Red Hat Enterprise Linux 7
성능 조정 가이드

Red Hat Enterprise Linux 7에서 서브시스템 처리량 최적화

Laura Bailey
Red Hat Subject Matter Experts

Red Hat Enterprise Linux 7 성능 조정 가이드

Red Hat Enterprise Linux 7에서 서브시스템 처리량 최적화

Laura Bailey
Red Hat Customer Content Services
lbailey@redhat.com

Red Hat Subject Matter Experts

법적 공지

초록

Red Hat Enterprise Linux 7 성능 조정 가이드에서는 Red Hat Enterprise Linux 7 성능을 최적화하는 방법에 대해 설명합니다. 이 문서에서는 Red Hat Enterprise Linux 7의 성능 관련 업그레이드에 대해서도 설명합니다. 성능 조정 가이드에서는 실제 테스트 및 검증된 절차를 제공하지만 프로덕션 환경에 있는 시스템에 적용하기 전 테스트 환경에서 계획된 모든 설정을 구성 및 테스트해야 합니다. 튜닝 전 모든 데이터 및 설정 내용을 백업해 두시는 것이 좋습니다.

차례

1장. Red Hat Enterprise Linux 7에서 성능 특징

다음 부분에서는 Red Hat Enterprise Linux 7에서의 성능 관련 변경 사항에 대해 간략하게 설명합니다.

1.1. 7.1에서 새로운 기능

- 커널 유휴 상태 밸런스 메커니즘에서는 동일한 CPU의 대기열에 있는 작업이 지연될 가능성이 적어지고 유휴 상태 밸런스가 사용되고 있는 경우 처리 대기 시간이 감소됩니다.
- **swapon** 명령에 새로운 ***--discard*** 매개 변수가 추가되었습니다. 이를 통해 관리자는 스왑시 사용할 삭제 정책을 선택할 수 있으므로 유연성을 확장하고 SSD (Solid State Disks)의 성능을 개선시킬 수 있습니다. 기본적으로 ***--discard*** 매개 변수는 **swapon** 실행 시 전체 스왑 영역을 삭제하고 스왑 중 여유 스왑 페이지를 삭제한 후 재사용을 실행합니다. 보다 자세한 내용은 **swapon** man 페이지 (**man swapon**)에서 참조하십시오.
- tuned 프로파일에서 ***cmdline*** 매개 변수 및 커널 매개 변수 목록을 지정하여 커널 명령행 매개 변수를 변경하도록 **tuned** 프로파일을 설정할 수 있습니다. ***cmdline***로 설정된 모든 커널 매개 변수는 재부팅 후 적용됩니다.

1.2. 7.0에서 새로운 기능

- 이 문서는 Red Hat Enterprise Linux 7에 맞게 완전히 재구성 및 재작성되었습니다.
- Red Hat Enterprise Linux 7에서 **deadline**은 기본 I/O 스케줄러로 **cfq**로 변경되었습니다. 이러한 변경으로 인해 대부분의 일반적인 사용에서의 성능이 향상되었습니다.
- XFS 파일 시스템은 기본 파일 시스템으로 ext4로 변경되어 최대 파일 시스템 지원 크기는 500 TB이며 최대 파일 오프셋은 8 EB (스파스 파일)입니다. XFS 튜닝 권장 사항이 명료하게 업데이트되었습니다.
- ext4 파일 시스템의 최대 파일 시스템 지원 크기는 50 TB이며 최대 파일 크기는 16 TB입니다. 이에 맞게 튜닝 권장 사항이 업데이트되었습니다. ext2 및 ext3 파일 시스템 지원은 ext4 드라이버에 의해 제공됩니다.
- btrfs 파일 시스템은 기술 프리뷰로 제공됩니다.
- Red Hat Enterprise Linux 7에는 GFS2 관련 일부 마이너 성능 개선 사항이 포함되어 있습니다.
- **Tuna**는 설정 파일을 지원하며 **tuned** 프로파일을 추가 및 저장할 수 있도록 업데이트되었습니다. 이러한 업데이트된 버전은 이벤트 기반 샘플링을 사용하여 보다 적은 프로세서 리소스를 소비하게 됩니다. 그래픽 버전도 업데이트되어 실시간 모니터링이 가능합니다. **Tuna**는 2.4절. “Tuna”, 3.3.8절. “Tuna를 사용하여 CPU, 스레드, 인터럽트 친화도 설정 ”, A.2절. “Tuna”에 문서화되어 있습니다.
- **tuned**의 기본 프로파일은 **throughput-performance**입니다. 이는 삭제된 **enterprise-storage** 프로파일을 대신합니다. 네트워킹 및 가상화의 새로운 프로파일도 추가되었습니다. 또한 **tuned**는 쉘 스크립트 호출 및 ***includes*** 기능을 제공합니다.
- **tuned-adm** 도구는 **recommend** 하위 명령을 제공하여 시스템에 적합한 튜닝 프로파일을 추천합니다. 또한 이는 설치 시 시스템에 기본 프로파일을 설정하여 기본 프로파일로 되돌아갈 때 사용할 수 있습니다.
- Red Hat Enterprise Linux 7에서는 자동 NUMA 밸런싱 기능을 제공합니다. 커널은 자동으로 어떤 메모리 페이지 프로세스가 많이 사용되고 있는 지를 감지하여 NUMA 노드를 통해 스레드 및 메모리를 그룹화합니다. 커널은 스레드를 재조정하고 메모리를 이전하여 최적의 NUMA 조정 및 성능을 위해 시스템의 밸런스를 유지합니다.
- 파일 시스템 제한/장벽을 활성화하는 성능 패널티는 3% 미만으로 미미합니다. 따라서 **tuned** 프로파일은 더이상 파일 시스템 제한/장벽을 비활성화하지 않습니다.

- OProfile은 새로운 **`operf`** 도구를 사용하여 Linux Performance Events 서브 시스템을 기반으로 하는 프로파일링을 지원합니다. 이러한 새로운 도구는 **`opcontrol`** 데몬 대신에 데이터를 수집하는데 사용할 수 있습니다.
- 시스템에서 프로세스 그룹에 리소스를 할당하기 위해 Control 그룹을 계속 사용할 수 있습니다. Red Hat Enterprise Linux 7 구현에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에 있는 *Red Hat Enterprise Linux 7 리소스 관리 가이드*에서 참조하십시오.

2장. 성능 모니터링 도구

다음 부분에서는 Red Hat Enterprise Linux 7에서 사용 가능한 일부 성능 모니터링 및 설정 도구에 대해 간략하게 설명합니다. 경우에 따라 도구 사용 방법 및 실제 도구 사용 예와 같은 내용을 자세히 설명합니다.

다음의 지식 베이스 문서에서 Red Hat Enterprise Linux에서 사용하기에 적합한 성능 모니터링 도구의 전체 목록을 확인합니다: https://access.redhat.com/site/solutions/173863.

2.1. /proc

proc "파일 시스템"은 Linux 커널의 현재 상태를 나타내는 파일의 계층 구조로된 디렉토리입니다. 이를 통해 사용자 및 애플리케이션은 시스템의 커널 상태를 확인할 수 있습니다.

proc 디렉토리에는 시스템 하드웨어 및 현재 실행 중인 프로세스에 대한 정보가 들어 있습니다. **/proc/sys**에 있는 파일의 대부분은 읽기 전용이지만 일부 파일 (/proc/sys에 있는 파일)은 사용자 및 애플리케이션에 의해 변경되어 커널에 설정 변경을 전달할 수 있습니다.

proc 디렉토리에 있는 파일 확인 및 편집에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 Red Hat Enterprise Linux 7 시스템 관리자 참조 가이드에서 확인하십시오.

2.2. GNOME 시스템 모니터

GNOME 데스크탑 환경에는 그래픽 도구, 시스템 모니터링이 포함되어 있어 시스템 동작을 모니터링 및 수정할 수 있습니다. 시스템 모니터링을 통해 기본 시스템 정보를 표시하고 시스템 프로세스 및 리소스 또는 파일 시스템 사용량 등을 모니터링할 수 있습니다.

시스템 모니터링에는 네 개의 탭이 있으며 각 탭에는 시스템에 관한 다른 정보가 표시됩니다.

시스템

시스템의 하드웨어 및 소프트웨어에 관한 기본 정보를 표시합니다.

프로세스

실행 중인 프로세스와 이러한 프로세스 간의 관계에 대한 자세한 정보를 표시합니다. 표시된 프로세스는 특정 프로세스를 보다 쉽게 검색할 수 있도록 필터링할 수 있습니다. 또한 표시된 프로세스에서 시작, 중지, 종료, 우선 순위 변경과 같은 작업을 수행할 수 있습니다.

리소스

현재 CPU 사용 시간, 메모리, 스왑 공간 사용량, 네트워크 사용량을 표시합니다.

파일 시스템

파일 시스템 유형, 마운트 지점, 메모리 사용량과 같은 기본적인 정보를 표시하고 마운트된 모든 파일 시스템을 나열합니다.

시스템 모니터링을 시작하려면 Super 키를 눌러 작업 개요 화면으로 들어가서 *시스템 모니터링*을 입력하고 Enter를 누릅니다.

시스템 모니터링에 대한 보다 자세한 내용은 애플리케이션에 있는 도움말 메뉴나 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에 있는 Red Hat Enterprise Linux 7 *시스템 관리자 가이드*에서 참조하십시오.

2.3. PCP (Performance Co-Pilot)

Red Hat Enterprise Linux 7에서는 시스템 수준의 성능 측정을 위한 값을 취득, 저장, 분석할 수 있는 도구, 서비스 라이브러리 모음, PCP (Performance Co-Pilot)를 지원합니다. 경량의 분산된 아키텍처는 복잡한 시스템의 중앙 집중식 분석에 적합합니다. 성능 측정 기준은 Python, Perl, C++, C 인터페이스를 사용하여 추가할 수 있습니다. 분석 도구는 클라이언트 API (Python, C++, C)를 직접 사용할 수 있고 웹 애플리케이션은 JSON 인터페이스를 사용하여 사용 가능한 모든 성능 데이터를 검색할 수 있습니다.

pcp 패키지는 명령행 도구 및 기본적인 기능을 제공합니다. 그래픽 도구에는 *pcp-gui* 패키지가 필요합니다.

Red Hat 고객 포털에는 PCP 사용 방법에 대한 몇몇 유용한 문서가 있습니다. 이러한 문서의 인덱스는 https://access.redhat.com/articles/1145953에서 참조하십시오.

또한 *pcp-doc* 패키지는 포괄적인 문서를 제공하며 기본적으로 **/usr/share/doc/pcp-doc**에 설치되어 있습니다. PCP는 각 도구에 대한 man 페이지도 제공합니다. 명령행에 **man *toolname***을 입력하여 해당 도구의 man 페이지를 확인합니다.

2.4. Tuna

Tuna는 스케줄러 정책, 스레드 우선 순위, CPU, 인터럽트 친화도와 같은 설정 상세 정보를 조정합니다. *tuna* 패키지는 명령행 도구 및 동일한 기능이 있는 그래픽 인터페이스를 제공합니다.

3.3.8절. “Tuna를 사용하여 CPU, 스레드, 인터럽트 친화도 설정 ”에서는 명령행에서 Tuna로 시스템을 설정하는 방법을 설명합니다. Tuna 사용 방법에 대한 자세한 내용은 A.2절. “Tuna” 또는 man 페이지를 참조하십시오:

```
$ man tuna
```

2.5. 내장된 명령행 도구

Red Hat Enterprise Linux 7에서는 명령행에서 시스템을 모니터링할 수 있는 여러 도구를 제공하며 이를 통해 런레벨 5 밖에서 시스템을 모니터링할 수 있습니다. 다음 부분에서는 각 도구에 대해 간략하게 설명하고 각 도구를 사용할 수 있는 위치 및 사용 방법에 대한 자세한 정보가 있는 링크를 알려드립니다.

2.5.1. top

procps-ng 패키지에서 제공하는 top 도구는 실행 중인 시스템에서 프로세스를 동적으로 표시합니다. 시스템 요약 및 Linux 커널에서 현재 관리되는 작업 목록을 포함하여 다양한 정보를 표시할 수 있습니다. 프로세스를 조정하고 시스템 재부팅 후 설정 변경 사항을 영구적으로 유지할 수 있는 한정된 기능도 제공합니다.

기본적으로 프로세스는 CPU 사용량에 따라 순서대로 표시되므로 가장 많은 리소스를 소비하는 프로세스를 쉽게 찾을 수 있습니다. top으로 표시되는 정보 및 동작 모두는 필요에 따라 다른 사용량에 중점을 둘 수 있도록 고급 설정 가능합니다.

top 사용에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man top
```

2.5.2. ps

procps-ng 패키지에서 제공되는 ps 도구는 선택한 실행 중인 프로세스 그룹의 스냅샷을 가져옵니다. 기본적으로 그룹은 현재 사용자가 소유하고 있고 ps가 실행되고 있는 터미널에 연결된 프로세스로 제한됩니다.

ps는 top 보다 더 자세한 프로세스 정보를 제공하지만 이는 기본적으로 프로세스 ID 순으로 데이터의 스냅샷을 제공합니다.

ps 사용에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man ps
```

2.5.3. 가상 메모리 통계 (vmstat)

가상 메모리 통계 도구 (vmstat)는 시스템 프로세스, 메모리, 페이징, 입/출력 차단, 인터럽트, CPU 동작에 관한 인스턴스 보고서를 제공합니다. Vmstat를 통해 샘플링 간격을 설정할 수 있기 때문에 거의 실시간으로 시스템 동작을 관찰할 수 있습니다.

vmstat는 *procps-ng* 패키지로 제공됩니다. vmstat에 관한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man vmstat
```

2.5.4. sar (System Activity Reporter)

sar (System Activity Reporter)는 최근의 시스템 동작에 대한 정보를 수집 및 보고합니다. 기본 출력에서는 하루의 시작 (00:00:00 시스템 클럭에 따라)에서 10 분 간격으로 현재의 CPU 사용량을 표시합니다.

`-i` 옵션을 사용하여 초 단위로 간격을 설정할 수 있습니다. 예를 들어 **`sar -i 60`**으로 설정하면 sar는 분 마다 CPU 사용량을 확인합니다.

sar는 수동으로 시스템 동작에 대한 보고서를 주기적으로 작성하기 위해 top 대신 사용할 수 있는 유용한 도구입니다. *sysstat* 패키지에서 제공됩니다. sar 사용에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man sar
```

2.6. tuned 및 tuned-adm

tuned는 튜닝 데몬으로 튜닝 프로파일을 설정하여 운영 체제가 특정 작업에서 보다 나은 성능을 발휘할 수 있게 합니다. tuned-adm은 명령행 도구로 이를 통해 사용자는 다른 튜닝 프로파일로 전환할 수 있습니다.

