. 애플리케이션 실행 기간에 대한 통계를 수집하고 콘솔에 대한 요약을 출력합니다.

- 대용량 옵션은 지정된 애플리케이션에서 사용하는 힙 공간을 측정합니다. 또한 유용한 공간 및 서약 및 정렬 목적으로 할당된 추가 공간을 모두 측정합니다.

추가 리소스

- **vmstat(8)** 및 **valgrind(1)** 도움말 페이지

- **/usr/share/doc/valgrind-version/valgrind_manual.pdf file**

35.2. 시스템 메모리 개요

Linux 커널은 시스템의 메모리 리소스**(RAM)**의 활용도를 극대화하도록 설계되었습니다. 이러한 설계 특성으로 인해 워크로드의 메모리 요구 사항에 따라 시스템 메모리의 일부는 워크로드를 대신하여 커널 내에서 사용 중인 반면 메모리의 일부는 사용 가능한 상태입니다. 이 사용 가능한 메모리는 특수 시스템 할당 및 우선 순위가 낮은 다른 시스템 서비스를 위해 예약됩니다.

나머지 시스템 메모리는 워크로드 자체에 국한되어 있으며 다음과 같은 두 가지 범주로 나뉩니다.

파일 메모리

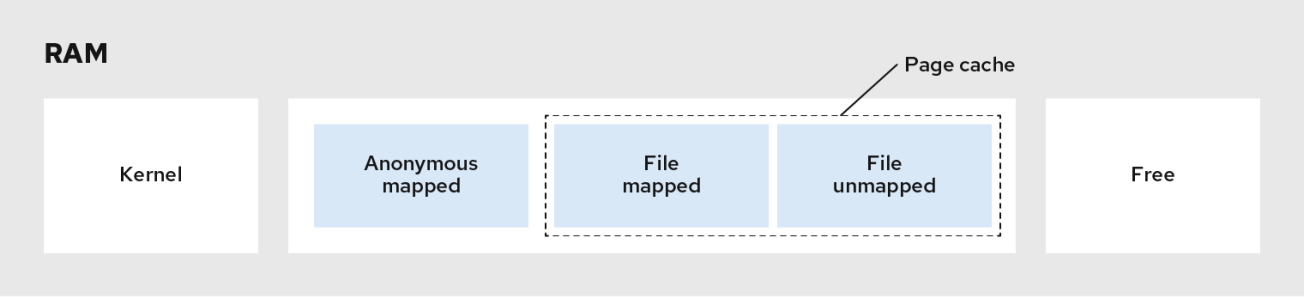
이 범주에 추가된 페이지는 영구저장장치의 파일 일부를 나타냅니다. 페이지 캐시에서 이러한 페이지를 애플리케이션 주소 공간에 매핑하거나 매핑 해제할 수 있습니다. 애플리케이션을 사용하여 **mmap** 시스템 호출을 사용하여 파일을 주소 공간에 매핑하거나 버퍼링된 **I/O** 읽기 또는 쓰기 시스템 호출을 통해 파일에서 작동할 수 있습니다.

버퍼링된 **I/O** 시스템 호출과 페이지를 직접 매핑하는 애플리케이션은 매핑되지 않은 페이지를 다시 사용할 수 있습니다. 결과적으로 이러한 페이지는 동일한 페이지 집합에서 값비싼 **I/O** 작업을 다시 진행하지 않도록 시스템에서 메모리 집약적인 작업을 실행하지 않는 커널의 캐시에 저장됩니다.

익명 메모리

이 범주의 페이지는 동적으로 할당된 프로세스에서 사용되거나 영구 스토리지의 파일과 관련이 없습니다. 이 페이지 집합은 애플리케이션 스택 및 힙 영역과 같은 각 작업의 메모리 내 제어 구조를 백업합니다.

그림 35.1. 메모리 사용 패턴



133_RHEL_0121

35.3. 가상 메모리 매개변수

가상 메모리 매개 변수는 /proc/sys/vm 디렉터리에 나열됩니다.

다음은 사용 가능한 가상 메모리 매개변수입니다.

vm.dirty_ratio

는 백분율 값입니다. 총 시스템 메모리의 이 백분율이 수정되면 시스템은 lush 작업을 사용하여 디스크 수정 사항 작성을 시작합니다. 기본값은 20%입니다.

vm.dirty_background_ratio

백분율 값입니다. 총 시스템 메모리의 이 백분율이 수정되면 시스템은 백그라운드에서 디스크에 대한 수정 사항 쓰기를 시작합니다. 기본값은 10%입니다.

vm.overcommit_memory

큰 메모리 요청이 수락되었는지 여부를 결정하는 조건을 정의합니다. 기본값은 0 입니다.

기본적으로 커널은 사용 가능한 메모리 양을 추정하고 너무 큰 요청에 실패하여 추론적 메모리 과다 할당 처리를 수행합니다. 그러나 메모리는 정확한 알고리즘이 아닌 추론을 사용하여 할당되므로 이 설정에서는 메모리 과부하가 가능합니다.

overcommit_memory 매개변수의 값을 설정합니다.

- 이 매개 변수를 **1** 로 설정하면 커널이 메모리 과다 할당 처리를 수행하지 않습니다. 이렇게 하면 메모리 과부하 가능성이 높아지지만, 메모리를 많이 사용하는 작업의 성능이 향상됩니다.

- 이 매개 변수가 **2** 로 설정되면 커널은 사용 가능한 총 스왑 공간의 합계와 동일한 메모리 요청과 **overcommit_ratio** 에 지정된 물리적 **RAM**의 백분율을 거부합니다. 이렇게 하면 메모리 과다 할당 위험이 줄어듭니다. 그러나 스왑 영역이 실제 메모리보다 큰 시스템에만 권장됩니다.

vm.overcommit_ratio

overcommit_memory 가 **2** 로 설정된 경우 물리적 **RAM**의 백분율을 지정합니다. 기본값은 **50** 입니다.

vm.max_map_count

프로세스에서 사용할 수 있는 최대 메모리 맵 영역 수를 정의합니다. 기본값은 **65530** 입니다. 애플리케이션에 더 많은 메모리 맵 영역이 필요한 경우 이 값을 늘립니다.

vm.min_free_kbytes

예약된 여유 페이지 풀의 크기를 설정합니다. 또한 **Linux** 커널 페이지의 동작을 제어하는 **min_page, low _page** 및 **high_page** 임계값을 설정하는 작업도 담당합니다. 또한 시스템 전체에서 사용할 수 있는 최소 킬로바이트 수를 지정합니다. 이렇게 하면 작은 메모리 영역마다 특정 값이 계산되며, 각 메모리 영역에는 크기에 비례하여 예약된 여러 개의 여유 페이지가 할당됩니다.

vm.min_free_kbytes 매개변수 값을 설정합니다.

- 매개 변수 값을 늘리면 애플리케이션 작업 세트 사용 가능한 메모리가 효과적으로 줄어듭니다. 따라서 드라이버 버퍼를 원자 컨텍스트에서 할당해야 하는 커널 중심 워크로드에만 사용할 수 있습니다.

- 매개 변수 값을 줄이면 메모리가 시스템에 많이 누적되는 경우 커널이 시스템 요청을 서비스할 수 없게 될 수 있습니다.

주의

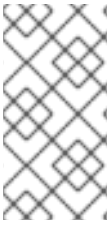
극심한 값은 시스템 성능 저하가 될 수 있습니다. **vm.min_free_kbytes** 를 매우 낮은 값으로 설정하면 시스템이 메모리를 효과적으로 회수하지 못하므로 시스템 충돌이 발생하고 서비스 인터럽트 또는 기타 커널 서비스에 오류가 발생할 수 있습니다. 그러나 **vm.min_free_kbytes** 를 너무 높게 설정하면 시스템 회수 활동이 너무 증가하여 잘못된 직접 회수 상태로 인해 할당 대기 시간이 발생했습니다. 이로 인해 시스템이 메모리 부족 상태가 즉시 입력될 수 있습니다.

vm.min_free_kbytes 매개 변수는 **min_pages** 라는 페이지 회수 워터마크도 설정합니다. 이 워터마크는 페이지를 회수하는 알고리즘을 제어하는 다른 두 개의 메모리 워터마크인 **low_pages** 및 **high_pages** 를 결정할 때 인수로 사용됩니다.

/proc/*PID*/oom_adj

시스템이 메모리가 부족하고 **panic_on_oom** 매개 변수가 **0** 으로 설정된 경우, 시스템이 복구될 때까지 가장 **high oom_ score**가 있는 프로세스부터 시작하여**oom_ killer** 함수에서 프로세스를 종료합니다.

The oom_adj 매개 변수는 프로세스의 **oom_score** 를 결정합니다. 이 매개 변수는 프로세스 식별자별로 설정됩니다. 값이 **-17** 이면 해당 프로세스의 **oom_killer** 가 비활성화됩니다. 다른 유효한 값은 **-16** 에서 **15** 사이입니다.



참고

조정된 프로세스에서 생성한 프로세스는 해당 프로세스의 **the oom_score** 를 상속받습니다.

vm.swappiness

0 에서 **100** 사이의 **swappiness** 값은 시스템이 익명 메모리 풀에서 메모리를 회수하는 정도 또는 페이지 캐시 메모리 풀을 제어합니다.

swapp iness 매개변수의 값 설정:

- 액세스가 덜 사용되는 프로세스의 익명 매핑 메모리는 스왑 아웃하는 동시에 파일 매핑

기반 워크로드를 선호합니다. 이 기능은 스토리지의 파일에서 데이터를 사용하는 파일 서버 또는 스트리밍 애플리케이션에서 서비스 요청에 대한 I/O 대기 시간을 줄이기 위해 메모리에 상주하는 데 유용합니다.

- 낮은 값은 페이지 캐시를 회수하는 동시에 익명 매핑 주도 워크로드를 선호합니다(파일 매핑된 메모리). 이 설정은 파일 시스템 정보에 크게 의존하지 않는 애플리케이션에 유용하며, 동적으로 할당된 프라이빗 메모리(예: 수학 및 번호 입력 애플리케이션, QEMU와 같은 하드웨어 가상화 관리자)를 많이 활용하는 데 유용합니다.

  vm.swappiness 매개 변수의 기본값은 30 입니다.