일반적인 사용 사례에 사용할 수 있도록 미리 정의된 프로파일이 포함되어 있지만 tuned-adm을 통해 사용자 정의 프로파일을 설정할 수 있으며 이는 미리 정의된 프로파일을 기반으로 사용자 정의하거나 처음 부터 새로 정의할 수 있습니다. Red Hat Enterprise Linux 7에서 기본 프로파일은 **`throughput-performance`**입니다.

tuned-adm에서 제공하는 프로파일은 절전 프로파일 및 성능 강화 프로 파일의 두 개의 카테고리로 나뉘어져 있습니다. 성능 강화 프로파일은 다음과 같은 사항에 중점을 두고 있습니다:

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

tuned 활성화 방법에 대한 보다 자세한 내용은 A.5절. “tuned”에서 참조하십시오.

tuned-adm에서 제공되는 성능 강화 프로파일에 대한 보다 자세한 내용은 A.6절. “tuned-adm”에서 참조하십시오.

tuned-adm에서 제공되는 절전 프로파일에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에 있는 Red Hat Enterprise Linux 7 Power Management Guide에서 참조하십시오.

tuned 및 tuned-adm 사용에 대한 보다 자세한 내용은 해당 man 페이지에서 참조하십시오:

```
$ man tuned
```

```
$ man tuned-adm
```

2.7. perf

perf 도구는 하드웨어 성능 카운터 및 커널 추적 포인트를 사용하여 시스템의 다른 명령이나 애플리케이션의 영향을 추적합니다. 여러 perf 하위 명령은 일반적인 성능 이벤트의 통계를 기록 및 표시가거나 기록된 데이터를 분석 및 보고합니다.

perf 및 perf 하위 명령에 대한 보다 자세한 내용은 A.7절. “perf”에서 참조하십시오.

http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에 있는 Red Hat Enterprise Linux 7 *Developer Guide*에서도 자세한 내용을 참조할 수 있습니다.

2.8. turbostat

Turbostat는 *kernel-tools* 패키지에서 제공합니다. Intel® 64 프로세서의 프로세서 토콜로지, 주파수, 유휴 전력 상태, 온도, 전력 사용량 등을 보고합니다.

Turbostat는 전력 사용량이나 유휴 시간이 비효율적인 서버를 식별하는데 유용합니다. 또한 시스템에서의 시스템 관리 인터럽트 (SMI)의 비율을 식별하는데 유용하며 전력 관리 튜닝 효과를 확인하는데 사용할 수 있습니다.

Turbostat를 실행하려면 root 권한이 필요합니다. 또한 다음과 같은 프로세서 지원이 필요합니다:

- 불변 타임 스탬프 카운터
- APERF 모델 별 레지스터
- MPERF 모델 별 레지스터

turbostat 출력 및 출력을 읽는 방법에 대한 보다 자세한 내용은 A.11절. “turbostat”에서 참조하십시오.

turbostat에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오.

```
$ man turbostat
```

2.9. iostat

iostat 도구는 *sysstat* 패키지에서 제공됩니다. 관리자가 물리적 디스크 사이의 입/출력 로드 밸런스를 저장하기 위한 방법을 결정할 때 사용할 수 있는 시스템의 입/출력 장치 로드를 보고 및 모니터링합니다. 이는 iostat가 마지막으로 실행된 때 또는 시작 시점에서의 프로세서 또는 장치 사용량을 보고합니다. iostat man 페이지에 지정된 매개 변수를 사용하여 특정 장치에서의 이러한 출력 보고를 중점적으로 모니터링할 수 있습니다:

```
$ man iostat
```

2.10. irqbalance

irqbalance는 시스템 성능을 향상시키기 위해 프로세스의 하드웨어 인터럽트를 분산시키기 위한 명령행 도구

입니다. **irqbalance**에 대한 자세한 내용은 A.1절. “irqbalance” 또는 man 페이지에서 참조하십시오:

```
$ man irqbalance
```

2.11. ss

ss는 소켓에 대한 통계 정보를 출력하는 명령행 유틸리티로 이를 통해 관리자는 장치 성능을 평가할 수 있습니다. 기본적으로 ss는 연결된 오픈 비수신 TCP 소켓을 나열하지만 특정 소켓에 대한 통계를 필터링하는데 관리자에게 유용한 여러 옵션도 제공합니다.

Red Hat은 Red Hat Enterprise Linux 7에서 netstat의 ss를 사용할 것을 권장합니다.

일반적으로 사용되는 **`ss -tmpie`**는 TCP 소켓, 메모리 사용량, 소켓을 사용하는 프로세스에 관한 자세한 정보(내부 정보 포함)를 표시합니다.

ss는 *iproute* 패키지에서 제공됩니다. 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man ss
```

2.12. numastat

numastat 도구는 프로세스의 메모리 통계 및 NUMA 노드 별 운영 체제를 표시합니다.

기본적으로 **numastat**는 커널 메모리 할당에서 노드 별 NUMA 히트와 미스 시스템 통계를 표시합니다. 최적의 성능은 높은 **`numa_hit`** 값과 낮은 **`numa_miss`** 값으로 표시됩니다. **Numastat** 는 다수의 명령행 옵션도 제공하며 이는 시스템의 NUMA 노드에서 시스템 및 프로세스 메모리가 분산된 방법을 표시합니다.

메모리가 할당된 노드와 동일한 노드에서 프로세스 스레드가 실행되고 있는지를 확인하려면 CPU 당 **top** 출력을 사용하여 노드 별 **numastat** 출력을 상호 참조합니다.

Numastat는 *numactl* 패키지에서 제공됩니다. numastat 사용 방법에 대한 보다 자세한 내용은 A.12절. “numastat”에서 참조하십시오. numastat에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man numastat
```

2.13. numad

numad는 자동 NUMA 친화도 관리 데몬입니다. 이는 시스템의 NUMA 토폴로지 및 리소스 사용량을 모니터링하여 동적으로 NUMA 리소스 할당 및 관리 (즉 시스템 성능)를 향상시킵니다. 시스템 작업량에 따라 numad는 성능 기준의 최대 50% 까지 성능을 개선시킬 수 있습니다. 또한 이는 사전 배치 조언 서비스를 통해 여러 작업 관리 시스템 쿼리에 대해 적절한 CPU와 메모리 리소스의 초기 바인딩을 제공할 수 있습니다.

numad는 **`/proc`** 파일 시스템에서 주기적으로 정보에 액세스하여 각 노드에서 사용 가능한 시스템 리소스를 모니터링합니다. 일정 수준의 지정된 리스소 사용량을 유지하며 필요에 따라 NUMA 노드 간 프로세스를 이동하여 리소스 할당을 재조정합니다. numad는 시스템의 NUMA 노드의 하위 집합에서 중요한 프로세스를 배치 및 격리하여 최적의 NUMA 성능을 구현할 수 있게 합니다.

numad는 주로 상당한 리소스 양을 소비하고 전체 시스템 리소스의 하위 집합에 있는 프로세스에서 장기간 실행 중인 프로세스가 있는 시스템에 이점을 제공합니다. 또한 여러 NUMA 노드의 리소스를 소비하는 애플리케이션에 유용하지만 시스템 리소스 소비율이 증가하면 numad에서 얻을 수 있는 효과는 감소합니다.

프로세스의 실행 시간이 몇 분이거나 많은 리소스를 소비하지 않는 경우 numad는 성능을 개선하지 않을 가능성이 높습니다. 대형 메모리 데이터베이스와 같은 지속적으로 예상치 못한 메모리 액세스 패턴을 갖는 시스템도 numad의 사용으로 혜택을 얻을 가능성이 낮습니다.

numad 사용에 대한 보다 자세한 내용은 3.3.5절. “numad를 사용하여 NUMA 친화도 자동 관리 ” 또는 A.14절. “numad” 또는 man 페이지에서 참조하십시오:

```
$ man numad
```

2.14. SystemTap

SystemTap은 운영 체제 활동 (특히 커널 활동)을 상세하게 모니터링하고 분석할 수 있는 추적 및 측정 도구입니다. 이는 top, ps, netstat, iostat와 같은 도구 출력과 유사한 정보를 제공하지만 수집된 정보에 대해 보다 상세히 필터링하고 분석할 수 있는 옵션이 포함되어 있습니다.

SystemTap을 통해 보다 깊이 있고 정확하게 시스템 활동 및 애플리케이션 동작을 분석하여 시스템 및 애플리케이션 병목 현상을 정확하게 지적할 수 있습니다.

SystemTap에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에 있는 Red Hat Enterprise Linux 7 *SystemTap Beginner's Guide* 및 Red Hat Enterprise Linux 7 *SystemTap TapSet Reference*에서 참조하십시오.

2.15. OProfile

OProfile은 시스템 전역 성능 모니터링 도구입니다. 이는 프로세서의 전용 성능 모니터링 하드웨어를 사용하여 메모리 참조 발생 시기, L2 캐시 요청 수, 전송된 하드웨어 요청 수와 같은 특정 이벤트 발생 빈도를 지정하기 위해 커널 및 시스템 실행 파일에 대한 정보를 검색합니다. 또한 OProfile은 프로세서 사용량 및 가장 많이 사용되는 애플리케이션 및 서비스를 지정하는데 사용될 수 있습니다.

하지만 OProfile에는 몇 가지 제한 사항이 있습니다:

- 성능 모니터링 샘플이 정확하지 않을 수 있습니다. 프로세서가 지시 사항을 순서대로 실행하지 않을 수 있기 때문에 인터럽트를 발생시킨 지시 사항 대신 인근 지시 사항에서 샘플을 기록할 수 있습니다.
- OProfile 프로세스가 여러번 시작 및 중지될 수 있으며 여러 실행을 통해 샘플이 누적 축적될 수 있습니다. 경우에 따라 이전 실행에서 얻은 데이터 샘플을 삭제해야 합니다.
- OProfile은 CPU 액세스에 의해 제한되는 프로세스로 문제를 식별하는데 중점을 두고 있습니다. 따라서 다른 이벤트에 대해 잠금 상태에서 대기하고 있는 동안 수면 상태에 있는 프로세스는 인식하지 않습니다.

OProfile에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에 있는 A.15절. “OProfile” 또는 *Red Hat Enterprise Linux 7 System Administrator's Guide*에서 참조하십시오. 또는 **/usr/share/doc/oprofile-*version***에 있는 시스템 문서에서 참조하실 수 있습니다.

2.16. Valgrind

Valgrind는 애플리케이션의 성능을 개선하기 위한 검색 및 프로파일링 도구를 제공합니다. 이러한 도구는 힙, 스택, 배열 오버런 이외에 메모리 및 스레드 관련 오류를 감지할 수 있으므로 애플리케이션 코드에서 오류를 확인하고 쉽게 수정할 수 있습니다. 캐시, 힙, 분기 예측을 프로파일링하여 애플리케이션 속도를 높이고 메모리 사용을 최소화할 수 있는 요소를 식별할 수 있습니다.

Valgrind는 애플리케이션을 통합 CPU에서 실행하고 기존 애플리케이션 코드를 실행하는 동안 계측하여 애플리케이션을 분석합니다. 다음에 애플리케이션 실행에 관련된 각각의 프로세스를 사용자 지정 파일, 파일 설명

자, 네트워크 소켓으로 명확하게 식별하는 설명을 출력합니다. 계측된 코드의 실행은 일반적 실행 보다 4-50배의 시간이 걸릴 수 있다는 것을 염두해 두십시오.

Valgrind는 다시 컴파일하지 않고 그대로 애플리케이션에서 사용할 수 있습니다. 하지만 Valgrind는 코드의 문제를 식별하기 위해 디버그 정보를 사용하므로 애플리케이션 및 지원 라이브러리가 유효한 디버깅 정보로 컴파일되지 않은 경우 Red Hat은 이러한 정보를 포함하도록 다시 컴파일할 것을 권장합니다.

Valgrind는 디버깅 효율을 높이기 위해 GNU Project Debugger (gdb)를 통합하고 있습니다.

Valgrind 및 Valgrind 하위 도구는 메모리 프로파일링에 유용합니다. 시스템 메모리를 프로파일링하기 위해 Valgrind를 사용하는 방법에 대한 보다 자세한 내용은 4.2.2절. “Valgrind로 애플리케이션 메모리 사용량 프로파일링 ”에서 참조하십시오.

Valgrind에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 Red Hat Enterprise Linux 7 Developer Guide에서 참조하십시오.