주의

vm.swappiness 를 0 으로 설정하면 익명 메모리를 디스크로 스와핑하는 것을 적극적으로 방지할 수 있으므로 메모리 또는 I/O 집약적 워크로드가 있을 때 the oom_killer 함수에 의해 프로세스가 종료될 위험이 높아집니다.

추가 리소스

- sysctl(8) 도움말 페이지
- 메모리 관련 커널 매개변수 설정

35.4. 파일 시스템 매개변수

파일 시스템 매개변수는 /proc/sys/fs 디렉터리에 나열됩니다. 다음은 사용 가능한 파일 시스템 매개변수입니다.

aio-max-nr

모든 활성 비동기 입력/출력 컨텍스트에서 허용되는 최대 이벤트 수를 정의합니다. 기본값은 65536 이며 이 값을 수정해도 커널 데이터 구조의 크기를 사전 할당하거나 크기를 조정하지 않습니다.

file-max

전체 시스템의 최대 파일 처리 수를 결정합니다. **Red Hat Enterprise Linux 8**의 기본값은 **8192** 또는 커널이 시작될 때 사용 가능한 메모리 페이지 중 **1/10**에 해당하는 값입니다.

이 값을 늘리면 사용 가능한 파일 처리 부족으로 인한 오류가 해결될 수 있습니다.

추가 리소스

- **sysctl(8)** 도움말 페이지

35.5. 커널 매개변수

커널 매개 변수의 기본값은 **/proc/sys/kernel/** 디렉터리에 있습니다. 이러한 값은 **sysctl** 을 통해 사용자가 지정한 기본값 또는 커널에서 제공하는 기본값입니다.

다음은 **msg*** 및 **shm*** **System V IPC(sysvipc)** 시스템 호출에 대한 제한을 설정하는 데 사용되는 사용 가능한 커널 매개변수입니다.

msgmax

메시지 큐에 있는 단일 메시지의 최대 허용 크기(바이트)를 정의합니다. 이 값은 대기열 크기 (**msgmnb**)를 초과해서는안 됩니다. **sysctl msgmax** 명령을 사용하여 시스템의 현재 **msgmax** 값을 확인합니다.

msgmnb

단일 메시지 큐의 최대 크기(바이트)를 정의합니다. **sysctl msgmnb** 명령을 사용하여 시스템의 현재 **msgmnb** 값을 확인합니다.

msgmni

최대 메시지 큐 식별자 수 및 최대 대기열 수를 정의합니다. **sysctl msgmni** 명령을 사용하여 시스템의 현재 **msgmni** 값을 확인합니다.

shmall

한 번에 시스템에서 사용할 수 있는 공유 메모리 페이지의 총 양을 정의합니다. 예를 들어, 페이지는 **AMD64** 및 **Intel 64** 아키텍처에서 **4096** 바이트입니다. **sysctl shmall** 명령을 사용하여 시스템의 현재 **shmall** 값을 확인합니다.

shmmax

커널에서 허용하는 단일 공유 메모리 세그먼트의 최대 크기(바이트)를 정의합니다. **sysctl shmmax** 명령을 사용하여 시스템의 현재 **shmmax** 값을 확인합니다.

shmmni

시스템 전체의 최대 공유 메모리 세그먼트 수를 정의합니다. 모든 시스템에서 기본값은 **4096** 입니다.

추가 리소스

- **sysvipc(7)** 및 **sysctl(8)** 도움말 페이지

35.6. 메모리 관련 커널 매개변수 설정

매개 변수를 일시적으로 설정하는 것은 시스템에 매개 변수가 미치는 영향을 결정하는 데 유용합니다. 매개변수 값에 원하는 효과가 있는지 확인할 때 나중에 매개 변수를 영구적으로 설정할 수 있습니다.

다음 절차에서는 메모리 관련 커널 매개 변수를 일시적으로 그리고 영구적으로 설정하는 방법을 설명합니다.

절차

- 메모리 관련 커널 매개 변수를 일시적으로 설정하려면 **/proc** 파일 시스템에서 해당 파일을 편집합니다.

  예를 들어 **vm.overcommit_memory** 매개변수를 일시적으로 ***1*** 로 설정하려면 다음을 실행합니다.

  ```
  # echo 1 > /proc/sys/vm/overcommit_memory
  # sysctl -w vm.overcommit_memory=1
  ```

- 메모리 관련 커널 매개 변수를 영구적으로 설정하려면 **sysctl** 도구를 사용합니다.

  예를 들어 **vm.overcommit_memory** 매개변수를 영구적으로 ***1*** 로 설정하려면 다음을 수행합니다.

  - **/etc/sysctl.conf** 파일에 다음 내용을 추가합니다.

    ```
    vm.overcommit_memory=1
    ```

- **/etc/sysctl.conf** 파일에서 **sysctl** 설정을 다시 로드합니다.

```
# sysctl -p
```

추가 리소스

- **sysctl(8)** 도움말 페이지

36장. 대규모 페이지 구성

물리적 메모리는 페이지라는 고정된 크기의 청크로 관리됩니다. Red Hat Enterprise Linux 8에서 지원되는 x86_64 아키텍처에서 메모리 페이지의 기본 크기는 4KB 입니다. 이 기본 페이지 크기는 다양한 종류의 워크로드를 지원하는 Red Hat Enterprise Linux와 같은 범용 운영 체제에 적합합니다.

그러나 특정 애플리케이션은 특정 경우에 더 큰 페이지 크기를 사용할 경우 이점을 얻을 수 있습니다. 예를 들어, 수백 메가바이트 또는 수십 기가바이트로 구성된 대규모 데이터 세트와 함께 작동하는 애플리케이션은 4KB 페이지를 사용할 때 성능 문제가 발생할 수 있습니다. 이러한 데이터 세트에는 운영 체제 및 CPU에서 오버헤드가 발생할 수 있는 상당한 양의 4KB 페이지가 필요할 수 있습니다.

이 섹션에서는 RHEL 8에서 사용할 수 있는 대규모 페이지 및 구성 방법에 대한 정보를 제공합니다.

36.1. 사용 가능한 대규모 페이지 기능

Red Hat Enterprise Linux 8을 사용하면 빅 데이터 세트에서 작업하는 애플리케이션에 대규모 페이지를 사용하고 이러한 애플리케이션의 성능을 향상시킬 수 있습니다.

다음은 RHEL 8에서 지원되는 대규모 페이지 방법입니다.

HugeTLB 페이지

HugeTLB 페이지를 정적 대규모 페이지라고도 합니다. HugeTLB 페이지를 예약하는 방법은 다음 두 가지가 있습니다.

- 부팅 시: 메모리가 아직 크게 조각화되지 않았기 때문에 성공 가능성이 높아집니다. 그러나 NUMA 시스템에서 페이지 수는 NUMA 노드 간에 자동으로 분할됩니다. 부팅 시 HugeTLB 페이지 동작에 영향을 주는 매개변수에 대한 자세한 내용은 부팅 시 HugeTLB 페이지를 예약하기 위한 매개 변수 및 부팅 시 HugeTLB 페이지를 구성하는 방법을 참조하십시오.
- 런타임 시: NUMA 노드당 대규모 페이지를 예약할 수 있습니다. 부팅 과정에서 런타임 예약이 최대한 빨리 수행되면 메모리 조각화 가능성이 낮습니다. 런타임 시 HugeTLB 페이지 동작에 영향을 주는 매개변수에 대한 자세한 내용은 런타임에 HugeTLB 페이지를 다시 예약하기 위한 매개 변수 및 런타임 시 HugeTLB 페이지를 구성하는 방법에 대한 자세한 내용은 런타임 시 HugeTLB 구성을 참조하십시오.

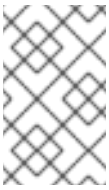
THP(Transparent HugePages)

THP에서는 커널이 대규모 페이지를 프로세스에 자동으로 할당하므로 정적 대규모 페이지를 수동

으로 예약할 필요가 없습니다. 다음은 THP에서 두 가지 작업 모드입니다.

- **system-wide:** 여기에서 커널은 대규모 페이지를 할당할 수 있을 때마다 대규모 페이지를 프로세스에 할당하려고 하며 프로세스에서 대규모 가상 메모리 영역을 사용합니다.

- 프로세스 별: 여기에서 커널은 개별 프로세스의 메모리 영역에만 대규모 페이지를 할당합니다. 이 영역에서는 시스템호출()을 사용하여 지정할 수 있습니다.



참고

THP 기능은 2MB 페이지만 지원합니다.

부팅 시 HugeTLB 페이지 동작에 영향을 주는 매개변수에 대한 자세한 내용은 투명한 hugepages 및 투명한 hugepages 비활성화를 참조하십시오.

36.2. 부팅 시 HUGETLB 페이지를 예약하기 위한 매개변수

다음 매개 변수를 사용하여 부팅 시 HugeTLB 페이지 동작에 영향을 줍니다.

부팅 시 이러한 매개변수를 사용하여 HugeTLB 페이지를 구성하는 방법에 대한 자세한 내용은 부팅 시 HugeTLB 구성을 참조하십시오.

표 36.1. 부팅 시 HugeTLB 페이지를 구성하는 데 사용되는 매개변수

매개변수	설명	기본값
hugepages	부팅 시 커널에 구성된 영구 대규모 페이지 수를 정의합니다. NUMA 시스템에서 이 매개 변수가 정의된 대규모 페이지는 노드 간에 동일하게 나뉩니다. **/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages** 파일의 노드 값을 변경하여 런타임에 특정 노드에 대규모 페이지를 할당할 수 있습니다.	기본값은 **0**입니다. 부팅 시 이 값을 업데이트하려면 **/proc/sys/vm/nr_hugepages** 파일에서 이 매개 변수의 값을 변경합니다.

매개변수	설명	기본값
hugepagesz	부팅 시 커널에 구성된 영구 대규모 페이지의 크기를 정의합니다.	유효한 값은 **2MB** 와 **1GB입니다**. 기본값은 **2MB** 입니다.
default_hugepagesz	부팅 시 커널에 구성된 영구 대규모 페이지의 기본 크기를 정의합니다.	유효한 값은 **2MB** 와 **1GB입니다**. 기본값은 **2MB** 입니다.

36.3. 부팅 시 HUGETLB 구성

HugeTLB 하위 시스템에서 지원하는 페이지 크기는 아키텍처에 따라 다릅니다. **x86_64** 아키텍처는 **2MB** 의 대규모 페이지와 **1GB** 기유지 페이지를 지원합니다.

다음 절차에서는 부팅 시 **1GB** 페이지를 예약하는 방법을 설명합니다.

절차

1. **/etc/default/grub** 파일의 커널 명령줄 옵션에 다음 행을 루트로 추가하여 **1GB** 페이지에 대한 **HugeTLB** 풀을 만듭니다.

```
default_hugepagesz=1G hugepagesz=1G
```

2. 편집된 기본 파일을 사용하여 **GRUB2** 설정을 다시 생성합니다.

   a. 시스템에서 **BIOS** 펌웨어를 사용하는 경우 다음 명령을 실행합니다.

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

   b. 시스템에서 **UEFI** 프레임워크를 사용하는 경우 다음 명령을 실행합니다.

   ```
   # grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
   ```

3. **/usr/lib/systemd/system/** 디렉토리에 **hugetlb-gigantic-pages.service** 라는 새 파일을 생성하고 다음 내용을 추가합니다.

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
```

```
DefaultDependencies=no
Before=dev-hugepages.mount
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/lib/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

4. **/usr/lib/systemd/** 디렉토리에 **hugetlb-reserve-pages.sh** 라는 새 파일을 생성하고 다음 내용을 추가합니다.

   다음 콘텐츠를 추가하는 동안 ***number_of_pages*** 를 예약할 **1GB** 페이지 수로 교체하고, ***node*** 를 해당 페이지를 예약할 노드 이름으로 바꿉니다.

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
   echo "ERROR: $nodes_path does not exist"
   exit 1
fi

reserve_pages()
{
   echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages number_of_pages node
```

   예를 들어 **node0**에 **1GB** 페이지 **2**개와 ***node 1에 1GB 페이지를 예약하려면 node 0*** 의 경우 ***number_of_pages*** 를 **2**로, ***node1 의 경우 1*** 로 교체합니다.

```
reserve_pages 2 node0
reserve_pages 1 node1
```

5. 실행 가능한 스크립트를 생성합니다.

```
# chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
```

6. 초기 부팅 예약 활성화:

-

```
# systemctl enable hugetlb-gigantic-pages
```

참고

- 언제든지 **nr_hugepages** 에 써서 런타임에 **1GB**페이지를 더 예약해 볼 수 있습니다. 그러나 이러한 예약은 메모리 조각화로 인해 실패할 수 있습니다. **1GB** 페이지를 예약하는 가장 안정적인 방법은 부팅 중에 조기에 실행되는 **hugetlb-reserve-pages.sh** 스크립트를 사용하는 것입니다.

- 정적 대규모 페이지를 예약하면 시스템에서 사용할 수 있는 메모리의 양을 효과적으로 줄이고 전체 메모리 용량을 제대로 활용하지 못하도록 할 수 있습니다. 예약된 대규모 페이지의 적절한 크기 풀은 이를 활용하는 애플리케이션에 도움이 될 수 있지만 예약된 대규모 페이지의 과도한 크기 또는 사용되지 않은 풀은 결과적으로 전체 시스템 성능에 지장을 줍니다. 예약된 대규모 페이지 풀을 설정할 때 시스템이 전체 메모리 용량을 적절히 활용할 수 있는지 확인합니다.

추가 리소스

- **systemd.service(5)** 도움말 페이지
- **/usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt file**

36.4. 런타임 시 HUGETLB 페이지를 예약하기 위한 매개변수

다음 매개 변수를 사용하여 런타임 시 **HugeTLB** 페이지 동작에 영향을 줍니다.

이러한 매개변수를 사용하여 런타임에 **HugeTLB** 페이지를 구성하는 방법에 대한 자세한 내용은 런타임 시 **HugeTLB** 구성을 참조하십시오.

표 **36.2.** 런타임 시 **HugeTLB** 페이지를 구성하는 데 사용되는 매개변수

매개변수	설명	파일 이름
nr_hugepages	지정된 NUMA 노드에 할당된 지정된 크기의 대규모 페이지 수를 정의합니다.	**/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages**

매개변수	설명	파일 이름
nr_overcommit_hugepages	메모리 과다 할당을 통해 시스템에서 생성 및 사용할 수 있는 최대 대규모 페이지 수를 정의합니다. 0이 아닌 값을 이 파일에 작성하면 영구 대규모 페이지 풀이 고갈되면 시스템이 커널의 일반 페이지 풀에서 대규모 페이지 수를 확보합니다. 이러한 과도하게 대규모 페이지가 사용되지 않으므로 해제되고 커널의 일반 페이지 풀로 돌아갑니다.	**/proc/sys/vm/nr_overcommit_hugepages**

36.5. 런타임 시 HUGETLB 구성

다음 절차에서는 ***20 2048 kB*** 대규모 페이지를 ***node2*** 에 추가하는 방법을 설명합니다.

요구 사항에 따라 페이지를 예약하려면 다음을 교체하십시오.

- ***20*** 예약하려는 대규모 페이지 수와 함께
- 대규모 페이지 크기가 *있는* ***2048*** KB,
- 페이지를 예약하려는 노드가 있는 **node *2*** 입니다.

절차

1. 메모리 통계를 표시합니다.

```
# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1 Node 2 Node 3  Total add
AnonHugePages         0      2      0      8     10
HugePages_Total       0      0      0      0      0
HugePages_Free        0      0      0      0      0
HugePages_Surp        0      0      0      0      0
```

2. 지정된 크기의 대규모 페이지 수를 노드에 추가합니다.

```
# echo 20 > /sys/devices/system/node/node2/hugepages/hugepages-2048kB/nr_hugepages
```

검증 단계

- 대규모 페이지 수가 추가되었는지 확인합니다.

```
# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1 Node 2 Node 3  Total
AnonHugePages         0      2      0      8     10
HugePages_Total       0      0     40      0     40
HugePages_Free        0      0     40      0     40
HugePages_Surp        0      0      0      0      0
```

추가 리소스

- **numastat(8)** 도움말 페이지

36.6. 투명한 HUGEPAGE 활성화

THP는 **Red Hat Enterprise Linux 8**에서 기본적으로 활성화되어 있습니다. 그러나 **THP**를 활성화하거나 비활성화할 수 있습니다.

이 절차에서는 **THP**를 활성화하는 방법을 설명합니다.

절차

1. **THP**의 현재 상태를 확인합니다.

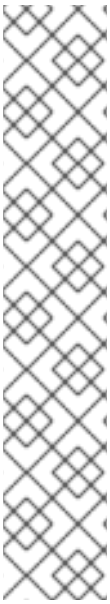
```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. **THP**를 활성화합니다.

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

3. 애플리케이션이 필요 이상의 메모리 리소스를 할당하지 않도록 하려면 시스템 전체에서 투명한 대규모 페이지를 비활성화하고 명시적으로 요청하는 애플리케이션에 대해서만 활성화합니다.

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```



참고

수명이 짧은 할당에 대기 시간을 제공하면 바로 수명이 긴 할당으로 최상의 성능을 달성하는 것보다 우선 순위가 높습니다. 이러한 경우 **THP**를 활성화하는 동안 직접 압축을 비활성화할 수 있습니다.

직접 압축은 대규모 페이지 할당 중에 동기 메모리 압축입니다. 직접 압축을 비활성화하면 메모리를 절약할 수 없지만 자주 발생하는 페이지 폴트 중에 대기 시간이 길어질 수 있습니다. 워크로드가 **THP**에서 상당한 이점을 얻는 경우 성능이 저하됩니다. 직접 압축을 비활성화합니다.

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
```

추가 리소스

- **jackvise(2)** 도움말 페이지
- 투명한 **hugepages** 비활성화.

36.7. 투명한 HUGEPAGE 비활성화

THP는 **Red Hat Enterprise Linux 8**에서 기본적으로 활성화되어 있습니다. 그러나 **THP**를 활성화하거나 비활성화할 수 있습니다.

이 절차에서는 **THP**를 비활성화하는 방법을 설명합니다.

절차

1. **THP**의 현재 상태를 확인합니다.

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. **THP**를 비활성화합니다.

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

36.8. 번역 조회 버퍼 크기에 대한 페이지 크기 영향

페이지 테이블에서 주소 매핑을 읽는 데 시간과 리소스가 많이 소요되므로 CPU는 최근 사용된 주소에 대한 캐시를 사용하여 빌드됩니다(TLB(Translation Lookaside Buffer). 그러나 기본 TLB는 특정 수의 주소 매핑만 캐시할 수 있습니다.

요청된 주소 매핑이 TLB 누락이라고 하는 TLB에 없는 경우에도 시스템은 여전히 페이지 테이블을 읽어 물리적 주소에서 가상 주소 매핑을 판별해야 합니다. 애플리케이션 메모리 요구 사항과 주소 매핑을 캐시하는 데 사용되는 페이지 크기 간의 관계로 인해 대규모 메모리 요구 사항이 있는 애플리케이션이 메모리 요구 사항을 최소화하는 애플리케이션보다 TLB 누락으로 인한 성능이 저하될 가능성이 높습니다. 따라서 가능한 TLB 누락을 방지하는 것이 중요합니다.

HugeTLB 및 투명한 대규모 페이지 기능 모두 애플리케이션에서 4KB보다 큰 페이지를 사용할 수 있습니다. 이를 통해 TLB에 저장된 주소가 더 많은 메모리를 참조하여 TLB 누락을 줄이고 애플리케이션 성능을 향상시킬 수 있습니다.

37장. SYSTEMTAP 시작하기

시스템 관리자는 SystemTap을 사용하여 실행 중인 Linux 시스템에서 버그 또는 성능 문제의 근본적인 원인을 확인할 수 있습니다.

애플리케이션 개발자는 SystemTap을 사용하여 Linux 시스템 내에서 애플리케이션이 작동하는 방식을 상세하게 모니터링할 수 있습니다.

37.1. SYSTEMTAP의 목적

SystemTap은 운영 체제의 활동을 연구하고 모니터링하는 데 사용할 수 있는 추적 및 프로빙 도구입니다(물론, 커널). SystemTap은 netstat,ps,top, iostat 등의 도구 출력과 유사한 정보를 제공합니다. 그러나 SystemTap은 수집된 정보에 대해 더 많은 필터링 및 분석 옵션을 제공합니다. SystemTap 스크립트에서 SystemTap이 수집하는 정보를 지정합니다.

SystemTap은 사용자에게 커널 활동을 추적하고 이 기능을 두 가지 특성과 결합하는 인프라를 제공하여 기존 Linux 모니터링 툴 세트를 보완하는 것을 목표로 합니다.

유연성

SystemTap 프레임워크를 사용하면 커널 공간에서 발생하는 다양한 커널 기능, 시스템 호출 및 기타 이벤트를 조사하고 모니터링하는 간단한 스크립트를 개발할 수 있습니다. 이를 통해 SystemTap은 자체 커널 특정 포렌식 및 모니터링 툴을 개발할 수 있는 시스템이므로 그렇게 많은 도구가 아닙니다.

사용 편의성

SystemTap을 사용하면 커널을 다시 컴파일하거나 시스템을 재부팅하지 않고도 커널 활동을 모니터링할 수 있습니다.

37.2. SYSTEMTAP 설치

SystemTap 사용을 시작하려면 필수 패키지를 설치합니다. 시스템에 여러 커널이 설치된 커널이 두 개 이상 설치된 커널에서 SystemTap을 사용하려면 *각* 커널 버전에 해당 필수 커널 패키지를 설치합니다.

사전 요구 사항

- 디버그 및 소스 리포지토리 활성화에 설명된 대로 디버그 리포지토리를 활성화했습니다.

절차

1. 필수 **SystemTap** 패키지를 설치합니다.