Valgrind 사용에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man valgrind
```

valgrind 패키지 설치 시 제공되는 문서는 **/usr/share/doc/valgrind-*version***에서 확인하실 수 있습니다.

3장. CPU

다음 부분에서는 Red Hat Enterprise Linux 7에서 애플리케이션 성능에 영향을 미치는 CPU 하드웨어 정보 및 설정 옵션에 대해 설명합니다. 3.1절. “고려 사항”에서는 성능에 영향을 미치는 CPU 관련 요인에 대해 설명합니다. 3.2절. “성능 관련 문제 모니터링 및 진단”에서는 CPU 하드웨어나 설정에 관련된 성능 문제를 분석하기 위한 Red Hat Enterprise Linux 7 도구 사용 방법에 대해 설명합니다. 3.3절. “권장 설정 ”에서는 Red Hat Enterprise Linux 7에서 CPU 관련 성능 문제를 해결하기 위해 사용할 수 있는 도구 및 전략에 대해 설명합니다.

3.1. 고려 사항

다음 부분에서는 시스템 및 애플리케이션 성능에 영향을 미치는 요인에 대해 설명합니다:

- 메모리와 같은 관련 리소스 및 프로세서 간 연결 형태
- 프로세서가 실행을 위해 스레드 스케줄링 계획 방법
- Red Hat Enterprise Linux 7에서 프로세서 인터럽트 처리 방법

3.1.1. 시스템 토폴로지

현대적인 컴퓨터 시스템에서 대부분의 시스템에는 여러 프로세서가 있기 때문에 *central* processing unit이라는 개념은 오해를 불러일으킬 수 있습니다. 이러한 프로세서가 서로 연결된 방법 및 다른 시스템 리소스와의 연결 방법 — 시스템 *토폴로지* —에 따라 시스템, 애플리케이션 성능 및 시스템 튜닝 시 고려 사항에 크게 영향을 미칠 수 있습니다.

최근 컴퓨팅에서 주로 사용되는 두 가지 유형의 토폴로지는 다음과 같습니다:

SMP (Symmetric Multi-Processor) 토폴로지

SMP 토폴로지에서는 모든 프로세서가 동일한 시간 동안 메모리에 액세스할 수 있습니다. 하지만 공유되는 동등한 메모리 액세스는 기본적으로 전체 CPU에서 직렬화된 메모리 액세스를 강제하기 때문에 SMP 시스템 확장 제약이 전반적으로 허용되지 않습니다. 이러한 이유로 실제로 최근 모든 서버 시스템은 NUMA 시스템을 사용하고 있습니다.

NUMA (Non-Uniform Memory Access) 토폴로지

NUMA 토폴로지는 SMP 토폴로지 보다 최근에 개발되었습니다. NUMA 시스템에서 여러 프로세서는 하나의 소켓에 물리적으로 그룹화되어 있습니다. 각 소켓에는 메모리 전용 영역과 이러한 메모리에 로컬 액세스하는 노드라고 부르는 프로세서가 있습니다.

동일한 노드에 있는 프로세서는 노드 메모리 뱅크에 고속으로 액세스할 수 있으며 다른 노드에 있는 메모리 뱅크로의 액세스는 느려집니다. 따라서 비-로컬 메모리에 액세스 하는 경우 성능이 저하됩니다.

이러한 성능 저하에 따라 NUMA 토폴로지를 사용하는 시스템에서 성능에 민감한 애플리케이션의 경우 애플리케이션을 실행하는 프로세서와 동일한 노드에 있는 메모리에 액세스하고 가능한 원격 메모리 액세스는 피하도록 합니다.

따라서 NUMA 토폴로지를 사용하는 시스템에서 애플리케이션 성능을 조정할 때 애플리케이션이 실행되는 위치와 실행되는 지점에서 가장 가까운 메모리 뱅크가 무엇인지를 고려해야 합니다.

NUMA 토폴로지를 사용하는 시스템에서 **/sys** 파일 시스템에는 프로세서, 메모리, 주변 장치가 연결된 방법에 관한 정보가 들어 있습니다. **/sys/devices/system/cpu** 디렉토리에는 시스템의 프로세서가 서로 연결된 방법에 대한 정보가 들어 있습니다. **/sys/devices/system/node** 디렉토리에는 시스템의 NUMA 노드와 노드 간의 상대 거리에 대한 정보가 들어 있습니다.

3.1.1.1. 시스템 토폴로지 지정

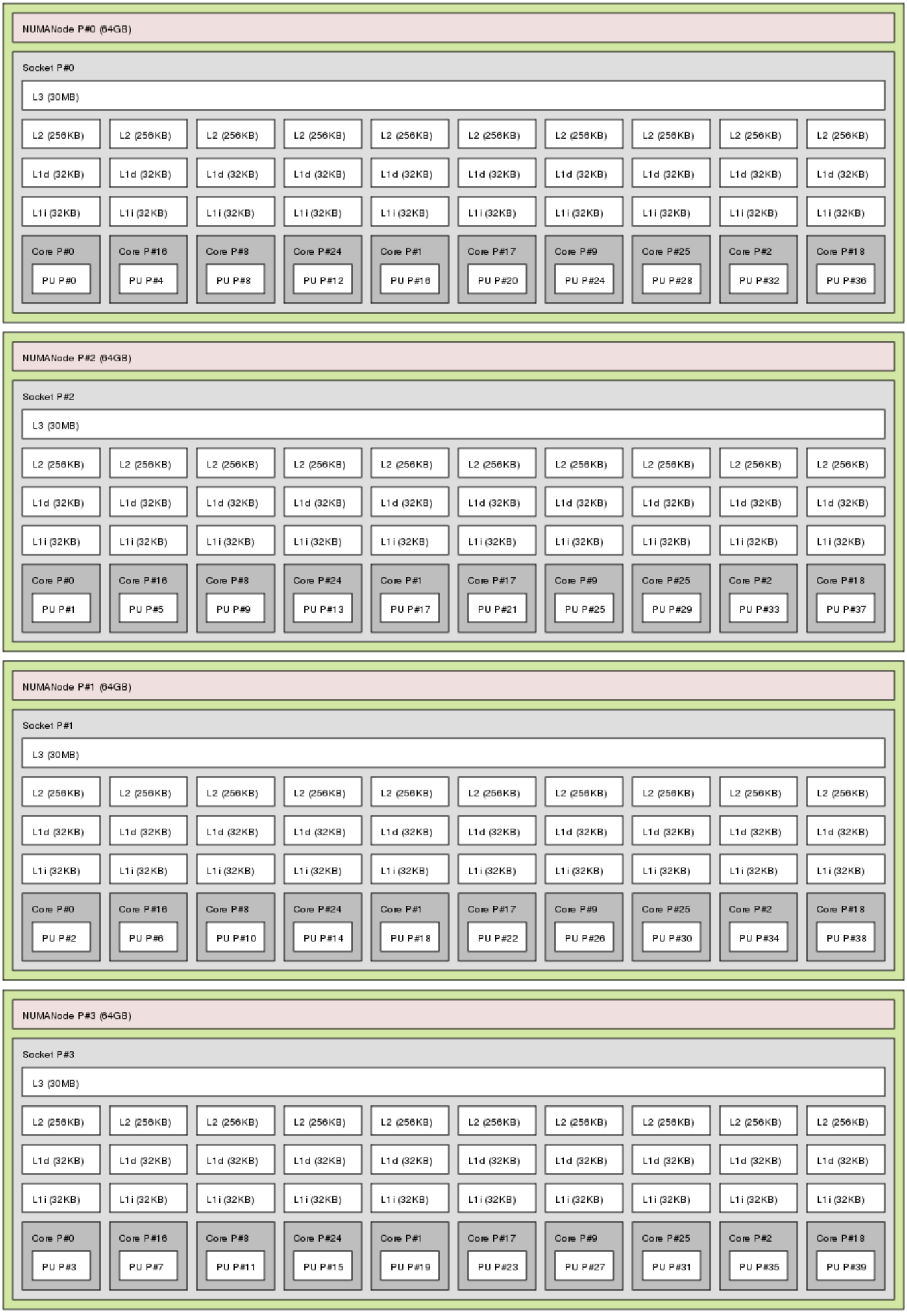
시스템 토폴로지를 이해하는데 도움이 될 수 있는 여러 명령이 있습니다. **numactl --hardware** 명령은 시스템 토폴로지 개요를 표시합니다.

```
$ 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
node 1 cpus: 2 6 10 14 18 22 26 30 34 38
node 1 size: 65536 MB
node 1 free: 44321 MB
node 2 cpus: 1 5 9 13 17 21 25 29 33 37
node 2 size: 65536 MB
node 2 free: 44304 MB
node 3 cpus: 3 7 11 15 19 23 27 31 35 39
node 3 size: 65536 MB
node 3 free: 44329 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

util-linux 패키지로 제공되는 **lscpu** 명령은 CPU, 스레드, 코어, 소켓, NUMA 노드 수와 같은 CPU 아키텍처에 대한 정보를 수집합니다.

```
$ 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
```

hwloc 패키지로 제공되는 **lstopo** 명령은 시스템의 그래픽 이미지를 표시합니다. **lstopo-no-graphics** 명령을 사용하면 텍스트 출력합니다.


NUMANode P#0 (64GB)
Socket P#0
L3 (30MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 PU P#0
Core P#16 PU P#4
Core P#8 PU P#8
Core P#24 PU P#12
Core P#1 PU P#16
Core P#17 PU P#20
Core P#9 PU P#24
Core P#25 PU P#28
Core P#2 PU P#32
Core P#18 PU P#36
NUMANode P#2 (64GB)
Socket P#2
L3 (30MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 PU P#1
Core P#16 PU P#5
Core P#8 PU P#9
Core P#24 PU P#13
Core P#1 PU P#17
Core P#17 PU P#21
Core P#9 PU P#25
Core P#25 PU P#29
Core P#2 PU P#33
Core P#18 PU P#37
NUMANode P#1 (64GB)
Socket P#1
L3 (30MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 PU P#2
Core P#16 PU P#6
Core P#8 PU P#10
Core P#24 PU P#14
Core P#1 PU P#18
Core P#17 PU P#22
Core P#9 PU P#26
Core P#25 PU P#30
Core P#2 PU P#34
Core P#18 PU P#38
NUMANode P#3 (64GB)
Socket P#3
L3 (30MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 PU P#3
Core P#16 PU P#7
Core P#8 PU P#11
Core P#24 PU P#15
Core P#1 PU P#19
Core P#17 PU P#23
Core P#9 PU P#27
Core P#25 PU P#31
Core P#2 PU P#35
Core P#18 PU P#39

lstopo 명령 출력

3.1.2. 스케줄링

Red Hat Enterprise Linux에서는 프로세스 실행의 최소 단위를 *스레드 (thread)*라고 합니다. 시스템 스케줄러는 스레드 실행 프로세서와 스레드 실행 기간을 지정합니다. 하지만 스케줄러의 주요 목적은 시스템을 계속 가동하는데 있으므로 애플리케이션 성능을 최적화하도록 스레드를 스케줄링하지 않을 수 있습니다.

예를 들어 NUMA 시스템에서 노드 B의 프로세서를 사용할 수 있을 때 애플리케이션이 노드 A에서 실행 중이라고 합시다. 노드 B 프로세서를 계속 가동시키려면 스케줄러는 애플리케이션 스레드 중 하나를 노드 B로 옮겨야 합니다. 스레드는 노드 B에서 실행되고 있고 노드 A 메모리는 더이상 스레드의 로컬 메모리가 아니기 때문에 액세스하는데 시간이 걸리게 됩니다. 스레드가 노드 B에서 실행 완료하는데 걸리는 시간이 노드 A에서 프로세서를 사용 가능하게 될 때 까지의 대기 시간과 로컬 메모리 액세스가 있는 원래의 노드에 있는 스레드를 실행하기 위한 시간 보다 더 걸릴 수 있습니다.

성능에 민감한 애플리케이션의 경우 설계자 또는 관리자가 스레드를 실행할 위치를 지정하는 것이 좋을 수 있습니다. 성능에 민감한 애플리케이션에 필요한 스레드를 적절하게 스케줄링하는 방법에 대한 자세한 내용은 3.3.6절. “스케줄링 정책 조정 ”에서 확인하십시오.

3.1.2.1. 커널 틱

Red Hat Enterprise Linux 이전 버전에서 Linux 커널은 완료해야할 작업을 확인하기 위해 정기적으로 각각의 CPU를 중단하고 이러한 결과를 사용하여 프로세스 스케줄링 및 부하 분산에 대한 결정을 내렸습니다. 이러한 정기적인 인터럽트는 커널 *틱 (tick)*이라고 했습니다.

코어에서 실행해야 할 작업 유무에 관계 없이 틱이 발생했습니다. 즉 유휴 상태인 코어도 인터럽트에 응답하기 위해 정기적으로 (초 당 최대 1000회) 높은 전력 상태로 강제되었습니다. 이로 인해 시스템에 최신 x86 프로세서가 포함된 경우 절전 상태를 효율적으로 사용할 수 없습니다.

Red Hat Enterprise Linux 6 및 7에서 기본적으로 커널은 전력 소비가 낮은 유휴 상태의 CPU를 인터럽트하지 않습니다. 이러한 동작은 틱리스 커널이라고 합니다. 실행 중인 작업이 적은 경우 주기적 인터럽트는 요청 시 인터럽트를 대신하고 있기 때문에 CPU는 유휴 상태 혹은 저전력 상태로 오래남아 있어 전력 사용을 줄일 수 있습니다.

Red Hat Enterprise Linux 7에서는 동적 틱리스 옵션 (**nohz_full**)을 제공하여 사용자 영역 작업에 의한 커널 간섭을 감소시킴으로써 기능이 개선되었습니다. 이러한 옵션은 ***nohz_full*** 커널 매개 변수를 사용하여 지정된 코어에서 활성화할 수 있습니다. 이러한 옵션을 코어에서 사용할 때 모든 시간 관리 동작은 대기 시간 제약이 없는 코어로 이동합니다. 이는 사용자 영역 작업이 커널 타이머 틱과 관련된 마이크로초 단위의 대기 시간 제약이 있는 고성능 계산 및 실시간 계산을 필요로하는 작업에 유용합니다.

Red Hat Enterprise Linux 7에서 동적 틱리스 동작을 활성화하는 방법에 대한 자세한 내용은 3.3.1절. “커널 틱 시간 설정 ”에서 참조하십시오.

3.1.3. IRQ (Interrupt Request) 처리

인터럽트 요청 또는 IRQ는 일부 하드웨어에서 프로세서에 신속한 대응을 요청하는 신호입니다. 시스템의 각 장치에는 하나 이상의 IRQ 번호가 지정되어 있어 고유한 인터럽트를 전송할 수 있습니다. 인터럽트를 활성화하면 인터럽트 요청을 수신하는 프로세서는 이러한 요청에 응답하기 위해 현재 애플리케이션 스레드 실행을 바로 중단합니다.

이는 정상 작동을 중지시키기 때문에 인터럽트 비율이 높을 경우 시스템 성능이 현저히 저하될 수 있습니다. 인터럽트 선호도를 설정하거나 여러 낮은 우선 순위의 인터럽트를 배치(여러 인터럽트를 *통합*)에 전송하여 인터럽트 시간을 감소시킬 수 있습니다.

인터럽트 요청 조정에 대한 보다 자세한 내용은 3.3.7절. “인터럽트 친화도 설정 ” or 3.3.8절. “Tuna를 사용하여 CPU, 스레드, 인터럽트 친화도 설정 ”에서 참조하십시오. 네트워크 인터럽트에 대한 자세한 내용은 6장. *네트워킹* 에서 확인하십시오.

3.2. 성능 관련 문제 모니터링 및 진단

Red Hat Enterprise Linux 7에는 시스템 성능을 모니터링하고 시스템 프로세스 및 설정과 관련된 성능 문제를 진단하는데 유용한 여러 도구가 있습니다. 다음 부분에서는 프로세서 관련 성능 문제를 모니터링하고 진단하기 위해 사용 가능한 도구 및 사용 방법에 대해 설명합니다.

3.2.1. turbostat

Turbostat는 관리자가 과도한 전력 소비, 절전 상태가 될 수 없는 문제, 불필요하게 SMI (system management interrupts)가 발생하는 것과 같은 서버에서의 예기치 않은 동작을 구별하는데 도움이 되는 카운터 결과를 지정한 간격으로 출력합니다.

turbostat 도구는 *kernel-tools* 패키지의 일부입니다. 이는 AMD64 및 Intel® 64 프로세서 시스템에서의 사용을 지원합니다. 실행, 고정 타임 스탬프 카운터에 대한 프로세서 지원, APERF 및 MPERF 모델 별 레지스터를 위해 root 권한이 필요합니다.