```
# yum install systemtap
```

2. 필수 커널 패키지를 설치합니다.

   a. **stap-prep** 사용 :

```
# stap-prep
```

   b. **stap-prep** 이 작동하지 않으면 필요한 커널 패키지를 수동으로 설치합니다.

```
# yum install kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -i)-$(uname -r) kernel-devel-$(uname -r)
```

   $(uname -i) 는 자동으로 시스템의 하드웨어 플랫폼으로 교체되고 **$(uname -r)** 는 실행 중인 커널 버전으로 자동 교체됩니다.

검증 단계

- 현재 **SystemTap**을 사용하여 조사할 커널이 사용 중인 경우 설치에 성공했는지 테스트합니다.

```
# stap -v -e 'probe kernel.function("vfs_read") {printf("read performed\n"); exit()}'
```

SystemTap 배포가 성공하면 다음과 유사한 출력이 생성됩니다.

```
Pass 1: parsed user script and 45 library script(s) in 340usr/0sys/358real ms.
Pass 2: analyzed script: 1 probe(s), 1 function(s), 0 embed(s), 0 global(s) in 290usr/260sys/568real ms.
Pass 3: translated to C into "/tmp/stapiArgLX/stap_e5886fa50499994e6a87aacdc43cd392_399.c" in 490usr/430sys/938real ms.
Pass 4: compiled C into "stap_e5886fa50499994e6a87aacdc43cd392_399.ko" in 3310usr/430sys/3714real ms.
Pass 5: starting run. (1)
read performed (2)
Pass 5: run completed in 10usr/40sys/73real ms. (3)
```

마지막 세 줄의 출력(**Pass 5**로 시작)은 다음을 나타냅니다.

1. **SystemTap**은 커널을 조사하고 계측을 실행하는 계측을 성공적으로 생성했습니다.
2. **SystemTap**은 지정된 이벤트를 감지했습니다(이 경우 **VFS** 읽기).
3. **SystemTap**은 유효한 핸들러를 실행했습니다(텍스트를 인쇄한 다음 오류 없이 종료).

37.3. SYSTEMTAP을 실행할 권한

SystemTap 스크립트를 실행하려면 높은 시스템 권한이 필요하지만 일부 인스턴스에서 권한이 없는 사용자는 시스템에서 **SystemTap** 계측을 실행해야 할 수 있습니다.

사용자가 루트 액세스없이 **SystemTap**을 실행할 수 있도록 하려면 다음 두 사용자 그룹에 사용자를 추가하십시오.

stapdev

이 그룹의 멤버는 **stap** 를 사용하여 **SystemTap** 스크립트를 실행하거나 **staprun** 을 사용하여 **SystemTap** 계측 모듈을 실행할 수 있습니다.

stap 를 실행하려면 **SystemTap** 스크립트를 커널 모듈로 컴파일하여 커널로 로드해야 합니다. 이를 위해서는 **stapdev** 멤버에게 부여된 시스템에 대한 상승된 권한이 필요합니다. 안타깝게도 이러한 권한은 **stapdev** 멤버에 대한 효과적인 **root** 액세스 권한도 부여합니다. 따라서 루트 액세스 권한으로 신뢰할 수 있는 사용자에게만 **stapdev** 그룹 멤버십을 부여합니다.

stapusr

이 그룹의 멤버는 **staprun** 만 사용하여 **SystemTap** 계측 모듈을 실행할 수 있습니다. 또한 **/lib/modules/*kernel_version*/systemtap/** 디렉터리에서 해당 모듈만 실행할 수 있습니다. 이 디렉터리는 **root** 사용자만 소유해야 하며 **root** 사용자만 쓸 수 있어야 합니다.

37.4. SYSTEMTAP 스크립트 실행

표준 입력 또는 파일에서 **SystemTap** 스크립트를 실행할 수 있습니다.

SystemTap 설치를 사용하여 배포되는 샘플 스크립트는 **/usr/share/systemtap/examples** 디렉터리에 있습니다.

사전 요구 사항

1. **SystemTap** 및 관련 필수 커널 패키지는 **Systemtap 설치에 설명된 대로** 설치됩니다.

2. 일반 사용자로 **SystemTap** 스크립트를 실행하려면 사용자를 **SystemTap** 그룹에 추가합니다.

```
# usermod --append --groups
stapdev,stapusr user-name
```

절차

- **SystemTap** 스크립트를 실행합니다.
  - 표준 입력에서 다음을 수행합니다.

```
# echo "probe timer.s(1) {exit()}" | stap -
```

이 명령은 **echo** 에서 표준 입력으로 전달된 스크립트를 실행하도록 **stap** 에 지시합니다. **stap** 옵션을 추가하려면 **-** 문자 앞에 삽입합니다. 예를 들어 이 명령의 결과를 보다 자세한 정보를 표시하려면 명령은 다음과 같습니다.

```
# echo "probe timer.s(1) {exit()}" | stap -v -
```

  - 파일에서 다음을 수행합니다.

```
# stap file_name
```

38장. SYSTEMTAP의 교차 계측

SystemTap의 교차 계측은 SystemTap 스크립트에서 SystemTap 스크립트에서 SystemTap을 완전히 배포하지 않은 다른 시스템에서 사용할 SystemTap 계측 모듈을 생성합니다.

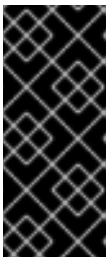
38.1. SYSTEMTAP 교차 계측

SystemTap 스크립트를 실행하면 커널 모듈이 해당 스크립트에서 빌드됩니다. 그런 다음 SystemTap은 모듈을 커널로 로드합니다.

일반적으로 SystemTap 스크립트는 SystemTap이 배포된 시스템에서만 실행할 수 있습니다. 10개의 시스템에서 SystemTap을 실행하려면 모든 시스템에 SystemTap을 배포해야 합니다. 이 방법이 적절하거나 바람직하지 않을 수도 있습니다. 예를 들어, 회사 정책은 특정 시스템에 컴파일러 또는 디버그 정보를 제공하는 패키지를 설치하지 못하도록 하여 SystemTap의 배포가 금지될 수 있습니다.

이 문제를 해결하려면 *교차 계측*을 사용합니다. 교차 계측은 다른 시스템에서 사용할 한 시스템에서 SystemTap 스크립트에서 SystemTap 계측 모듈을 생성하는 프로세스입니다. 이 프로세스는 다음과 같은 이점을 제공합니다.

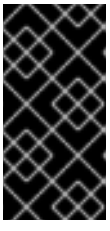
- 다양한 시스템의 커널 정보 패키지를 단일 호스트 시스템에 설치할 수 있습니다.



중요

커널 패키징 버그로 인해 설치를 방지할 수 있습니다. 이 경우 *호스트 시스템과 대상 시스템의* kernel-debuginfo 및 kernel -devel 패키지가 일치해야 합니다. 버그가 발생하면 https://bugzilla.redhat.com/ 에서 버그를 보고하십시오.

- 각 *대상 시스템에*는 생성된 SystemTap 계측 모듈을 사용하기 위해 하나의 패키지만 설치해야 합니다(system tap-runtime).



중요

*호스트 시스템*은 빌드된 *계측 모듈이* 작동하려면 *대상 시스템과* 동일한 Linux 배포판을 실행하고 동일한 아키텍처여야 합니다.

용어

계측 모듈

SystemTap 스크립트에서 빌드된 커널 모듈. **SystemTap** 모듈은 *호스트 시스템에* 빌드되며 *대상 시스템의 대상 커널에* 로드됩니다.

호스트 시스템

계측 모듈(**SystemTap** 스크립트에서)이 컴파일되어 *대상 시스템에* 로드되는 시스템입니다.

대상 시스템

계측 모듈이 빌드되는 시스템(**SystemTap** 스크립트에서).

대상 커널

대상 시스템의 커널입니다. *계측 모듈을* 로드하고 실행하는 커널입니다.

38.2. SYSTEMTAP의 교차 계측 초기화

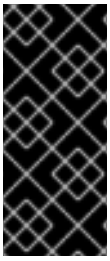
SystemTap의 교차 계측을 초기화하여 한 시스템에서 **SystemTap** 스크립트에서 **SystemTap** 계측 모듈을 빌드하고 **SystemTap**이 완전히 배포되지 않은 다른 시스템에서 사용합니다.

사전 요구 사항

- **SystemTap**은 Systemtap 설치에 설명된 대로 *호스트 시스템에* 설치됩니다.
- **systemtap-runtime** 패키지는 각 *대상 시스템에* 설치됩니다 :

```
# yum install systemtap-runtime
```

- *호스트 시스템과 대상 시스템은* 모두 동일한 아키텍처입니다.
- *호스트 시스템과 대상 시스템은* 모두 동일한 주요 **Red Hat Enterprise Linux** 버전(예: **Red Hat Enterprise Linux 8**)을 실행하고 있으며 다양한 마이너 버전(예: **8.1** 및 **8.2**)을 실행할 *수 있습니다*.



중요

커널 패키징 버그로 인해 여러 **kernel-debuginfo** 및 **kernel -devel** 패키지가 하나의 시스템에 설치되지 않을 수 있습니다. 이 경우 *호스트 시스템 및 대상 시스템의* 부 버전이 일치해야 합니다. 버그가 발생하면 **https://bugzilla.redhat.com/** 보고하십시오.

절차

1. *각 대상 시스템에서* 실행중인 커널 확인 :

```
$ uname -r
```

각 대상 시스템에 대해 이 단계를 반복합니다.

2. *호스트 시스템에서* **Systemtap** 설치에 설명된 방법으로 각 *대상 시스템의 대상 커널* 및 관련 패키지를 설치합니다.

3. *호스트 시스템에서* 계측 모듈을 빌드하고 이 모듈을 에 복사한 다음 *대상 시스템에서* 다음 중 하나를 실행합니다.

   a. 원격 구현 사용:

   ```
   # stap --remote target_system script
   ```

   이 명령은 *대상 시스템에서* 지정된 스크립트를 원격으로 구현합니다. 이 작업을 수행하려면 *호스트 시스템에서 대상 시스템에* ***SSH*** *연결을 수행할 수 있는지 확인해야* 합니다.

   b. 수동:

      i. *호스트 시스템에서* 계측 모듈을 빌드합니다.

      ```
      # stap -r kernel_version script -m module_name -p 4
      ```

      여기서 ***kernel_version*** 은 **1**단계에서 결정된 *대상 커널* 버전을 참조하고, *스크립트는 계측 모듈로 변환할 스크립트를 참조하며,* ***module _name*** 은 *계측 모듈* 의 원하는

이름입니다. p 4 옵션은 SystemTap에 컴파일된 모듈을 로드하고 실행하지 않도록 지시합니다.

ii.

***계측 모듈이* 컴파일되면 대상 시스템에 복사한 후 다음 명령을 사용하여 로드합니다.**

```
# staprun module_name.ko
```

39장. SYSTEMTAP을 사용하여 네트워크 활동 모니터링

systemtap-testsuite 패키지 를 설치한 후 /usr/share/systemtap/testsuite/systemtap.examples/ 디렉터리에서 사용할 수 있는 유용한 예제 SystemTap 스크립트를 사용하여 시스템의 네트워크 활동을 모니터링하고 조사할 수 있습니다.

39.1. SYSTEMTAP을 사용하여 네트워크 활동 프로파일링

nettop.stp 예제 SystemTap 스크립트를 사용하여 네트워크 활동을 프로파일링할 수 있습니다. 스크립트는 시스템에서 네트워크 트래픽을 생성하는 프로세스를 추적하고 각 프로세스에 대한 다음 정보를 제공합니다.

PID

나열된 프로세스의 ID입니다.

UID

사용자 ID. 사용자 ID 0은 root 사용자를 나타냅니다.

DEV

데이터를 보내거나 받는 데 사용한 프로세스의 이더넷 장치(예: eth0, eth1)는 무엇입니까.

XMIT_PK

프로세스에서 전송한 패킷 수입니다.

RECV_PK

프로세스에서 수신한 패킷 수입니다.

XMIT_KB

프로세스에서 보낸 데이터 양(킬로바이트)입니다.

RECV_KB

서비스에서 수신한 데이터 양(킬로바이트)입니다.

사전 요구 사항

- SystemTap 설치에 설명된 대로 SystemTap 을 설치했습니다.

절차

- **nettop.stp** 스크립트를 실행합니다.

```
# stap  --example nettop.stp
```

nettop.stp 스크립트는 **5**초마다 네트워크 프로필 샘플링을 제공합니다.

nettop.stp 스크립트의 출력은 다음과 유사합니다.

```
[...]
  PID   UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
    0     0 eth0         0       5       0       0 swapper
11178     0 eth0         2       0       0       0 synergyc
  PID   UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
 2886     4 eth0        79       0       5       0 cups-polld
11362     0 eth0         0      61       0       5 firefox
    0     0 eth0         3      32       0       3 swapper
 2886     4 lo           4       4       0       0 cups-polld
11178     0 eth0         3       0       0       0 synergyc
  PID   UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
    0     0 eth0         0       6       0       0 swapper
 2886     4 lo           2       2       0       0 cups-polld
11178     0 eth0         3       0       0       0 synergyc
 3611     0 eth0         0       1       0       0 Xorg
  PID   UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
    0     0 eth0         3      42       0       2 swapper
11178     0 eth0        43       1       3       0 synergyc
11362     0 eth0         0       7       0       0 firefox
 3897     0 eth0         0       1       0       0 multiload-apple
```

39.2. SYSTEMTAP을 사용하여 네트워크 소켓 코드에서 호출되는 함수 추적

socket-trace.stp 예제 **SystemTap** 스크립트를 사용하여 커널의 **net/socket.c** 파일에서 호출되는 함수를 추적할 수 있습니다. 이를 통해 각 프로세스가 커널 수준에서 네트워크와 상호 작용하는 방식을 자세히 파악하는 데 도움이 됩니다.

사전 요구 사항

- **SystemTap** 설치에 설명된 대로 **SystemTap** 을 설치했습니다.

절차

- **socket-trace.stp** 스크립트를 실행합니다.

```
# stap  --example socket-trace.stp
```

socket-trace.stp 스크립트의 출력에 대한 **3**초 발췌 내용은 다음과 유사합니다.

```
[...]
0 Xorg(3611): -> sock_poll
3 Xorg(3611): <- sock_poll
0 Xorg(3611): -> sock_poll
3 Xorg(3611): <- sock_poll
0 gnome-terminal(11106): -> sock_poll
5 gnome-terminal(11106): <- sock_poll
0 scim-bridge(3883): -> sock_poll
3 scim-bridge(3883): <- sock_poll
0 scim-bridge(3883): -> sys_socketcall
4 scim-bridge(3883):  -> sys_recv
8 scim-bridge(3883):   -> sys_recvfrom
12 scim-bridge(3883):-> sock_from_file
16 scim-bridge(3883):<- sock_from_file
20 scim-bridge(3883):-> sock_recvmsg
24 scim-bridge(3883):<- sock_recvmsg
28 scim-bridge(3883):   <- sys_recvfrom
31 scim-bridge(3883):  <- sys_recv
35 scim-bridge(3883): <- sys_socketcall
[...]
```

39.3. SYSTEMTAP을 사용하여 네트워크 패킷 드롭 모니터링

Linux의 네트워크 스택은 다양한 이유로 패킷을 폐기할 수 있습니다. 일부 **Linux** 커널에는 패킷이 폐기되는 위치를 추적하는 **tracepoint, kernel.trace("kfree_skb")'** 가 포함되어 있습니다.

dropwatch.stp SystemTap 스크립트는 **kernel.trace("kfree_skb")** 를 사용하여 패킷 삭제를 추적합니다. 스크립트는 **5**초마다 패킷 삭제 위치를 요약합니다.

사전 요구 사항

- **SystemTap** 설치에 설명된 대로 **SystemTap** 을 설치했습니다.

절차

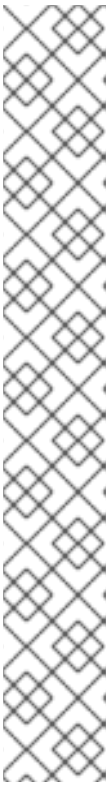
- **dropwatch.stp** 스크립트를 실행합니다.

-

```
# stap  --example dropwatch.stp
```

dropwatch.stp 스크립트를 **15**초 동안 실행하면 출력은 다음과 유사합니다.

```
Monitoring for dropped packets
51 packets dropped at location 0xffffffff8024cd0f
2 packets dropped at location 0xffffffff8044b472
51 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
97 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
Stopping dropped packet monitor
```



참고

패킷의 위치를 보다 의미 있게 만들려면 **/boot/System.map-$(uname -r)** 파일을 참조하십시오. 이 파일에는 각 함수의 시작 주소가 나열되므로 **dropwatch.stp** 스크립트의 주소를 특정 함수 이름에 매핑할 수 있습니다. **/boot/System.map-$(uname -r)** 파일의 스니펫을 고려할 때 **0xffffffff8024cd0f** 는 **unix_stream_recvmsg** 함수에 매핑되고 **0xffffff8044b472** 주소는 **arp_rcv**:

```
[...]
ffffffff8024c5cd T unlock_new_inode
ffffffff8024c5da t unix_stream_sendmsg
ffffffff8024c920 t unix_stream_recvmsg
ffffffff8024cea1 t udp_v4_lookup_longway
[...]
ffffffff8044addc t arp_process
ffffffff8044b360 t arp_rcv
ffffffff8044b487 t parp_redo
ffffffff8044b48c t arp_solicit
[...]
```

40장. SYSTEMTAP을 사용하여 커널 활동 프로파일링

다음 섹션에서는 기능 호출을 모니터링하여 커널 활동을 프로파일링하는 스크립트를 보여줍니다.

40.1. SYSTEMTAP을 사용하여 함수 호출 수

functioncallcount.stp SystemTap 스크립트를 사용하여 특정 커널 함수 호출 수를 계산할 수 있습니다. 이 스크립트를 사용하여 여러 커널 함수를 대상으로 지정할 수도 있습니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- **functioncallcount.stp** 스크립트를 실행합니다.

```
# stap --example functioncallcount.stp 'argument'
```

이 스크립트는 타겟 커널 기능을 인수로 사용합니다. 인수 와일드카드를 사용하여 특정 범위까지 여러 커널 함수를 대상으로 지정할 수 있습니다.

스크립트의 출력에는 알파벳순으로 호출되는 함수 이름과 샘플 시간 동안 호출된 횟수가 포함됩니다.

다음 예제를 고려하십시오.

```
# stap -w -v --example functioncallcount.stp "*@mm*.c" -c /bin/true
```

다음과 같습니다.

- **-w** : 경고를 표시하지 않습니다.
- **-v** : 시작 커널의 출력이 표시됩니다.

- **-c *명령* : 이 예제에서는 /bin/true인 명령을 실행하는 동안 함수 호출을 계산하도록 SystemTap에 지시합니다.**

  출력은 다음과 유사해야 합니다.