사용 예는 man 페이지에서 참조하십시오:

```
$ man turbostat
```

3.2.2. numastat

중요

이 도구는 Red Hat Enterprise Linux 6 라이프 사이클에서 상당한 업데이트를 받습니다. 기본 출력은 Andi Kleen에 의해 작성된 기존 도구와 호환성을 유지하지만 numastat에 다른 옵션이나 매개변수를 지정은 출력 형식에서 크게 변경되어 있습니다.

numastat 도구는 프로세스와 운영 체제의 메모리 통계를 NUMA 노드 단위로 표시하여 관리자는 프로세스 메모리가 시스템 전체에 확산되었는지 또는 특정 노드에 집중되어 있는지를 확인할 수 있습니다.

프로세스 메모리가 할당된 노드와 동일한 노드에서 프로세스 스레드가 실행되고 있는지 확인하려면 프로세서당 **top** 출력을 사용하여 **numastat** 출력을 상호 참조합니다.

Numastat는 *numactl* 패키지로 제공됩니다. **numastat** 출력에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man numastat
```

3.2.3. /proc/interrupts

`/proc/interrupts` 파일에는 특정한 I/O 장치에서 각 프로세서로 전송된 인터럽트 수가 나열되어 있습니다. 이는 인터럽트 요청 (IRQ) 수, 시스템의 각 프로세서의해 처리된 유형별 인터럽트 요청 수, 나열된 인터럽트 요청에 응답하는 쉼표로 구분된 장치 목록 등을 표시합니다.

특정 애플리케이션이나 장치가 원격 프로세서에 의해 처리되도록 대량의 인터럽트 요청을 생성할 경우 성능에 영향을 미칠 수 있습니다. 이러한 경우 동일한 노드에 있는 프로세서가 애플리케이션이나 장치가 인터럽트 요청을 처리하도록 하면 성능 저하를 완화시킬 수 있습니다. 특정 프로세서에 인터럽트 처리를 할당하는 방법에 대한 자세한 내용은 3.3.7절. “인터럽트 친화도 설정 ”에서 참조하십시오.

3.3. 권장 설정

Red Hat Enterprise Linux에는 시스템을 설정할 때 관리자에게 유용한 여러 도구가 있습니다. 다음 부분에서는 이러한 도구에 대해 설명하고 Red Hat Enterprise Linux 7에서 프로세서 관련 성능 문제를 해결하는 방법에 대해 설명합니다.

3.3.1. 커널 틱 시간 설정

기본적으로 Red Hat Enterprise Linux 7에서는 틱리스 커널을 사용합니다. 이 커널은 전력 소비를 감소시키기 위해 유휴 상태의 CPU를 인터럽트하지 않도록 하여 새로운 프로세서가 완전 절전 상태가 될 수 있게 합니다.

Red Hat Enterprise Linux 7에서는 동적 틱리스 옵션 (기본적으로 비활성화)을 제공하여 고성능 컴퓨팅 또는 실시간 컴퓨팅과 같은 대기 시간 제약이 있는 작업에 유용합니다.

특정 코어에서 동적 틱리스 동작을 활성화하려면 커널 명령행에 ***nohz_full*** 매개 변수를 사용하여 이러한 코어를 지정합니다. 16 코어 시스템에서 **nohz_full=1-15**를 지정하면 동적 틱리스 동작이 코어 1에서 15 까지 활성화되고 모든 시간 기록은 지정되지 않은 코어 (코어 0)로 이동됩니다. 이러한 동작은 부팅 시 임시적으로 활성화하거나 **/etc/default/grub** 파일에서 영구적으로 활성화할 수 있습니다. 영구적으로 활성화하려면 **grub2-mkconfig -o /boot/grub2/grub.cfg** 명령을 실행하여 설정을 저장합니다.

동적 틱리스 동작을 활성화에는 일부 수동 관리가 필요합니다.

- 시스템 시작 시 rcu 스레드를 대기 시간 제약이 없는 코어로 이동시켜야 합니다. 이 경우 코어 0가 됩니다.

```
# for i in `pgrep rcu[^c]` ; do taskset -pc 0 $i ; done
```

- 커널 명령행에서 ***isolcpus*** 매개 변수를 사용하여 특정 코어를 사용자 공간 작업에서 분리합니다.
- 옵션으로 커널 write-back bdi-flush 스레드의 CPU 선호도를 housekeeping 코어로 설정합니다:

```
echo 1 > /sys/bus/workqueue/devices/writeback/cpumask
```

다음 명령을 실행하여 동적 틱리스 설정이 올바르게 작동하고 있는지 확인합니다. 여기서 *stress*는 1 초 동안 CPU 에서 실행되는 프로그램입니다.

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 stress -t 1 -
c 1
```

*stress*를 대신하여 **while :; do d=1; done**과 같은 스크립트를 실행할 수 있습니다. 또한 다음의 링크에 있는 사용 가능한 프로그램도 이를 대신하여 사용할 수 있습니다: https://dl.fedoraproject.org/pub/epel/6/x86_64/repoview/stress.html.

기본 커널 타이머 설정은 사용 중인 CPU에서 1000 틱으로 되어 있습니다:

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 stress -t 1 -
c 1
1000 irq_vectors:local_timer_entry
```

동적 틱리스 커널을 설정하면 1틱이 됩니다:

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 stress -t 1 -
c 1
1 irq_vectors:local_timer_entry
```

3.3.2. 하드웨어 성능 정책 설정 (x86_energy_perf_policy)

x86_energy_perf_policy 도구를 사용하여 관리자는 성능 및 전력 효율성의 상대적 중요성을 지정할 수 있습니다. 이러한 정보를 사용하여 성능과 전력 효율성 사이에서 절충할 수 있는 옵션을 선택하면 기능을 지원하는 프로세서를 작동시킬 수 있습니다.

기본적으로 이는 **performance** 모드의 모든 프로세서에서 작동합니다. **CPUID.06H.ECX.bit3**의 표시가 있으면 프로세서의 지원이 필요한 것이므로 root 권한을 사용하여 실행해야 합니다.

x86_energy_perf_policy는 *kernel-tools* 패키지에서 제공됩니다. **x86_energy_perf_policy** 사용 방법에 대한 보다 자세한 내용은 A.10절. “x86_energy_perf_policy” 또는 man 페이지에서 참조하십시오:

```
$ man x86_energy_perf_policy
```

3.3.3. taskset으로 프로세스 친화도 설정

taskset 도구는 *util-linux* 패키지로 제공됩니다. **Taskset**을 사용하여 관리자는 실행 중인 프로세스의 친화도를 검색 및 설정하거나 지정된 프로세스 친화도로 프로세스를 시작할 수 있습니다.

중요

taskset은 로컬 메모리 할당을 보장하지 않습니다. 로컬 메모리 할당을 통해 추가적으로 성능을 향상시켜야 할 경우 Red Hat은 **taskset** 대신 **numactl**을 사용할 것을 권장합니다.

taskset에 대한 보다 자세한 내용은 A.16절. “taskset” 또는 man 페이지에서 참조하십시오:

```
$ man taskset
```

3.3.4. numactl로 NUMA 친화도 관리

관리자는 **numactl**을 사용하여 지정된 스케줄링 또는 메모리 배치 정책에 따라 프로세스를 실행할 수 있습니다. **Numactl**은 공유 메모리 세그먼트나 파일에 영구 정책을 설정하고 프로세서 친화도 및 메모리 친화도를 설정할 수 있습니다.

NUMA 토폴로지 시스템에서 프로세스의 메모리 액세스 속도는 프로세서와 메모리 뱅크 사이의 거리가 멀어질수록 느려집니다. 따라서 성능에 민감한 애플리케이션은 가장 가까운 메모리 뱅크에서 메모리를 할당하도록 설정해야 합니다. 동일한 NUMA 노드에 있는 메모리와 CPU를 사용하는 것이 가장 좋습니다.

성능에 민감한 멀티 스레드 애플리케이션은 특정 프로세서 대신 특정 NUMA 노드에서 실행하도록 설정하는 것이 좋습니다. 이러한 설정이 적합한 지에 대한 여부는 시스템 및 시스템 애플리케이션의 요구 사항에 따라 다릅니다. 여러 애플리케이션 스레드가 동일한 캐시 데이터에 액세스하는 경우 이러한 스레드를 동일한 프로세서에서 실행되도록 설정하는 것이 적합할 수 있습니다. 하지만 다른 데이터에 액세스하여 캐시되는 여러 스레드가 동일한 프로세스에서 실행되는 경우 각 스레드는 이전 스레드에 의해 액세스된 캐시된 데이터를 삭제할 수 있습니다. 즉 이로 인해 각각의 스레드가 캐시를 '누락'시키고 디스크에서 데이터를 가져와서 이를 캐시 교체하는데 실행 시간이 낭비될 수 있습니다. A.7절. “perf”에서 설명되어 있듯이 **perf** 도구를 사용하여 캐시 누락이 과도하게 발생하고 있는지 확인할 수 있습니다.

Numactl에서는 프로세스와 메모리 친화도를 관리하는데 유요한 여러 옵션을 제공합니다. 보다 자세한 내용은 A.12절. “numastat” 또는 man 페이지에서 참조하십시오:

```
$ man numactl
```

참고

numactl 패키지에는 **libnuma** 라이브러리가 포함되어 있습니다. 이러한 라이브러리는 커널에서 지원하는 NUMA 정책에 대한 간단한 프로그래밍 인터페이스를 제공하며 **numactl** 애플리케이션 보다 더 정교한 튜닝에 유용합니다. 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man numa
```

3.3.5. numad를 사용하여 NUMA 친화도 자동 관리

numad는 NUMA 친화도 자동 관리 데몬입니다. 이는NUMA 리소스 할당 및 관리 개선을 위해 NUMA 토폴로지 및 시스템 리소스 사용을 모니터링합니다.

numad는 다양한 작업 관리 시스템을 쿼리할 수 있는 사전 배포 컨설팅 서비스도 제공하고 있으며 프로세스의 CPU 및 메모리 리소스의 초기 바인딩을 제공합니다. 이러한 사전 배포 컨설팅 서비스는 시스템에서 numad가 실행 가능한 파일로 또는 서비스로 실행되고 있는지에 대한 여부와 상관 없이 사용 가능합니다.

numad 사용 방법에 대한 자세한 내용은 A.14절. “numad” 또는 man 페이지에서 참조하십시오:

```
$ man numad
```

3.3.6. 스케줄링 정책 조정

Linux 스케줄러는 여러 스케줄링 정책을 통해 스레드 실행 위치 및 실행 기간을 지정할 수 있습니다. 스케줄링 정책은 일반 정책과 실시간 정책이라는 두 가지 주요 범주로 나뉘어 집니다. 일반 정책은 일반적인 우선 순위 작업에 사용되며 실시간 정책은 인터럽트없이 완료해야 하는 시간적 제약이 있는 작업에 사용됩니다.

실시간 스레드는 지정된 타임 슬라이스에 영향을 받지 않습니다. 이 스레드는 차단, 종료, 자체 중단 또는 실행할 준비가 된 보다 높은 우선 순위를 갖는 스레드로 대체될 때 까지 실행됩니다. 우선 순위가 가장 낮은 실시간 스레드는 일반 정책 스레드 보다 먼저 스케줄링됩니다.

3.3.6.1. 스케줄링 정책

3.3.6.1.1. SCHED_FIFO를 사용한 정적 우선 순위 스케줄링

SCHED_FIFO (정적 우선 순위 스케줄링이라고도 함)는 실시간 정책으로 각 스레드에 고정된 우선 순위를 지정합니다. 이러한 정책을 통해 관리자는 이벤트 응답 시간을 개선하고 대기 시간을 감소시킬 수 있으므로 오랜 기간 동안 실행하지 않는 시간적 제약이 있는 작업의 경우 사용 권장됩니다.

SCHED_FIFO를 사용하고 있을 때 스케줄러는 우선 순위에서 모든 **SCHED_FIFO** 스레드 목록을 스캔하고 실행 준비된 가장 높은 우선 순위의 스레드를 스케줄링합니다. **SCHED_FIFO** 스레드의 우선 순위는 1에서 99로 지정할 수 있으며 99가 가장 높은 우선순위가 됩니다. Red Hat은 낮은 우선 순위 번호에서 시작하고 대시 시간 문제가 확인되는 경우에만 우선 순위를 증가시킬 것을 권장합니다.

주의

실시간 스레드는 타임 슬라이스에 영향을 받지 않기 때문에 Red Hat은 우선 순위를 99로 설정할 것을 권장하지 않습니다. 이러한 우선 순위로 설정하면 마이그레이션 및 워치독 스레드와 동일한 우선 순위로 프로세스를 배치하게 됩니다. 스레드가 연산 루프로 들어가 차단되면 스레드 실행을 할 수 없게 됩니다. 단일 프로세스 시스템은 이러한 경우 결국 중단됩니다.

관리자는 **SCHED_FIFO** 대역폭을 제한하여 실시간 애플리케이션 프로그래머가 프로세서를 독점하는 실시간 작업을 시작하지 않도록 할 수 있습니다.

/proc/sys/kernel/sched_rt_period_us

이 매개 변수는 프로세서 대역폭의 100 %로 간주되는 시간을 마이크로 초 단위로 지정합니다. 기본값은 **1000000** µs, 또는 1 초입니다.

/proc/sys/kernel/sched_rt_runtime_us

이 매개 변수는 실시간 스레드 실행에 할당된 시간을 마이크로 초 단위로 지정합니다. 기본값은 **950000** µs, 또는 0.95 초입니다.

3.3.6.1.2. SCHED_RR을 사용한 라운드 로빈 방식의 우선 순위 스케줄링

SCHED_RR은 **SCHED_FIFO**의 라운드 로빈 변형입니다. 이 정책은 여러 스레드를 동일한 우선 순위로 실행해야 할 경우 유용합니다.

SCHED_FIFO와 같이 **SCHED_RR**은 각 스레드에 고정된 우선 순위를 지정하는 실시간 정책입니다. 스케줄러는 우선 순위에 있는 모든 **SCHED_RR** 스레드 목록을 스캔하고 실행 준비된 가장 우선 순위가 높은 스레드를 스케줄링합니다. 하지만 **SCHED_FIFO**와는 다르게 동일한 우선 순위를 갖는 스레드는 특정 타임 슬라이스 내에서 라운드 로빈 방식으로 스케줄링됩니다.

이러한 타임 슬라이스 값을 ***sched_rr_timeslice_ms*** 커널 매개 변수 (**/proc/sys/kernel/sched_rr_timeslice_ms**)를 사용하여 밀리초로 설정할 수 있습니다. 가장 작은 값은 1 밀리 초입니다.

3.3.6.1.3. SCHED_OTHER를 사용한 일반적인 스케줄링

SCHED_OTHER는 Red Hat Enterprise Linux 7에서 기본 스케줄링 정책입니다. 이러한 정책은 CFS (Completely Fair Scheduler)를 사용하여 이러한 정책에 스케줄링된 모든 스레드에 공정한 프로세서 액세스를 제공합니다. 이 정책은 시간이 지남에 따라 보다 더 효율적인 스케줄링을 가능하게 하므로 대량의 스레드가 있거나 데이터 처리량이 중요한 경우 아주 유용합니다.

이 정책을 사용하면 스케줄러는 각 프로세스 스레드의 niceness 값에 따라 동적 우선 순위 목록을 생성합니다. 관리자는 프로세스의 niceness 값을 변경할 수 있지만 스케줄러의 동적 우선 순위 목록을 직접 변경할 수 없습니다.

niceness 프로세스를 변경하는 방법에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에 있는 *Red Hat Enterprise Linux 7 Deployment Guide*에서 참조하십시오.

3.3.6.2. CPU 분리

isolcpus 부트 매개 변수를 사용하여 스케줄러에서 여러 CPU를 분리할 수 있습니다. 이를 통해 스케줄러가 CPU에 있는 사용자 영역 스레드를 스케줄링하지 못하게 합니다.

CPU가 분리되면 CPU 친화도 시스템 호출 또는 numactl 명령을 사용하여 분리된 CPU에 수동으로 프로세스를 할당해야 합니다.

시스템에 3 번째 및 6 번째에서 8 번째 CPU로 분리시키려면 다음을 커널 명령행에 추가합니다:

```
isolcpus=2,5-7
```

Tuna 도구를 사용하여 CPU를 분리할 수 있습니다. **Tuna**를 사용하여 부팅 때에만 아니라 언제든지 CPU를 분리할 수 있습니다. 하지만 이러한 분리 방법은 ***isolcpus*** 매개 변수를 사용하는 경우와는 조금 다르기 때문에 ***isolcpus***를 통한 성능 향상은 기대하기 어렵습니다. 이 도구에 대한 보다 자세한 내용은 3.3.8절. “Tuna를 사용하여 CPU, 스레드, 인터럽트 친화도 설정 ”에서 참조하십시오.

3.3.7. 인터럽트 친화도 설정

인터럽트 요청에 있는 관련 친화도 속성 ***smp_affinity***는 인터럽트 요청을 처리하는 프로세서를 지정합니다. 애플리케이션 성능을 향상시키려면 인터럽트 친화도 및 프로세스 친화도를 동일한 프로세 또는 동일한 코어에 있는 프로세서에 지정합니다. 이를 통해 지정된 인터럽트 및 애플리케이션 스레드는 캐시 라인을 공유할 수 있습니다.

특정 인터럽트 요청의 인터럽트 친화도 값은 해당 **/proc/irq/*irq_number*/smp_affinity** 파일에 저장됩니다. ***smp_affinity***는 시스템에서 모든 프로세서를 표시하는 16 진수 비트 마스크로 저장됩니다. 기본값은 **f**이며 이는 인터럽트 요청이 시스템에 있는 모든 프로세서에서 처리될 수 있음을 의미합니다. 이 값을 **1**로 설정하면 프로세서 0만이 인터럽트를 처리할 수 있음을 의미합니다.

32 이상의 프로세서가 있는 시스템에서 32 비트 그룹에 ***smp_affinity*** 값을 구분하여 설정해야 합니다. 예를 들어 64 프로세서 시스템의 첫 번째 32 프로세서에만 인터럽트 요청을 처리하고자 할 경우 다음을 실행합니다:

```
# echo 0xffffffff,00000000 > /proc/irq/IRQ_NUMBER/smp_affinity
```

다른 방법으로 BIOS가 NUMA 토폴로지를 내보낼 경우 **irqbalance** 서비스는 하드웨어 요청 서비스에 로컬이 되는 노드에서 인터럽트 요청을 처리하는데 이러한 정보를 사용할 수 있습니다. **irqbalance**에 대한 보다 자세한 내용은 A.1절. “irqbalance”에서 참조하십시오.

참고

인터럽트 스티어링을 지원하는 시스템에서 인터럽트 요청의 ***smp_affinity***를 수정하여 하드웨어를 설정하고 인터럽트를 특정 프로세서에서 실행하는 결정을 커널 개입없이 하드웨어 수준에서 수행할 수 있습니다. 인터럽트 스티어링에 대한 보다 자세한 내용은 6장. *네트워킹*에서 참조하십시오.

3.3.8. Tuna를 사용하여 CPU, 스레드, 인터럽트 친화도 설정

Tuna는 CPU, 스레드, 인터럽트 친화도를 관리하고 엔티티 유형에 대한 다양한 동작을 제공합니다. **Tuna** 기능에 대한 전체 목록은 A.2절. “Tuna”에서 확인하십시오.

지정된 여러 CPU에서 모든 스레드를 분리하려면 다음 명령을 실행합니다. 여기서 *CPUs*는 분리하고자 하는 CPU 번호로 변경합니다.

```
# tuna --cpus CPUs --isolate
```

특정 스레드를 실행할 수 있는 CPU 목록에 특정 CPU를 포함시키려면 다음 명령을 실행합니다. 여기서 *CPUs*는 포함시키고자 하는 CPU 번호로 변경합니다.

```
# tuna --cpus CPUs --include
```

특정 CPU에 인터럽트 요청을 옮기려면 다음 명령을 실행합니다. 여기서 *CPU*를 CPU 번호로 변경하고 *IRQs*를 옮기고자 하는 인터럽트 요청을 쉼표로 구분하여 변경합니다.

```
# tuna --irqs IRQs --cpus CPU --move
```

다른 방법으로 다음 명령을 사용하여 **sfc1*** 패턴의 모든 인터럽트 요청을 검색할 수 있습니다.

```
# tuna -q sfc1* -c7 -m -x
```

스레드 정책과 우선 순위를 변경하려면 다음 명령을 실행합니다. 여기서 *thread*는 변경하고자 하는 스레드로 변경하고 *policy*는 스레드에 적용할 정책 이름으로 변경하며 *level*은 0 (가장 낮은 우선 순위)에서 99 (가장 높은 우선 순위) 까지의 정수로 변경합니다.

```
# tuna --threads thread --priority policy:level
```

4장. 메모리

다음 부분에서는 Red Hat Enterprise Linux 7의 메모리 관리 기능에 대해 설명합니다. 4.1절. “고려 사항”에서는 성능에 영향을 미치는 메모리 관련 요인에 대해 다룹니다. 4.2절. “성능 관련 문제 모니터링 및 진단”에서는 메모리 사용이나 설정에 관련된 성능 문제를 진단하기 위한 Red Hat Enterprise Linux 7 도구 사용 방법에 대해 설명합니다. 4.3절. “설정 도구 ”에서는 Red Hat Enterprise Linux 7에서 메모리 관련 성능 문제를 해결하는데 사용할 수 있는 도구 및 전략에 대해 설명합니다.

4.1. 고려 사항

기본적으로 Red Hat Enterprise Linux 7은 보통 수준의 작업 부하에 대해 최적화되어 있습니다. 애플리케이션 이난 사용 사례에 대량의 메모리가 필요할 경우 시스템이 가상 메모리를 관리하는 방법을 변경하면 애플리케이션의 성능이 향상될 수 있습니다.

4.1.1. 페이지 크기

물리적 메모리는 페이지라는 부분으로 관리됩니다.각 페이지의 물리적 위치가 가상 위치에 맵핑되어 프로세스는 메모리에 액세스할 수 있습니다. 이러한 맵핑은 페이지 테이블이라는 데이터 구조에 저장됩니다.

기본적으로 페이지 크기는 약 4 KB입니다. 기본 페이지 크기가 너무 작기 때문에 대량의 메모리를 관리하려면 다수의 페이지가 필요합니다. 하지만 페이지 테이블에 저장할 수 있는 주소 맵핑 수에는 제한이 있기 때문에 저장할 수 있는 주소 맵핑 수를 증가시키는 것은 비용이 많이 들고 메모리 요구 사항에 따라 성능 수준을 관리한다는 점에서 어렵습니다.

Red Hat Enterprise Linux에서는 정적 huge 페이지로 페이지 당 대용량 메모리를 관리할 수 있는 기능을 제공합니다. 정적 huge 페이지는 최대 1 GB 까지 설정할 수 있습니다. 하지만 이를 수동으로 관리하는데에는 어려움이 있기 때문에 부팅시에 지정해야 합니다. Red Hat Enterprise Linux 7.1에서는 노드 기반으로 이를 지정할 수 있습니다.

Transparent huge 페이지는 정적 huge 페이지 대신 사용할 수 있는 대규모 자동화 페이지입니다. Transparent huge 페이지 크기는 2 MB이며 기본적으로 활성화되어 있습니다. 때때로 대기 시간 제약이 있는 애플리케이션에 방해가 될 수 있기 때문에 이러한 애플리케이션에서는 주로 비활성화되어 있습니다.

애플리케이션 성능 개선을 위한 huge 페이지 설정에 대한 보다 자세한 내용은 4.3.1절. “Huge 페이지 설정 ”에서 참조하십시오.

4.1.2. TLB (Translation Lookaside Buffer) 크기

페이지 테이블에서 주소 맵핑을 불러오는데는 시간 및 자원이 필요하기 때문에 Linux 운영 체제에서는 최근 사용된 주소 캐시 TLB (Translation Lookaside Buffer)가 제공됩니다. 하지만 기본 TLB에서 캐시할 수 있는 주소 맵핑 수는 제한되어 있습니다. 요청된 주소 맵핑이 TLB에 있지 않을 경우 (즉, TLB *miss*되어 있을 경우) 시스템은 가상 주소 맵핑에 물리 주소를 지정하기 위해 페이지 테이블을 읽을 수 있어야 합니다.

애플리케이션 메모리 요구 사항 및 주소 팹핑 캐시에 사용되는 페이지 크기 사이의 관계로 인해 대용량 메모리 요구 사항이 있는 애플리케이션은 최소 메모리 요구 사항이 있는 애플리케이션 보다 TLB miss에 의해 성능 저하될 가능성이 있습니다. 따라서 가능한 이러한 TLB miss가 발생하지 않도록 하는 것이 중요합니다.

Red Hat Enterprise Linux는 HugeTLB (Huge Translation Lookaside Buffer)를 제공하여 매우 큰 세그먼트에서 메모리 관리를 가능하게 합니다. 한 번에 대량의 주소 맵핑을 캐시할 수 있으므로 TLB 미스 가능성을 감소시키고 그 결과 대용량 메모리 요구 사항이 있는 애플리케이션의 성능을 향상시킬 수 있습니다.

HugeTLB 설정에 대한 보다 자세한 내용은 4.3.1절. “Huge 페이지 설정 ”에서 참조하십시오.

4.2. 성능 관련 문제 모니터링 및 진단

Red Hat Enterprise Linux 7에는 시스템 성능을 모니터링하고 시스템 메모리 관련 성능 문제를 진단하는데 유용한 여러 도구가 있습니다. 다음 부분에서는 메모리 관련 성능 문제를 모니터링하고 진단하기 위해 사용 가능한 도구 및 사용 방법에 대해 설명합니다.

4.2.1. vmstat로 메모리 사용량 모니터링

Vmstat는 *procps-ng* 패키지로 제공되며 시스템 프로세스, 메모리, 페이징, 입/출력 차단, 인터럽트, CPU 동작에 대한 보고를 출력합니다. 마지막으로 컴퓨터를 부팅한 시간 또는 마지막 보고 시간에서 이벤트 평균을 바로 보고합니다.

다음 명령은 다양한 이벤트 카운터 및 메모리 통계 테이블을 표시합니다.

```
$ vmstat -s
```

vmstat 사용 방법에 대한 보다 자세한 내용은 A.9절. “vmstat” 또는 man 페이지에서 참조하십시오:

```
$ man vmstat
```

4.2.2. Valgrind로 애플리케이션 메모리 사용량 프로파일링

Valgrind는 사용자 공간 바이너리에 대한 계측을 제공하는 프레임워크입니다. 이에는 프로그램 성능을 프로파일링하고 분석하는데 사용할 수 있는 여러 도구가 탑재되어 있습니다. 다음 부분에서 설명하고 있는 **valgrind** 도구는 초기화되지 않은 메모리 사용 및 부적절한 메모리 할장 및 할당 해제와 같은 메모리 관련 오류를 감지하는데 유용합니다.

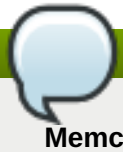
valgrind 또는 기타 다른 도구를 사용하려면 *valgrind* 패키지를 설치합니다:

```
# yum install valgrind
```

4.2.2.1. Memcheck로 메모리 사용량 프로파일링

Memcheck는 기본 **valgrind** 도구입니다. 다음과 같이 감지 및 진단하기 어려운 여러 메모리 오류를 감지 및 보고합니다:

- 실행하지 않아야 하는 메모리 액세스
- 지정되지 않았거나 초기화되지 않은 값 사용
- 잘못된 힙 메모리 해제
- 포인터 중복
- 메모리 누수



참고

Memcheck은 이러한 오류를 보고만 할 수 있지 발생하지 않도록 예방할 수 없습니다. 일반적으로 세그먼트 오류를 발생시킬 수 있는 방식으로 프로그램이 메모리에 액세스하는 경우 세그먼트 오류는 여전히 발생합니다. 하지만 **memcheck**는 오류 직전에 오류 메세지를 기록합니다.

memcheck는 계측을 사용하므로 **memcheck**로 실행되는 애플리케이션은 일반적인 실행 속도보다 10 배에서 30 배 정도 느리게 실행됩니다.

애플리케이션에서 **memcheck**를 실행하려면 다음 명령을 수행합니다:

```
# valgrind --tool=memcheck application
```

다음 옵션을 사용하여 특정 문제 유형에 **memcheck** 출력에 중점을 두도록 할 수 있습니다.

--leak-check

애플리케이션 실행 완료 후 **memcheck**는 메모리 누수를 검색합니다. 기본값은 **--leak-check=summary**로 검색된 메모리 누수 수를 출력합니다. **--leak-check=yes** 또는 **--leak-check=full**을 지정하여 각각의 누수 상세 정보를 출력할 수 있습니다. 비활성화하려면 **--leak-check=no**를 지정합니다.

--undef-value-errors

기본값은 **--undef-value-errors=yes**이며 이는 지정되지 않은 값이 사용될 경우 오류를 보고합니다. **--undef-value-errors=no**를 지정하여 이러한 보고를 비활성화하면 Memcheck 속도가 약간 빨라집니다.

--ignore-ranges

예를 들어 **--ignore-ranges=0xPP-0xQQ,0xRR-0xSS**와 같이 메모리 적용 가능성을 검사할 때 **memcheck**이 무시해야 하는 여러 범위를 지정합니다.

memcheck 옵션의 전체 목록은 **/usr/share/doc/valgrind-version/valgrind_manual.pdf**에 포함된 문서에서 참조하십시오.

4.2.2.2. Cachegrind로 캐시 사용량 프로파일링

Cachegrind는 시스템의 캐시 계층 및 분기 예측을 사용하여 프로그램의 상호 작용을 시뮬레이션합니다. 시뮬레이션된 첫 번째 레벨의 명령 및 데이터 캐시 사용량을 추적하여 이러한 이러한 레벨 캐시와의 잘못된 코드 상호 작용을 감지합니다. 또한 메모리 액세스를 추적하기 위해 마지막 레벨 캐시 (두 번째 또는 세 번째 레벨)도 추적합니다. 결과적으로 **cachegrind**와 함께 실행되는 애플리케이션은 일반적인 실행에 비해 20-100배 더 느리게 실행됩니다.

Cachegrind는 애플리케이션 실행 시간에 대한 통계를 수집하고 콘솔에 요약 내용을 출력합니다. 애플리케이션에서 **cachegrind**를 실행하려면 다음 명령을 실행합니다:

```
# valgrind --tool=cachegrind application
```

다음 옵션을 사용하여 특정 문제에서 **cachegrind** 출력에 중점을 둘 수 있습니다.

--I1

다음과 같이 첫 번째 레벨 명령 캐시의 크기, 연결성, 행 크기를 지정합니다: **--I1=*size*,*associativity*,*line_size***.

--D1

다음과 같이 첫 번째 레벨 데이터 캐시의 크기, 연결성, 행 크기를 지정합니다: **--D1=*size*,*associativity*,*line_size***.

--LL

다음과 같이 마지막 레벨 명령 캐시의 크기, 연결성, 행 크기를 지정합니다: **--LL=*size*,*associativity*,*line_size***

--cache-sim

캐시 액세스 및 미스 카운트 모음을 활성화 또는 비활성화합니다. 이는 기본적으로 활성화(**--cache-sim=yes**)되어 있습니다. --cache-sim 및 **--branch-sim** 모두를 비활성화하면 **cachegrind**에는 수집할 정보가 없게 됩니다.

--branch-sim

분기 명령 및 잘못된 예측 수 수집을 활성화 또는 비활성화합니다. 이는 기본적으로 활성화(**--branch-sim=yes**) 되어 있습니다. 이 옵션과 **--cache-sim**을 비활성화하면 **cachegrind**에는 수집할 정보가 없게 됩니다.

Cachegrind는 프로세스 당 자세한 프로파일링 정보를 **cachegrind.out.*pid*** 파일에 작성합니다. 여기서 *pid*는 프로세스 식별자입니다. 이 정보는 다음과 같이 **cg_annotate** 도구로 처리될 수 있습니다:

```
# cg_annotate cachegrind.out.pid
```

Cachegrind는 cg_diff 도구를 제공하여 코드 변경 전 후의 프로그램 성능을 더 쉽게 도표화할 수 있습니다. 출력 파일을 비교하려면 다음 명령을 실행합니다. 여기서 first는 초기 프로 파일 출력 파일로 second는 후속 프로 파일 출력 파일로 변경합니다.

```
# cg_diff first second
```

결과 출력 파일의 상세 정보는 **cg_annotate** 도구로 표시할 수 있습니다.

cachegrind 옵션의 전체 목록은 **/usr/share/doc/valgrind-version/valgrind_manual.pdf**에 있는 문서에서 참조하십시오.

4.2.2.3. Massif를 사용하여 힙 및 스택 영역 프로파일링

Massif는 지정된 애플리케이션이 사용하는 힙 영역을 측정합니다. 이는 유용한 공간 및 회계 관리 및 조정을 위해 할당된 추가 공간 모두를 측정합니다. **massif**는 실행 속도를 향상시키기 위해 애플리케이션의 메모리 사용을 감소시킬 수 있는 방법 및 애플리케이션이 시스템 스왑 공간을 고갈할 가능성을 감소시킬 수 있는 방법을 이해하는데 도움이 됩니다. **massif**로 애플리케이션을 실행하면 정상 적인 실행 속도 보다 20 배 정도 느려집니다.

애플리케이션에서 **massif**를 실행하려면 다음 명령을 수행합니다:

```
# valgrind --tool=massif application
```

다음 옵션을 사용하여 특정 문제에서 **massif** 출력에 중점을 두도록 할 수 있습니다.

--heap

massif가 힙 프로파일링을 수행할 지에 대한 여부를 지정합니다. 기본값은 **--heap=yes**입니다. 힙 프로파일링은 **--heap=no**로 설정하여 비활성화할 수 있습니다.

--heap-admin

힙 프로파일링을 활성화할 때 관리에 사용할 블록 당 바이트 수를 지정합니다. 기본값은 **8** 바이트입니다.

--stacks

massif가 스택 프로파일링을 수행할 지에 대한 여부를 지정합니다. 스택 프로파일링을 통해 **massif** 동작이 현저하게 느려질 수 있으므로 기본값은 **--stack=no**입니다. 스택 프로파일링을 활성화하

려면 이 옵션을 **--stack=yes**로 설정합니다. 프로파일링할 애플리케이션 관련 스택 크기 변경을 보다 쉽게 표시하기 위해 **massif**는 메인 스택 시작 크기가 0이라고 가정하고 있다는 점에 유의합니다.

--time-unit

massif가 프로파일링 데이터를 수집하는 간격을 지정합니다. 기본값은 **i** (명령 실행)입니다. **ms** (밀리초 단위 또는 실시간) 및 **B** (힙 및 스택에서 할당 또는 할당 해제된 바이트 수)를 지정할 수 있습니다. 다른 하드웨어에서 재현 가능하므로 단시간 실행 애플리케이션 및 테스트 목적의 경우 할당된 바이트 수를 확인하는 것이 유용합니다.

Massif는 프로파일링 데이터를 **massif.out.*pid*** 파일에 출력합니다. 여기서 *pid*는 지정된 애플리케이션의 프로세스 식별자입니다. **ms_print** 도구는 이러한 프로파일링 데이터를 그래프화하여 애플리케이션 실행에서 메모리 사용량을 표시하고 메모리 할당 절정에 할당할 사이트에 대한 상세 정보를 표시합니다. **massif.out.*pid*** 파일에서 데이터를 그래프화하려면 다음 명령을 실행합니다:

```
# ms_print massif.out.pid
```

Massif 옵션의 전체 목록은 **/usr/share/doc/valgrind-version/valgrind_manual.pdf**에 있는 문서에서 참조하십시오.

4.3. 설정 도구

일반적으로 메모리 사용량은 여러 커널 매개 변수 값으로 설정됩니다. 이러한 매개 변수는 /proc 파일 시스템의 파일 내용을 변경하여 임시적으로 설정하거나 *procps-ng* 패키지를 통해 제공되는 sysctl 도구를 사용하여 영구적으로 변경할 수 있습니다.

예를 들어 overcommit_memory 매개 변수를 1로 임시 설정하려면 다음 명령을 실행합니다:

```
# echo 1 > /proc/sys/vm/overcommit_memory
```

이 값을 영구적으로 설정하려면 **/etc/sysctl.conf**에 sysctl vm.overcommit_memory=1을 추가하고 다음 명령을 실행합니다:

```
# sysctl -p
```

매개 변수를 임시적으로 설정하는 것이 시스템에 미치는 매개 변수 영향을 확인하는데 유용합니다. 그 후 매개 변수 값이 원하는 결과를 가져올 경우 이를 영구 설정할 수 있습니다.

4.3.1. Huge 페이지 설정

Huge 페이지는 인접한 메모리 영역에 의존하기 때문에 메모리가 조각화되기 전 부팅 시 huge 페이지를 지정하는 것이 좋습니다. 이를 위해 커널 부트 명령행에서 다음의 매개 변수를 추가합니다:

hugepages

부팅 시 커널에서 설정되는 영구적 huge 페이지 수를 지정합니다. 기본값은 0입니다. 시스템에 물리적으로 인접한 여유 페이지가 충분할 경우에만 huge 페이지를 할당 (또는 할당 해제)할 수 있습니다. 이러한 매개 변수에 의해 예약된 페이지를 다른 용도로 사용할 수 없습니다.

이 값은 부팅 후 **/proc/sys/vm/nr_hugepages** 파일 값을 변경하여 조정할 수 있습니다.

NUMA 시스템에서 이러한 매개 변수로 지정한 huge 페이지는 노드 간에 균등하게 분할됩니다. 노드의 **/sys/devices/system/node/*node_id*/hugepages/hugepages-1048576kB/nr_hugepages** 파일 값을 변경하여 런타임 시 특정 노드에 huge 페이지를 할당할 수

있습니다.

보다 자세한 정보는 기본적으로 **/usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt**에 설치된 커널 관련 문서에서 참조하십시오.

hugepagesz

부팅 시 커널에서 설정되는 영구적 huge 페이지 크기를 지정합니다. 사용 가능한 값은 2 MB와 1 GB입니다. 기본값은 2 MB입니다.

default_hugepagesz

부팅 시 커널에서 설정되는 영구적 huge 페이지의 기본 크기를 지정합니다. 사용 가능한 값은 2 MB 및 1 GB입니다. 기본값은 2 MB입니다.

또한 다음 매개 변수를 사용하여 런타임 시 huge 페이지 작동에 영향을 미칠 수 있습니다.

/sys/devices/system/node/*node_id*/hugepages/hugepages-*size*/nr_hugepages

지정된 NUMA 노드에 할당된 huge 페이지 수 지정 크기를 정의합니다. 이는 Red Hat Enterprise Linux 7.1에서 지원됩니다. 다음 예에서는 2048 kB huge 페이지 20 페이지를 **node2**에 추가하고 있습니다.

```
# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1 Node 2 Node 3  Total add

sysctl vm.overcommit_memory=1

in /etc/sysctl.conf

and execute
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
# echo 20 > /sys/devices/system/node/node2/hugepages/hugepages-
2048kB/nr_hugepages
# 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
```

/proc/sys/vm/nr_overcommit_hugepages

오버 커밋 중인 메모리를 통해 생성 및 사용 가능한 최대 추가 huge 페이지 수를 지정합니다. 이 파일에 0이 아닌 값을 작성하면 영구 huge 페이지 풀이 모두 소모된 경우 커널의 일반 페이지 풀에서 시스템에 지정된 huge 페이지 수가 표시됩니다. 이러한 나머지 huge 페이지가 사용되지 않으면 사용 해제된 커널의 일반 페이지 풀로 반환됩니다.

4.3.2. 시스템 메모리 용량 설정

다음 부분에서는 시스템에서 메모리 사용량 개선에 유용한 메모리 관련 커널 매개 변수에 대해 설명합니다. 이러한 매개 변수는 **/proc** 파일 시스템에서 해당 파일 값을 변경하여 테스토 용도로 임시 설정할 수 있습니다. 매개 변수가 사용 환경에 적합한 최상의 성능을 제공하는 것을 확인한 후 **sysctl** 명령을 사용하여 이러한 매개 변수를 영구 설정할 수 있습니다.

참고

보다 전문적인 지식은 Red Hat Enterprise Linux Performance Tuning (RH442) 교육 과정을 통해 습득하실 수 있습니다.

4.3.2.1. 가상 메모리 매개 변수

다음 부분에 나열된 매개 변수는 따로 지정하지 않는 한 **/proc/sys/vm**에 있습니다.

dirty_ratio

백분율입니다. 총 시스템 메모리의 백분율이 변경되면 시스템은 변경 내용을 **pdflush** 연산 디스크에 작성하기 시작합니다. 기본값은 **20** %입니다.

dirty_background_ratio

백분율입니다. 총 시스템 메모리의 백분율이 변경되면 시스템은 변경 내용을 백그라운드 디스크에 작성하기 시작합니다. 기본값은 **10** %입니다.

overcommit_memory

대량 메모리 요청을 수락 또는 거부할 지에 대해 결정하는 조건을 지정합니다.

기본값은 **0**입니다. 기본적으로 커널은 사용 가능한 메모리 양과 메모리 양이 너무 커서 요청 실패 수를 추정하여 추론적 메모리 오버커밋을 처리합니다. 하지만 정확한 알고리즘이 아닌 추론을 통해 메모리가 할당되기 때문에 이러한 설정에서 메모리 오버로드가 발생할 수 있습니다.

이 매개변수를 **1**로 설정하면 커널은 메모리 오버커밋 처리를 수행하지 않습니다. 이로 인해 메모리 오버로드 발생 가능성이 증가하지만 메모리 집약적 작업 성능은 향상됩니다.

매개 변수를 **2**로 설정하면 커널은 사용가능한 총 swap 공간과 ***overcommit_ratio***에 지정된 물리적 RAM 비율 합계와 같거나 또는 큰 메모리 요청을 거부합니다. 이는 메모리 오버커밋 발생 가능성을 감소시키지만 swap 영역이 물리적 메모리 보다 큰 시스템에만 권장됩니다.

overcommit_ratio

overcommit_memory가 **2**로 설정되어 있는 경우 고려해야 할 물리적 RAM의 백분율을 지정합니다. 기본값은 **50**입니다.

max_map_count

프로세스에서 사용할 수 있는 최대 메모리 맵 영역 수를 지정합니다. 대부분의 경우 기본값으로 (**65530**)이 적절합니다. 애플리케이션에 이 파일 보다 많은 수를 맵핑해야 할 경우 이 값을 늘립니다.

min_free_kbytes

시스템 전체에 걸쳐 빈 공간의 최소 크기를 KB 단위로 지정합니다. 이는 각각의 낮은 메모리 영역에 적절한 값을 지정하는데 사용되며 그 크기에 비례하여 예약된 여유 페이지 수를 할당합니다.

주의

설정 값이 너무 낮거나 높으면 시스템이 손상될 수 있습니다. ***min_free_kbytes***를 너무 낮은 값으로 설정하면 시스템이 메모리를 회수하지 못하게 합니다. 이로 인해 시스템이 중단되고 메모리 부족으로 인해 프로세스가 종료될 수 있습니다. 하지만 ***min_free_kbytes***를 너무 높은 값으로 설정하면 (예: 총 시스템 메모리의 5–10%) 시스템에 바로 메모리 부족 현상이 나타나기 때문에 메모리를 회수하는데 시간을 너무 오래 소요하게 됩니다.

oom_adj

시스템의 메모리가 부족하고 ***panic_on_oom*** 매개 변수가 **0**로 설정되어 있으면 **oom_killer** 함수는 시스템을 복원할 수 있을 때 까지 프로세스를 종료하고 가장 높은 **oom_score**로 프로세스를 시작합니다.

oom_adj 매개 변수는 프로세스의 ***oom_score***를 지정하는데 유용합니다. 이 매개 변수는 프로세스 식별자 마다 설정되어 있습니다. **-17** 값은 프로세스의 **oom_killer**를 비활성화합니다. 그 밖에도 **-16**에서 **15**까지의 값을 사용할 수 있습니다.

참고

프로세스의 **oom_score**를 상속하는 조정 과정을 통해 프로세스가 생성됩니다.

스왑 (swappiness)

0에서 **100** 까지의 값으로 시스템이 익명의 메모리 또는 페이지 캐시를 선호하는 정도를 제어합니다. 높은 값은 파일 시스템 성능이 향상되지만 RAM에서 덜 활동적인 프로세스를 적극적으로 스왑합니다. 낮은 값은 메모리 프로세스의 스왑을 피하기 때문에 대기 시간이 줄어들고 I/O 성능이 저하됩니다. 기본값은 **60**입니다.

주의

swappiness==0로 설정하면 매우 적극적으로 스왑을 피하기 위해 메모리 및 I/O 압력에서 메모리 부족으로 인한 프로세스 강제 종료의 위험이 높아집니다.

4.3.2.2. 파일 시스템 매개 변수

다음 부분에 나열된 매개 변수는 따로 지정하지 않는 한 **/proc/sys/fs**에 있습니다.

aio-max-nr

모든 활성 비동기 입/출력 컨텍스트에서 허용되는 최대 이벤트 수를 지정합니다. 기본값은 **65536**입니다. 이 값을 변경해도 커널 데이터 구조를 미리 할당하거나 크기가 변경되지 않습니다.

file-max

커널에서 할당되는 최대 파일 처리 수를 지정합니다. 기본값은 커널에 있는 **files_stat.max_files** 값과 일치하며 ***NR_FILE*** (Red Hat Enterprise Linux에서 8192) 또는 다음 결과 중 더 큰 값으로 설정됩니다.