```
[...]
__vma_link 97
__vma_link_file 66
__vma_link_list 97
__vma_link_rb 97
__xchg 103
add_page_to_active_list 102
add_page_to_inactive_list 19
add_to_page_cache 19
add_to_page_cache_lru 7
all_vm_events 6
alloc_pages_node 4630
alloc_slabmgmt 67
anon_vma_alloc 62
anon_vma_free 62
anon_vma_lock 66
anon_vma_prepare 98
anon_vma_unlink 97
anon_vma_unlock 66
arch_get_unmapped_area_topdown 94
arch_get_unmapped_exec_area 3
arch_unmap_area_topdown 97
atomic_add 2
atomic_add_negative 97
atomic_dec_and_test 5153
atomic_inc 470
atomic_inc_and_test 1
[...]
```

40.2. SYSTEMTAP을 사용하여 함수 호출 추적

para-callgraph.stp SystemTap 스크립트를 사용하여 함수 호출 및 함수 반환을 추적할 수 있습니다.

사전 요구 사항

- **Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.**

절차

- **para-callgraph.stp 스크립트를 실행합니다.**

```
# stap --example para-callgraph.stp 'argument1' 'argument2'
```

스크립트 **para-callgraph.stp** 는 두 개의 명령줄 인수를 사용합니다.

1. 추적하려는 항목/종료 기능의 이름입니다.

2. 스레드별로 추적을 활성화 또는 비활성화하는 선택적 트리거 기능입니다. 트리거 함수가 아직 종료되지 않은 한 각 스레드에서 추적이 계속됩니다.

다음 예제를 고려하십시오.

```
# stap -wv --example para-callgraph.stp 'kernel.function("*@fs/proc.c*")' 'kernel.function("vfs_read")' -
c "cat /proc/sys/vm/* || true"
```

다음과 같습니다.

- **-w :** 경고를 표시하지 않습니다.

- **-v :** 시작 커널의 출력이 표시됩니다.

- **-c** *명령* **:** 이 예제에서는 **/bin/true**인 명령을 실행하는 동안 함수 호출을 계산하도록 **SystemTap**에 지시합니다.

출력은 다음과 유사해야 합니다.

```
[...]
   267 gnome-terminal(2921): <-do_sync_read return=0xfffffffffffffff5
   269 gnome-terminal(2921):<-vfs_read return=0xfffffffffffffff5
     0 gnome-terminal(2921):->fput file=0xffff880111eebbc0
     2 gnome-terminal(2921):<-fput
     0 gnome-terminal(2921):->fget_light fd=0x3 fput_needed=0xffff88010544df54
     3 gnome-terminal(2921):<-fget_light return=0xffff8801116ce980
     0 gnome-terminal(2921):->vfs_read file=0xffff8801116ce980 buf=0xc86504 count=0x1000
pos=0xffff88010544df48
     4 gnome-terminal(2921): ->rw_verify_area read_write=0x0 file=0xffff8801116ce980
ppos=0xffff88010544df48 count=0x1000
     7 gnome-terminal(2921): <-rw_verify_area return=0x1000
```

```
   12 gnome-terminal(2921): ->do_sync_read filp=0xffff8801116ce980 buf=0xc86504 len=0x1000
ppos=0xffff88010544df48
   15 gnome-terminal(2921): <-do_sync_read return=0xfffffffffffffff5
   18 gnome-terminal(2921):<-vfs_read return=0xfffffffffffffff5
    0 gnome-terminal(2921):->fput file=0xffff8801116ce980
```

40.3. SYSTEMTAP을 사용하여 커널 및 사용자 공간에 소비된 시간 결정

thread-times.stp SystemTap 스크립트를 사용하여 주어진 스레드가 커널 또는 사용자 공간에서 지출하는 시간을 확인할 수 있습니다.

사전 요구 사항

- **Systemtap** 설치에 설명된 대로 **SystemTap**을 설치했습니다.

절차

- **thread-times.stp** 스크립트를 실행합니다.

```
# stap --example thread-times.stp
```

이 스크립트는 **5**초 동안 **CPU** 시간을 사용하는 상위 **20**개의 프로세스와 샘플 중에 수행된 총 **CPU** 틱 수가 표시됩니다. 또한 이 스크립트의 출력은 각 프로세스가 사용된 **CPU** 시간과 해당 시간이 커널 공간 또는 사용자 공간에 사용되었는지 여부를 기록합니다.

```
tid   %user %kernel (of 20002 ticks)
  0   0.00%  87.88%
32169   5.24%   0.03%
9815   3.33%   0.36%
9859   0.95%   0.00%
3611   0.56%   0.12%
9861   0.62%   0.01%
11106   0.37%   0.02%
32167   0.08%   0.08%
3897   0.01%   0.08%
3800   0.03%   0.00%
2886   0.02%   0.00%
3243   0.00%   0.01%
3862   0.01%   0.00%
3782   0.00%   0.00%
21767   0.00%   0.00%
2522   0.00%   0.00%
3883   0.00%   0.00%
3775   0.00%   0.00%
3943   0.00%   0.00%
3873   0.00%   0.00%
```

40.4. SYSTEMTAP을 사용하여 폴링 애플리케이션 모니터링

timeout.stp SystemTap 스크립트를 사용하여 폴링 중인 애플리케이션을 식별하고 모니터링할 수 있습니다. 이렇게 하면 불필요하거나 과도한 폴링을 추적할 수 있으므로 **CPU** 사용량 및 전력 절감 측면에서 개선을 위한 영역을 정확하게 파악하는 데 도움이 됩니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- **timeout.stp** 스크립트를 실행합니다.

```
# stap --example timeout.stp
```

이 스크립트는 각 애플리케이션이 시간 경과에 따라 다음 시스템 호출을 사용하는 횟수를 추적합니다.

- **poll**
- 선택
- 기간
- **itimer**
- **futex**
- **nanosleep**
- 신호

이 예제 출력에서는 시스템 호출과 몇 번 사용된 프로세스를 확인할 수 있습니다.

```
uid |   poll  select   epoll  itimer   futex nanosle  signal| process
28937 | 148793      0      0    4727   37288      0       0| firefox
22945 |      0   56949      0       1       0      0       0| scim-bridge
    0 |      0       0      0   36414       0      0       0| swapper
4275 |  23140       0      0       1       0      0       0| mixer_applet2
4191 |      0   14405      0       0       0      0       0| scim-launcher
22941 |   7908       1      0      62       0      0       0| gnome-terminal
4261 |      0       0      0       2       0   7622       0| escd
3695 |      0       0      0       0       0   7622       0| gdm-binary
3483 |      0    7206      0       0       0      0       0| dhcdbd
4189 |   6916       0      0       2       0      0       0| scim-panel-gtk
1863 |   5767       0      0       0       0      0       0| iscsid
```

40.5. SYSTEMTAP을 사용하여 가장 자주 사용되는 시스템 호출 추적

topsys.stp SystemTap 스크립트를 사용하여 시스템에서 **5**초 간격으로 사용하는 상위 **20**개의 시스템 호출을 나열할 수 있습니다. 또한 해당 기간 동안 각 시스템 호출이 사용된 횟수가 나열됩니다.

사전 요구 사항

- **Systemtap** 설치에 설명된 대로 **SystemTap**을 설치했습니다.

절차

- **topsys.stp** 스크립트를 실행합니다.

```
# stap --example topsys.stp
```

다음 예제를 고려하십시오.

```
# stap -v --example topsys.stp
```

여기서 **-v**를 사용하면 시작 커널의 출력이 표시됩니다.

출력은 다음과 유사해야 합니다.

```
----------------------------------------------------------------
              SYSCALL      COUNT
```

```
          gettimeofday       1857
                read       1821
               ioctl       1568
                poll       1033
               close        638
                open        503
              select        455
               write        391
              writev        335
               futex        303
             recvmsg        251
              socket        137
       clock_gettime        124
      rt_sigprocmask        121
              sendto        120
           setitimer        106
                stat         90
                time         81
           sigreturn         72
               fstat         66
------------------------------------------------------------
```

40.6. SYSTEMTAP을 사용하여 프로세스당 시스템 호출 볼륨 추적

syscalls_by_proc.stp SystemTap 스크립트를 사용하여 가장 많은 시스템 호출 볼륨을 수행하는 프로세스를 확인할 수 있습니다. **20**개의 프로세스로 대부분의 시스템 호출을 수행합니다.

사전 요구 사항

- **Systemtap** 설치에 설명된 대로 **SystemTap**을 설치했습니다.

절차

- **syscalls_by_proc.stp** 스크립트를 실행합니다.

```
# stap --example syscalls_by_proc.stp
```

syscalls_by_proc.stp 스크립트의 출력은 다음과 유사합니다.

```
Collecting data... Type Ctrl-C to exit and display results
#SysCalls  Process Name
1577       multiload-apple
692        synergyc
408        pcscd
376        mixer_applet2
299        gnome-terminal
```

```
293      Xorg
206      scim-panel-gtk
95       gnome-power-man
90       artsd
85       dhcdbd
84       scim-bridge
78       gnome-screensav
66       scim-launcher
[...]
```

41장. SYSTEMTAP을 사용하여 디스크 및 I/O 활동 모니터링

다음 섹션에서는 디스크 및 I/O 활동을 모니터링하는 스크립트를 보여줍니다.

41.1. SYSTEMTAP을 사용하여 디스크 읽기/쓰기 트래픽 요약

disktop.stp SystemTap 스크립트를 사용하여 상위 디스크 읽기 및 쓰기를 수행하는 프로세스를 식별할 수 있습니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- **disktop.stp** 스크립트를 실행합니다.

```
# stap --example disktop.stp
```

스크립트는 **heaviest**가 디스크에서 읽거나 쓰는 상위 **10**개의 프로세스를 표시합니다.

출력에는 나열된 프로세스당 다음 데이터가 포함됩니다.

UID

사용자 **ID**. 사용자 **ID 0** 은 **root** 사용자를 나타냅니다.

PID

나열된 프로세스의 **ID**입니다.

PPID

나열된 프로세스의 상위 프로세스의 프로세스 **ID**입니다.

CMD

나열된 프로세스의 이름입니다.

장치

나열된 프로세스가 읽거나 에 쓰는 스토리지 장치는 무엇입니까.

T

나열된 프로세스에서 수행한 작업 유형입니다. 여기서 **W** 는 쓰기를 참조하며, **R** 은 읽기를 나타냅니다.

바이트

디스크에서 읽거나 쓴 데이터 양입니다.

disktop.stp 스크립트의 출력은 다음과 유사합니다.

```
[...]
Mon Sep 29 03:38:28 2008 , Average:  19Kb/sec, Read: 7Kb, Write: 89Kb
UID     PID    PPID                CMD   DEVICE    T    BYTES
0    26319   26294             firefox     sda5    W       90229
0     2758    2757      pam_timestamp_c     sda5    R        8064
0     2885       1             cupsd     sda5    W        1678
Mon Sep 29 03:38:38 2008 , Average:   1Kb/sec, Read: 7Kb, Write: 1Kb
UID     PID    PPID                CMD   DEVICE    T    BYTES
0     2758    2757      pam_timestamp_c     sda5    R        8064
0     2885       1             cupsd     sda5    W        1678
```

41.2. SYSTEMTAP을 사용하여 각 파일을 읽거나 쓰는 I/O 시간 추적

the iotime.stp SystemTap 스크립트를 사용하여 각 프로세스가 파일에서 읽거나 쓰는 데 걸리는 시간을 모니터링할 수 있습니다. 이를 통해 시스템에서 로드 속도가 느린 파일을 확인할 수 있습니다.

사전 요구 사항

- **Systemtap 설치**에 설명된 대로 **SystemTap**을 설치했습니다.

절차

- **the iotime.stp** 스크립트를 실행합니다.

```
# stap --example iotime.stp
```

이 스크립트는 시스템 호출을 열고, 닫히고, 읽고, 파일에 쓸 때마다 추적합니다. 각 파일에서

시스템 호출 액세스에 대해 이 명령은 모든 읽기 또는 쓰기가 완료되는 데 걸리는 마이크로초 수를 계산하고, 바이트 단위로, 파일에서 읽거나 쓰는 데이터의 양을 추적합니다.

출력에는 다음이 포함됩니다.

- 타임스탬프(마이크로초)
- 프로세스 ID 및 프로세스 이름
- 액세스 or iotime 플래그
- 액세스된 파일

프로세스가 데이터를 읽거나 쓸 수 있는 경우 한 쌍의 액세스 및 iotime 행이 함께 표시되어야 합니다. 액세스 행은 지정된 프로세스가 파일에 액세스하기 시작한 시간을 나타냅니다. 액세스 행의 끝은 읽거나 쓴 데이터의 양을 표시합니다. The iotime 행은 읽기 또는 쓰기를 수행하기 위해 프로세스가 소요한 시간(마이크로초)을 표시합니다.

the iotime.stp 스크립트의 출력은 다음과 유사합니다.

```
[...]
825946 3364 (NetworkManager) access /sys/class/net/eth0/carrier read: 8190 write: 0
825955 3364 (NetworkManager) iotime /sys/class/net/eth0/carrier time: 9
[...]
117061 2460 (pcscd) access /dev/bus/usb/003/001 read: 43 write: 0
117065 2460 (pcscd) iotime /dev/bus/usb/003/001 time: 7
[...]
3973737 2886 (sendmail) access /proc/loadavg read: 4096 write: 0
3973744 2886 (sendmail) iotime /proc/loadavg time: 11
[...]
```

41.3. SYSTEMTAP을 사용하여 누적 I/O 추적

traceio.stp SystemTap 스크립트를 사용하여 시스템에 대한 누적 I/O 양을 추적할 수 있습니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- traceio.stp 스크립트를 실행합니다.

```
# stap --example traceio.stp
```

이 스크립트는 시간 경과에 따라 I/O 트래픽을 생성하는 상위 10개의 실행 파일을 출력합니다. 또한 해당 실행 파일이 수행한 I/O 읽기 및 쓰기의 누적 양을 추적합니다. 이 정보는 1초 간격으로 내림차순으로 추적 및 인쇄됩니다.

traceio.stp 스크립트의 출력은 다음과 유사합니다.

```
[...]
          Xorg r:   583401 KiB w:        0 KiB
      floaters r:       96 KiB w:     7130 KiB
multiload-apple r:      538 KiB w:      537 KiB
          sshd r:       71 KiB w:       72 KiB
pam_timestamp_c r:      138 KiB w:        0 KiB
       staprun r:       51 KiB w:       51 KiB
         snmpd r:       46 KiB w:        0 KiB
         pcscd r:       28 KiB w:        0 KiB
    irqbalance r:       27 KiB w:        4 KiB
         cupsd r:        4 KiB w:       18 KiB
          Xorg r:   588140 KiB w:        0 KiB
      floaters r:       97 KiB w:     7143 KiB
multiload-apple r:      543 KiB w:      542 KiB
          sshd r:       72 KiB w:       72 KiB
pam_timestamp_c r:      138 KiB w:        0 KiB
       staprun r:       51 KiB w:       51 KiB
         snmpd r:       46 KiB w:        0 KiB
         pcscd r:       28 KiB w:        0 KiB
    irqbalance r:       27 KiB w:        4 KiB
         cupsd r:        4 KiB w:       18 KiB
```

41.4. SYSTEMTAP을 사용하여 특정 장치에서 I/O 활동 모니터링

traceio2.stp SystemTap 스크립트를 사용하여 특정 장치의 I/O 활동을 모니터링할 수 있습니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- **traceio2.stp** 스크립트를 실행합니다.

```
# stap --example traceio2.stp 'argument'
```

이 스크립트는 전체 장치 번호를 인수로 사용합니다. 이 번호를 찾으려면 다음을 사용할 수 있습니다.

```
# stat -c "0x%D" directory
```

어느 *디렉터리* 가 모니터링하려는 장치에 있습니다.

출력에는 다음이 포함됩니다.

- 읽기 또는 쓰기를 수행하는 프로세스의 이름 및 **ID**
- 실행 중인 함수(**vfs_read** 또는 **vfs_write**)
- 커널 장치 번호

 # stap traceio2.stp 0x805의 출력을 고려하십시오.

```
[...]
synergyc(3722) vfs_read 0x800005
synergyc(3722) vfs_read 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
[...]
```

41.5. SYSTEMTAP을 사용하여 파일에 읽기 및 쓰기 모니터링

inodewatch.stp SystemTap 스크립트를 사용하여 파일의 읽기 및 쓰기를 실시간으로 모니터링할 수

있습니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- inodewatch.stp 스크립트를 실행합니다.

```
# stap --example inodewatch.stp 'argument1' 'argument2' 'argument3'
```

inodewatch.stp 스크립트는 세 가지 명령줄 인수를 사용합니다.

1. 파일의 주요 장치 번호입니다.
2. 파일의 부 장치 번호입니다.
3. 파일의 inode 번호입니다.

다음 번호를 사용하여 얻을 수 있습니다.

```
# stat -c '%D %i' filename
```

여기서 ***filename*** 은 절대 경로입니다.

다음 예제를 고려하십시오.

```
# stat -c '%D %i' /etc/crontab
```

출력은 다음과 같아야 합니다.

```
805 1078319
```

다음과 같습니다.

- **805** 는 **base-16 (hexadecimal)** 장치 번호입니다. 마지막 두 자리는 부 장치 번호이며 나머지 숫자는 주 번호입니다.
- **1078319** 는 **inode** 번호입니다.

/etc/crontab 모니터링을 시작하려면 다음을 실행합니다.

```
# stap inodewatch.stp 0x8 0x05 1078319
```

처음 두 인수에서는 **base-16** 숫자에 **0x** 접두사를 사용해야 합니다.

출력에는 다음이 포함됩니다.

- 읽기 또는 쓰기를 수행하는 프로세스의 이름 및 **ID**
- 실행 중인 함수(**vfs_read** 또는 **vfs_write**)
- 커널 장치 번호

이 예제의 출력은 다음과 같아야 합니다.

```
cat(16437) vfs_read 0x800005/1078319
cat(16437) vfs_read 0x800005/1078319
```

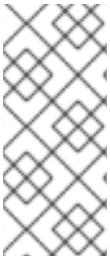
42장. BPF COMPILER COLLECTION을 사용하여 시스템 성능 분석

시스템 관리자는 BCC(BPF Compiler Collection) 라이브러리를 사용하여 Linux 운영 체제의 성능 분석 및 정보 수집을 위한 툴을 생성할 수 있습니다. 이러한 툴은 다른 인터페이스를 통해 확보하기 어려울 수 있습니다.

42.1. BCC 소개

BCC(BPF Compiler Collection)는 eBPF(extended Berkeley Packet Filter) 프로그램을 쉽게 생성할 수 있는 라이브러리입니다. eBPF 프로그램의 주요 유틸리티는 오버헤드나 보안 문제가 발생하지 않고 OS 성능 및 네트워크 성능을 분석하는 것입니다.

BCC는 사용자가 eBPF의 심층적인 기술 세부 사항을 알고 있어야 하는 필요성을 없애고, 미리 생성된 eBPF 프로그램을 사용하는 bcc-tools 패키지와 같이 많은 즉시 사용 가능한 시작점을 제공합니다.



참고

eBPF 프로그램은 디스크 I/O, TCP 연결 및 프로세스 생성과 같은 이벤트에서 트리거됩니다. 커널의 안전한 가상 시스템에서 실행되므로 커널이 충돌하거나 반복하거나 응답하지 않게 되는 프로그램이 발생할 가능성이 낮습니다.

42.2. BCC-TOOLS 패키지 설치

이 섹션에서는 BCC(BPF Compiler Collection) 라이브러리도 종속성으로 설치하는 bcc-tools 패키지를 설치하는 방법에 대해 설명합니다.

절차

1. bcc-tools 를 설치합니다.

```
# yum install bcc-tools
```

BCC 툴은 /usr/share/bcc/tools/ 디렉토리에 설치됩니다.

2. 선택적으로 툴을 검사합니다.