```
(mempages * (PAGE_SIZE / 1024)) / 10
```

이 값을 증가시키면 사용가능한 파일 처리 수 부족으로 인한 오류를 해결할 수 있습니다.

4.3.2.3. 커널 매개 변수

다음 부분에 나열된 매개 변수는 따로 지정하지 않는 한 **/proc/sys/kernel**에 있습니다.

msgmax

메세지 큐에서 단일 메세지의 최대 크기를 바이트 단위로 지정합니다. 이 값은 큐의 크기 (***msgmnb***)를 초과해서는 안됩니다. 기본값은 **65536**입니다.

msgmnb

단일 메세지 큐의 최대 크기를 바이트 단위로 지정합니다. 기본값은 **65536**입니다.

msgmni

메세지 큐 식별자의 최대 수 (즉 큐의 최대 수)를 지정합니다. 64 비트 아키텍처 시스템에서 기본값은 **1985**입니다.

shmall

한 번에 시스템에서 사용할 수 있는 총 공유 메모리 양을 페이지 단위로 지정합니다. Red Hat Enterprise Linux에서 기본값은 **1<<24** 또는 33554432 페이지입니다.

shmmax

커널에서 허용되는 단일 공유 메모리 세그먼트의 최대 크기를 페이지 단위로 지정합니다. Red Hat Enterprise Linux에서 기본값은 **1<<24** 또는 33554432 바이트입니다.

shmmni

시스템 전체의 공유 메모리 세그먼트의 최대 수를 지정합니다. 모든 시스템에서 기본값은 **4096**입니다.

threads-max

시스템 전체에서 커널이 한번에 사용할 수 있는 최대 스레드 작업 수를 지정합니다. 기본값은 커널 매개 변수 ***max_threads*** 값과 같거나 다음과 같은 값이어야 합니다:

```
mempages / (8 * THREAD_SIZE / PAGE SIZE )
```

최저 값은 **20**입니다.

5장. 스토리지 및 파일 시스템

다음 부분에서는 Red Hat Enterprise Linux 7의 I/O 및 파일 시스템 모두에서 애플리케이션에 영향을 미치는 지원되는 파일 시스템 및 설정 옵션에 대해 설명합니다. 5.1절. “고려 사항”에서는 성능에 영향을 미치는 I/O 및 파일 시스템 관련 요인에 대해 다룹니다. 5.2절. “성능 관련 문제 모니터링 및 진단”에서는 I/O이나 파일 시스템 설정에 관련된 성능 문제를 진단하기 위한 Red Hat Enterprise Linux 7 도구 사용 방법에 대해 설명합니다. 5.3절. “설정 도구 ”에서는 Red Hat Enterprise Linux 7에서 I/O이나 파일 시스템 관련 성능 문제를 해결하는데 사용할 수 있는 도구 및 전략에 대해 설명합니다.

5.1. 고려 사항

스토리지 및 파일 시스템 성능에 적합한 설정은 스토리지 사용 용도에 따라 다릅니다. I/O 및 파일 시스템 성능에 영향을 미치는 요인은 다음과 같습니다:

- 데이터 쓰기 및 읽기 패턴
- 기본 지오메트리로 데이터 정렬
- 블록 크기
- 파일 시스템 크기
- 저널 크기 및 위치
- 액세스 시간 기록
- 데이터 안정성 확보
- 미리 읽기 데이터
- 사전 할당 디스크 공간
- 파일 조각화
- 리소스 충돌

다음 부분에서는 파일 시스템 처리량, 확장성, 응답성, 리소스 사용량, 가용성 등에 영향을 미치는 포맷 및 마운트 옵션에 대해 설명합니다.

5.1.1. SSD (Solid-State Disks)

SSD (Solid-State Disks)는 영구적 데이터를 저장하기 위해 자기 플래터를 회전시키는 것이 아니라 NAND 플래시 칩을 사용합니다. 이는 전체 논리 블록 주소 범위에 걸쳐 데이터 액세스 시간을 일정하게 제공하며 액세스 시 발생하는 탐색 시간이 없습니다. 스토리지 공간은 기가 바이트 단위로 보다 높은 비용에 비해 스토리지 용량이 작지만 HDD 보다 지연 시간이 짧아 처리량이 높습니다.

SSD 사용 블록이 디스크 용량에 가까워 질수록 성능이 저하됩니다. 성능 저하 정도는 벤더에 따라 다르지만 모든 장치는 이러한 상황에서 성능 저하가 발생합니다. 삭제 동작을 활성화하여 이러한 성능 저하를 완화시킬 수 있습니다. 보다 자세한 내용은 5.1.4.3절. “유지 관리”에서 참조하십시오.

기본 I/O 스케줄러 및 가상 메모리 옵션은 SSD 사용에 적합합니다.

5.1.2. I/O 스케줄러

I/O 스케줄러는 스토리지 장치에서 I/O 작업 실행 시간 (언제 실행할 지 및 얼마나 오래 실행할 지)을 결정합니다. 이는 또한 I/O 엘리베이터라고도 합니다.

Red Hat Enterprise Linux 7에서는 3 가지 종류의 I/O 스케줄러를 제공합니다.

deadline

SATA 디스크를 제외한 모든 블록 장치의 기본 I/O 스케줄러입니다. **Deadline**은 요청이 I/O 스케줄러에 도달한 시점에서 요청 대기 시간을 보장합니다. 이러한 스케줄러는 대부분의 사용자에게 적합하지만 특히 쓰기 작업 보다 읽기 작업이 보다 자주 발생하는 환경에 적합합니다.

대기 큐에 있는 I/O 요청은 읽기 또는 쓰기 배치로 나뉘어 LBA 오름 차순으로 실행이 예약됩니다. 애플리케이션은 읽기 I/O에서 블록될 가능성이 높기 때문에 기본적으로 읽기 배치가 쓰기 배치보다 우선합니다. 배치 처리 후 **deadline**은 쓰기 작업의 프로세서 대기 시간을 확인하고 다음의 읽기 또는 쓰기 배치를 적절히 예약합니다. 배치 당 처리할 요청 수, 쓰기 배치 당 실행할 읽기 배치 수 요청 만료 전 까지의 시간 등은 모두 설정 가능합니다. 보다 자세한 내용은 5.3.4절. “deadline 스케줄러 튜닝 ”에서 확인하십시오.

cfq

SATA 디스크로 식별되는 장치 전용 기본 스케줄러입니다. **cfq** (Completely Fair Queueing) 스케줄러는 실시간, 최선형, 유휴 상태의 3 가지 다른 클래스로 분리합니다. 실시간 클래스에 있는 프로세스는 항상 최선형 클래스에 있는 프로세스 보다 항상 먼저 처리되고 최선형 클래스에 있는 프로세스는 유휴 상태 클래스에 있는 프로세서 보다 먼저 처리됩니다. 즉 실시간 클래스에 있는 프로세스는 최선형 및 유휴 상태의 프로세서 시간을 제공하지 않을 수 있음을 의미합니다. 기본적으로 프로세스는 최선형 클래스로 지정됩니다.

Cfq는 과거 사용 기록 데이터를 사용하여 애플리케이션에서 발행되는 차후 I/O 요청 수를 예상합니다. 더 많은 I/O가 예상되는 경우 다른 프로세스에서 I/O가 실행을 기다리고 있는 상태에서도 **cfq**는 유휴 상태가 되어 새로운 I/O를 기다립니다.

cfq 스케줄러는 유휴 상태가 되기 쉽기 때문에 의도적으로 조정하지 않은 경우 대형 탐색 페널티에서 손상을 받지 않는 하드웨어와 함께 사용하지 않도록 합니다. 또한 호스트 기반 하드웨어 RAID 컨트롤러와 같은 기타 다른 비 작업 유지 스케줄러와 함께 사용하지 않습니다. 이러한 스케줄러를 사용하면 대기 시간이 길어질 수 있습니다.

Cfq 동작은 고급 설정 가능합니다. 보다 자세한 내용은 5.3.5절. “cfq 스케줄러 튜닝 ”에서 참조하십시오.

noop

noop I/O 스케줄러는 간단한 FIFO (first-in first-out) 스케줄링 알고리즘을 구현합니다. 요청은 일반 블록 층에서 병합되며 이는 마지막 적중 항목 캐시입니다. 시스템이 CPU에 바인딩되어 있고 스토리지가 빠른 경우 이는 최상의 스케줄러가 됩니다.

다른 기본 I/O 스케줄러 설정 방법 및 특정 장치에 대해 다른 스케줄러를 지정하는 방법은 5.3절. “설정 도구 ”에서 참조하십시오.

5.1.3. 파일 시스템

다음 부분에서는 Red Hat Enterprise Linux 7에서 지원되는 파일 시스템 및 이러한 시스템의 권장 사용 사례, 일반적으로 파일 시스템에서 사용할 수 있는 포맷 및 마운트 옵션에 대해 설명합니다. 파일 시스템 조정을 위한 자세한 권장 사항은 5.3.7절. “성능 개선을 위한 파일 시스템 설정 ”에서 참조하십시오.

5.1.3.1. XFS

XFS는 강력하고 확장성이 뛰어난 64 비트 파일 시스템입니다. 이는 Red Hat Enterprise Linux 7에서 기본 파일 시스템입니다. XFS는 익스텐트 기반 할당을 사용하며 조각화를 줄이고 성능을 지원하는 사전 할당 및 지연 할당 등과 같은 여러 할당 체계를 특징으로 합니다. 또한 캐시 복구를 가능하게 하는 메타데이터 저널링도 지원하고 있습니다. XFS는 마운트되어 활성화되어 있는 동안 조각화를 완료하고 확장할 수 있으며 Red Hat Enterprise Linux 7에서는 여러 XFS 백업 및 복구 유틸리티를 지원하고 있습니다.

Red Hat Enterprise Linux 7.0 GA에서 XFS는 500 TB의 최대 파일 시스템 크기 및 8 EB의 최대 파일 오프셋(sparse 파일)을 지원합니다. XFS 관리 방법에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 *Red Hat Enterprise Linux 7 Storage Administration Guide*에서 참조하십시오. 특정 용도로 XFS를 조정하는 방법에 대한 자세한 내용은 5.3.7.1절. “XFS 튜닝 ”에서 참조하십시오.

5.1.3.2. Ext4

Ext4는 ext3 파일 시스템의 확장성을 높인 파일 시스템입니다. 기본 동작은 대부분의 작업에 적합합니다. 하지만 50 TB의 최대 파일 시스템 크기와 16 TB의 최대 파일 크기만을 지원합니다. ext4 관리 방법에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 *Red Hat Enterprise Linux 7 Storage Administration Guide*에서 참조하십시오. 특정 용도로 ext4를 조정하는 방법은 5.3.7.2절. “ext4 튜닝 ”에서 참조하십시오.

5.1.3.3. Btrfs (기술 프리뷰)

현재 Red Hat Enterprise Linux 7의 기본 파일 시스템은 XFS이지만 BTRFS 버전도 기술 프리뷰로 제공되고 있습니다. BTRFS는 새로운 copy-on-write 파일 시스템입니다. BTRFS의 고유한 기능은 다음과 같습니다:

- 전체 파일 시스템이 아닌 특정 파일, 볼륨, 하위 볼륨의 스냅샷 찍기 기능이 있습니다.
- 저렴한 디스크의 여러 중복 배열 버전(RAID)을 지원합니다.
- 역참조는 파일 시스템 개체에 I/O 오류를 매핑합니다.
- 자동 압축 (파티션에 있는 모든 파일은 자동으로 압축됩니다).
- 데이터 및 메타데이터에서 체크섬.

BTRFS는 안정적인 파일 시스템이지만 새로운 기능이 지속적으로 개발되고 있으며 복구 도구와 같은 일부 기능은 다른 파일 시스템의 완성도에 비해 매우 기본적입니다.

현재 BTRFS는 고급 기능 (스냅샷, 압축 및 파일 데이터 체크섬 등)이 필요하지만 성능은 상대적으로 중요하지 않을 때 적절한 선택입니다. 이러한 기능이 필요하지 않은 경우 시간이 지남에 따라 오류가 발생하거나 성능이 저하될 수 있으므로 파일 시스템으로 적합하지 않습니다. 다른 파일 시스템과 비교할 때 또 다른 단점은 지원되는 최대 파일 시스템 크기가 50 TB라는 것입니다.

Red Hat Enterprise Linux 7에서 btrfs는 기술 프리뷰로 제공됩니다. 기술 프리뷰 기능에 대한 보다 자세한 내용은 https://access.redhat.com/site/support/offerings/techpreview/에서 참조하십시오.

BTRFs 관리에 대한 보다 자세한 내용은 https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Storage_Administration_Guide/ch-btrfs.html의 *Red Hat Enterprise Linux 7 Storage Administration Guide*에서 참조하십시오. 특정 용도로 BTRFS를 튜닝하는 방법은 http://documentation-devel.engineering.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Performance_Tuning_Guide/sect-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Storage_and_File_Systems-Configuration_tools.html#sect-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Configuration_tools-Configuring_file_systems_for_performance에 있는 5.3.7.3 BTRFS 튜닝 섹션에서 참조하십시오.

5.1.3.4. GFS2

GFS2는 고가용성 애드온의 일부로 Red Hat Enterprise Linux 7에서 클러스터 파일 시스템을 지원합니다. GFS2는 클러스터의 모든 서버에 걸쳐 일관된 파일 시스템 이미지를 제공하여 서버가 단일 공유 파일 시스템에서 읽기 및 쓰기할 수 있도록 합니다.

GFS2는 250 TB의 최대 파일 시스템 크기를 지원합니다.

GFS2 관리에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 *Red Hat Enterprise Linux 7 Storage Administration Guide*에서 참조하십시오. 특정 용도로 GFS2를 튜닝하는 방법은 5.3.7.4절. “GFS2 튜닝 ”에서 참조하십시오.

5.1.4. 파일 시스템 튜닝 시 일반적으로 고려할 사항

다음 부분에서는 모든 파일 시스템에서 튜닝 시 일반적으로 고려해야 할 사항에 대해 설명합니다. 특정 파일 시스템의 튜닝 관련 권장 사항은 5.3.7절. “성능 개선을 위한 파일 시스템 설정 ”에서 참조하십시오.

5.1.4.1. 시간 형식에서 고려할 사항

장치 포맷 후 일부 파일 시스템 설정 지정을 변경할 수 없습니다. 다음 부분에서는 스토리지 장치 포맷 전 지정해야 할 옵션에 대해 설명합니다.

크기

워크로드에 적합한 크기의 파일 시스템을 만듭니다. 크기가 작은 파일 시스템은 비례적으로 백업 시간이 단축되며 파일 시스템 검사에 걸리는 시간 및 메모리도 작아 집니다. 하지만 파일 시스템이 너무 작으면 조각화로 인해 성능이 저하됩니다.

블록 크기

블록은 파일 시스템의 작업 단위입니다. 블록 크기는 단일 블록에 저장할 수 있는 데이터 양이므로 한 번에 쓰기 또는 읽기할 최소 데이터 양입니다.

기본 블록 크기는 대부분의 사용 경우에 적합합니다. 하지만 블록 크기 (또는 여러 블록 크기)가 한 번에 읽기 또는 쓰기 가능한 데이터 양과 동일하거나 약간 큰 경우 파일 시스템 성능이 향상되며 데이터를 보다 효율적으로 저장할 수 있습니다. 파일 크기가 작아도 전체 블록이 사용됩니다. 파일은 여러 블록에 걸쳐 분산될 수 있지만 이는 추가 런타임 오버헤드를 일으킬 수 있습니다. 또한 일부 파일 시스템은 특정 블록 수에 제한되어 파일 시스템의 최대 크기가 제한될 수 있습니다.

mkfs 명령으로 장치를 포맷할 때 블록 크기는 파일 시스템 옵션으로 지정됩니다. 블록 크기를 지정하는 매개 변수는 파일 시스템에 따라 다릅니다. 파일 시스템에 대한 보다 자세한 내용은 **mkfs** man 페이지에서 참조하십시오. 예를 들어 XFS 파일 시스템 포맷 시 사용할 수 있는 옵션을 확인하려면 다음 명령을 실행합니다.