```
# ll /usr/share/bcc/tools/
```

```
...
-rwxr-xr-x. 1 root root  4198 Dec 14 17:53 dcsnoop
-rwxr-xr-x. 1 root root  3931 Dec 14 17:53 dcstat
-rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
-rw-r--r--. 1 root root  7105 Dec 14 17:53 deadlock_detector.c
drwxr-xr-x. 3 root root  8192 Mar 11 10:28 doc
-rwxr-xr-x. 1 root root  7588 Dec 14 17:53 execsnoop
-rwxr-xr-x. 1 root root  6373 Dec 14 17:53 ext4dist
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
...
```

위의 목록의 **doc** 디렉터리에는 각 툴에 대한 문서가 포함되어 있습니다.

42.3. 선택한 BCC-TOOLS를 성능 분석에 사용

이 섹션에서는 **BCC(BPF Compiler Collection)** 라이브러리에서 미리 생성된 특정 프로그램을 사용하여 이벤트별로 효율적이고 안전하게 시스템 성능을 분석하는 방법에 대해 설명합니다. **BCC** 라이브러리에서 미리 생성된 프로그램 집합은 추가 프로그램 생성의 예제 역할을 할 수 있습니다.

사전 요구 사항

- 설치된 **bcc-tools** 패키지
- 루트 권한

execsnoop를 사용하여 시스템 프로세스 검사

1. 하나의 터미널에서 **execsnoop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/execsnoop
```

2. 예를 들어 다른 터미널 실행에서 다음을 실행합니다.

```
$ ls /usr/share/bcc/tools/doc/
```

위의 경우 **ls** 명령의 수명이 짧은 프로세스가 생성됩니다.

3. **execsnoop** 를 실행하는 터미널에는 다음과 유사한 출력이 표시됩니다.

```
PCOMM PID    PPID   RET ARGS
ls    8382   8287     0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

execsnoop 프로그램은 시스템 리소스를 사용하는 각 새 프로세스의 출력 행을 인쇄합니다. **ls** 와 같이 매우 빨리 실행되는 프로그램의 프로세스도 감지하며 대부분의 모니터링 툴은 등록되지 않았습니다.

execsnoop 출력에는 다음 필드가 표시됩니다.

- **PCOMM -** 부모 프로세스 이름입니다**(ls)**
- **PID -** 프로세스 **ID**입니다**. (8382)**
- **PPID -** 상위 프로세스 **ID**입니다**. (8287)**
- **RET -** 프로그램 코드를 새 프로세스로 로드하는 **exec()** 시스템 호출**(0)**의 반환 값입니다.
- **ARGS -** 인수가 포함된 시작된 프로그램의 위치입니다.

execsnoop 에 대한 자세한 내용, 예제 및 옵션을 보려면 **/usr/share/bcc/tools/doc/execsnoop_example.txt** 파일을 참조하십시오.

exec() 에 대한 자세한 내용은 **exec(3)** 도움말 페이지를 참조하십시오.

opensnoop를 사용하여 명령이 여는 파일을 추적합니다.

1. 하나의 터미널에서 **opensnoop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/opensnoop -n uname
```

위의 출력은 **uname** 명령의 프로세스에 의해서만 열려 있는 파일의 출력을 인쇄합니다.

2. 다른 터미널에서 다음을 실행합니다.

```
$ uname
```

위의 명령은 다음 단계에서 캡처된 특정 파일을 엽니다.

3. **opensnoop**를 실행하는 터미널에는 다음과 유사한 출력이 표시됩니다.

```
PID    COMM  FD ERR PATH
8596   uname  3  0   /etc/ld.so.cache
8596   uname  3  0   /lib64/libc.so.6
8596   uname  3  0   /usr/lib/locale/locale-archive
...
```

opensnoop 프로그램은 전체 시스템에서 **open()** 시스템 호출을 감시하고, 그 과정에서 **uname** 이 열려고 시도한 각 파일의 출력을 출력합니다.

opensnoop 출력에는 다음 필드가 표시됩니다.

- **PID -** 프로세스 **ID**입니다. **(8596)**
- **COMM -** 프로세스 이름입니다(동일하지 않음)
- **fd -** 파일 설명자 **- open()**가 열려 있는 파일을 참조하도록 반환되는 값입니다. **(3)**
- **ERR -** 모든 오류.
- **PATH -** 열기를 시도한 파일의 위치입니다.

명령이 존재하지 않는 파일을 읽으려고 하면 **FD** 열은 **-1** 을 반환하고 **ERR** 열은 관련 오류에 해당하는 값을 출력합니다. 그 결과 **opensnoop** 는 제대로 작동하지 않는 애플리케이션을 식별하는 데 도움이 될 수 있습니다.

opensnoop 에 대한 자세한 내용, 예제 및 옵션을 보려면

/usr/share/bcc/tools/doc/opensnoop_example.txt 파일을 참조하십시오.

open() 에 대한 자세한 내용은 **open(2)** 매뉴얼 페이지를 참조하십시오.

biotop을 사용하여 디스크의 **I/O** 작업 검사

1. 하나의 터미널에서 **biotop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/biotop 30
```

명령을 사용하면 디스크에서 **I/O** 작업을 수행하는 최상위 프로세스를 모니터링할 수 있습니다. 인수를 사용하면 명령이 **30**초 요약을 생성할 수 있습니다.

참고

인수를 제공하지 않으면 기본적으로 출력 화면이 **1**초마다 새로 고쳐집니다.

2. 다른 터미널의 예를 들면 다음과 같습니다.

```
# dd if=/dev/vda of=/dev/zero
```

위의 명령은 로컬 하드 디스크 장치에서 콘텐츠를 읽고 **/dev/zero** 파일에 출력을 씁니다. 이 단계에서는 **biotop** 을 설명하는 특정 **I/O** 트래픽을 생성합니다.

3. **biotop** 을 실행하는 터미널은 다음과 유사한 출력을 보여줍니다.

```
PID    COMM             D MAJ MIN DISK       I/O  Kbytes     AVGms
9568   dd               R 252 0   vda      16294 14440636.0  3.69
48     kswapd0          W 252 0   vda       1763 120696.0    1.65
7571   gnome-shell      R 252 0   vda        834 83612.0     0.33
1891   gnome-shell      R 252 0   vda       1379 19792.0     0.15
7515   Xorg             R 252 0   vda        280  9940.0     0.28
7579   llvmpipe-1       R 252 0   vda        228  6928.0     0.19
9515   gnome-control-c  R 252 0   vda         62  6444.0     0.43
8112   gnome-terminal-  R 252 0   vda         67  2572.0     1.54
7807   gnome-software   R 252 0   vda         31  2336.0     0.73
9578   awk              R 252 0   vda         17  2228.0     0.66
7578   llvmpipe-0       R 252 0   vda        156  2204.0     0.07
9581   pgrep            R 252 0   vda         58  1748.0     0.42
7531   InputThread      R 252 0   vda         30  1200.0     0.48
```

```
7504   gdbus          R 252 0   vda        3  1164.0     0.30
1983   llvmpipe-1      R 252 0   vda        39   724.0     0.08
1982   llvmpipe-0      R 252 0   vda        36   652.0     0.06
...
```

biotop 출력에는 다음 필드가 표시됩니다.

- **PID -** 프로세스 **ID**입니다. **(9568)**
- **COMM -** 프로세스 이름입니다. **(dd)**
- **DISK -** 읽기 작업을 수행하는 디스크입니다 **(vda)**
- **I/O -** 수행된 읽기 작업 수입니다. **(16294)**
- **kbytes -** 읽기 작업에서 얻은 **K**바이트 양입니다. **(14,440,636)**
- **mms -** 평균 읽기 작업 **I/O** 시간입니다. **(3.69)**

biotop 에 대한 자세한 내용, 예제 및 옵션을 보려면 **/usr/share/bcc/tools/doc/biotop_example.txt** 파일을 참조하십시오.

dd 에 대한 자세한 내용은 **dd(1)** 도움말 페이지를 참조하십시오.

예기치 않게 느린 파일 시스템 작업 노출을 위해 xfsslower 사용

1. 한 터미널에서 **xfsslower** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/xfsslower 1
```

위의 명령은 **XFS** 파일 시스템이 읽기, 쓰기, 열기 또는 동기화**(fsync)** 작업을 수행하는 데 소비하는 시간을 측정합니다. **1** 인수를 사용하면 프로그램이 **1ms**보다 느린 작업만 표시합니다.

참고

인수를 지정하지 않으면 기본적으로 **xfsslower** 는 **10ms**보다 느리게 작업을 표시합니다.

2. 다른 터미널에서 다음을 실행합니다.

```
$ vim text
```

위의 명령은 **vim** 편집기에서 텍스트 파일을 생성하여 **XFS** 파일 시스템과의 특정 상호 작용을 시작합니다.

3. **xfsslower**를 실행하는 터미널에는 이전 단계에서 파일을 저장할 때 유사한 내용이 표시됩니다.

```
TIME     COMM          PID    T BYTES   OFF_KB   LAT(ms) FILENAME
13:07:14 b'bash'       4754   R 256     0          7.11 b'vim'
13:07:14 b'vim'        4754   R 832     0          4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'        4754   R 32      20         1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'        4754   R 1982    0          2.30 b'vimrc'
13:07:14 b'vim'        4754   R 1393    0          2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'        4754   S 0       0          6.71 b'text'
13:07:45 b'pool'       2588   R 16      0          5.58 b'text'
...
```

위의 각 행은 파일 시스템에서 특정 임계값보다 더 많은 시간이 걸리는 작업을 나타냅니다. **xfsslower** 는 예기치 않게 느린 작업을 수행할 수 있는 가능한 파일 시스템 문제를 노출하는 데 효과적입니다.

xfsslower 출력에는 다음 필드가 표시됩니다.

- **COMM -** 프로세스 이름입니다. **(b'bash')**
- **t -** 작업 유형입니다. **(R)**
  - 읽기

- w rite
- s ync

- OFF_KB - KB 단위의 파일 오프셋. (0)
- FILENAME - 읽기, 쓰기 또는 동기화되는 파일입니다.

xfsslower 에 대한 자세한 내용, 예제 및 옵션을 보려면 /usr/share/bcc/tools/doc/xfsslower_example.txt 파일을 참조하십시오.

fsync 에 대한 자세한 내용은 fsync(2) 매뉴얼 페이지를 참조하십시오.