```
$ man mkfs.xfs
```

지오메트리

파일 시스템 지오메트리는 파일 시스템에 데이터를 배분하는 것과 관련되어 있습니다. 시스템이 RAID와 같은 스트라이프 형 스토리지를 사용하는 경우 장치 포맷 시 기본이 되는 스토리지 지오메트리로 데이터 및 메타 데이터를 정렬하여 성능을 개선할 수 있습니다.

대부분의 장치는 장치를 특정 파일 시스템으로 포맷할 때 자동으로 설정되는 권장 지오메트리를 내보냅니다. 장치가 권장 지오메트리를 내보내기하지 않거나 권장 설정을 변경하고자 할 경우 **mkfs**로 장치를 포맷할 때 수동으로 지오메트리를 지정해야 합니다.

파일 시스템 지오메트리를 지정하는 매개 변수는 파일 시스템에 따라 다릅니다. 자세한 내용은 파일 시스템의 **mkfs** man 페이지에서 참조하십시오. 예를 들어 ext4 파일 시스템을 포맷할 때 사용 가능한 옵션을 확인하려면 다음 명령을 실행합니다.

```
$ man mkfs.ext4
```

외부 저널

파일 시스템 저널링은 쓰기 작업이 실행되기 전 쓰기 작업 동안 저널 파일에서의 변경 사항을 기록합니다. 이는 시스템 충돌이나 정전이 발생한 경우 스토리지 장치의 손상 가능성을 줄이고 복구 프로세스를 속도를 높입니다.

메타데이터 집약적 워크로드에서는 매우 자주 저널에 업데이트됩니다. 저널이 클수록 메모리 사용량이 높아지지만 쓰기 작업의 빈도는 줄어듭니다. 또한 메타데이터 집약적 워크로드가 있는 장치 탐색 시간을 개선하려면 저널을 기본 스포리지보다 더 빠른 전용 스토리지에 배치합니다.

주의

신뢰할 수 있는 외부 저널인지 확인합니다. 외부 저널 장치가 손실되면 파일 시스템이 손상됩니다.

외부 저널은 마운트 시 지정된 저널 장치와 함께 포맷 시 생성해야 합니다. 보다 자세한 내용은 **mkfs** 및 **mount** man 페이지에서 참조하십시오.

```
$ man mkfs
```

```
$ man mount
```

5.1.4.2. 마운트 시 고려할 사항

다음 부분에서는 대부분의 파일 시스템에 적용할 수 있고 장치 마운트 시 지정할 수 있는 튜닝에 대해 설명합니다.

장애

파일 시스템 장애로 인해 파일 시스템 메타데이터가 영구 스토리지에 정확히 기록되어 순서대로 정렬되어 있고 정전 시 **fsync**로 전송된 데이터가 유지되도록 합니다. 이전 Red Hat Enterprise Linux 버전에서 파일 시스템 장애를 활성화하면 **fsync**에 크게 의존하는 애플리케이션 속도 또는 생성/삭제된 여러 작은 파일의 속도가 현저히 저하될 수 있습니다.

Red Hat Enterprise Linux 7에서 파일 시스템 장애 성능이 향상되어 파일 시스템 장애 비활성화로 인해 성능에 미치는 영향을 무시할 수 있습니다. (3% 미만)

보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 *Red Hat Enterprise Linux 7 Storage Administration Guide*에서 참조하십시오.

액세스 시간

파일을 읽을 때 마다 메타데이터는 액세스 시간 (**atime**)을 업데이트합니다. 이에는 추가 쓰기 I/O가 포함됩니다. 기본적으로 Red Hat Enterprise Linux 7은 이전 액세스 시간이 마지막 변경 시간 (**mtime**) 또는 상태 변경 시간 (**ctime**) 보다 오래된 경우에만 **atime** 필드를 업데이트하므로 대부분의 경우 오버헤드가 최소화됩니다.

하지만 메타데이터 업데이트에 시간이 소요되고 정확한 액세스 시간 데이터가 필요하지 않은 경우 **noatime** 마운트 옵션을 사용하여 파일 시스템을 마운트할 수 있습니다. 이는 파일을 읽을 때 메타데이터로의 업데이트를 비활성화합니다. 또한 **nodiratime** 동작을 활성화하여 디렉토리를 읽을 때 메타데이터로의 업데이트를 비활성화할 수 있습니다.

미리 읽기

미리 읽기 동작은 곧 필요한 데이터를 먼저 불러와서 페이지 캐시에 로드하여 디스크에서 더 빨리 데이터를 검색할 수 있기 때문에 파일 액세스 속도가 빨라집니다. 미리 읽기 값이 높을 수록 더 많은 데이터를 먼저 가져옵니다.

Red Hat Enterprise Linux는 파일 시스템의 감지 대상에 따라 적절한 미리 읽기 값을 설정하려 합니다. 하지만 항상 정확한 미리 읽기 값을 설정하는 것은 불가능합니다. 예를 들어 스토리지 어레이가 단일 LUN으로 시스템에 표시되면 시스템은 단일 LUN을 감지하고 어레이에 적절한 미리 읽기 값을 설정하지 않습니다.

연속 I/O에 의한 대량의 스트리밍이 필요한 워크로드는 높은 미리 읽기 값을 통해 이점을 얻을 수 있습니다. Red Hat Enterprise Linux 7에서의 스토리지 관련 조정 프로파일은 LVM 스트라이프를 사용하므로 미리 읽기 값이 높게 설정되어 있지만 이러한 조정 값이 모든 워크로드에 항상 충분하지는 않습니다.

미리 읽기 동작을 지정하는 매개 변수는 파일 시스템에 따라 다릅니다. 보다 자세한 내용은 mount man 페이지에서 참조하십시오.

```
$ man mount
```

5.1.4.3. 유지 관리

SSD (solid-state disks) 및 신 프로비저닝 스토리지 모두의 파일 시스템에서 사용되고 있지 않은 블록을 정기적으로 삭제할 것을 권장합니다. 사용되지 않는 블록을 삭제하는 데는 일괄 삭제 및 온라인 삭제라는 두 가지 방법이 있습니다.

일괄 삭제

이러한 유형의 삭제는 **fstrim** 명령의 일부입니다. 이는 관리자가 지정한 기준과 일치하는 파일 시스템에서 사용하지 않는 모든 블록을 삭제합니다.

Red Hat Enterprise Linux 7에서는 물리적 삭제 작업을 지원하는 XFS 및 ext4 포맷 장치에서 일괄 삭제를 지원합니다. (즉, **/sys/block/*devname*/queue/discard_max_bytes** 값이 0이 아닌 HDD 장치 및 **/sys/block/sda/queue/discard_granularity** 값이 **0**가 아닌 SSD 장치)

온라인 삭제

이러한 유형의 삭제 동작은 마운트 시 **discard** 옵션을 사용하여 설정되며 사용자 개입없이 실시간으로 실행됩니다. 하지만 온라인 삭제는 사용됨 상태 에서 사용 가능 상태로 전환하고 있는 블록만을 삭제하지 않습니다. Red Hat Enterprise Linux 7에서는 XFS 및 ext4 포맷 장치에서 온라인 삭제를 지원합니다.

Red Hat은 성능 유지를 위해 온라인 삭제를 해야하거나 시스템의 워크로드에 일괄 삭제를 수행할 수 없는 경우를 제외하고 일괄 삭제를 사용할 것을 권장합니다.

사전 할당

사전 할당은 디스크 공간에 데이터를 기입하지 않고 디스크 공간이 파일에 할당되어 있음을 표시합니다. 이는 데이터 조각화 및 읽기 성능 저하를 제한하는 경우 유용합니다. Red Hat Enterprise Linux 7에서는 마운트시 XFS, ext4, GFS2 장치에 사전 할당 공간을 지원합니다. 파일 시스템에 적합한 매개 변수는 **mount** man 페이지에서 확인하십시오. 애플리케이션은 **fallocate(2) glibc** 호출을 사용하여 공간을 미리 할당할 수 있습니다.

5.2. 성능 관련 문제 모니터링 및 진단

Red Hat Enterprise Linux 7에서는 시스템 성능을 모니터링하고 시스템 프로세스 및 설정과 관련된 성능 문제를 진단하는데 유용한 여러 도구가 있습니다. 다음 부분에서는 프로세서 관련 성능 문제를 모니터링하고 진단하기 위해 사용 가능한 도구 및 사용 방법에 대해 설명합니다.

5.2.1. vmstat로 시스템 성능 모니터링

Vmstat는 시스템 전체의 프로세스, 메모리, 페이징, I/O 블록, 인터럽트, CPU 동작에 대해 보고합니다. 이를 통해 관리자는 I/O 서브 시스템이 성능 문제에 관련있는지를 판단할 수 있습니다.

I/O 성능 관련 정보는 다음에서 확인하십시오:

si

스왑인 또는 스왑 영역에 쓰기 (KB 단위)

so

스왑 아웃 또는 스왑 영역에서 읽기 (KB 단위)

bi

블록인 또는 쓰기 동작 블록 (KB 단위)

bo

블록 아웃 또는 읽기 동작 블록 (KB 단위)

wa

I/O 작업 완료를 기다리는 큐 부분

스왑인과 스왑 아웃 기능은 스왑 영역 및 데이터가 동일한 장치에 있는 경우 메모리 사용량을 표시하는데 특히 유용합니다.

또한 free, buff, cache 칼럼은 쓰기 저장 빈도를 확인하는데 유용합니다. cache 값이 갑자기 내려가고 free 값이 증가하는 경우 쓰기 저장 및 페이지 캐시 무효화가 시작되었음을 나타내는 것입니다.

vmstat 분석에서 I/O 서브시스템이 성능 저하와 관련된 것으로 표시되는 경우 관리자는 **iostat**를 사용하여 I/O 장치를 지정할 수 있습니다.

vmstat는 *procps-ng* 패키지로 제공됩니다. **vmstat**에 관한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man vmstat
```

5.2.2. iostat로 I/O 성능 모니터링

Iostat는 *sysstat* 패키지에서 제공됩니다. 이는 시스템에서 I/O 장치 로드를 보고합니다. **vmstat** 분석에서 I/O 서브시스템이 성능 저하와 관련된 것으로 표시되는 경우 관리자는 **iostat**를 사용하여 I/O 장치를 지정할 수 있습니다.

iostat man 페이지에 지정된 매개 변수를 사용하여 특정 장치의 **iostat** 출력 보고에 중점을 둘 수 있습니다:

```
$ man iostat
```

5.2.2.1. blktrace로 고급 I/O 분석

Blktrace는 I/O 서브시스템에서 시간이 배분된 방법에 대한 상세 정보를 제공합니다. 제공된 유틸리티 **blkparse**는 **blktrace**에서 원시 출력을 읽고 **blktrace**에 의해 기록되는 입/출력 동작 요약을 읽기 쉬운 형태로 제공합니다.

이 도구에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man blktrace
```

```
$ man blkparse
```

5.2.2.2. btt로 blktrace 출력 분석

Btt는 *blktrace* 패키지의 일부로 제공됩니다. 이는 **blktrace** 출력을 분석하여 각각의 I/O 스택 영역에서 데이터가 소비하는 시간을 표시하므로 I/O 하위 시스템에서 병목 현상 지점을 쉽게 감지할 수 있습니다.

예를 들어 **btt**에서 블록 계층 (**Q2Q**)에 전송된 요청 사이의 시간이 블록 계층 (**Q2C**)에서 소비한 총 요청 시간 보다 길게 표시될 경우 I/O 서브 시스템은 성능 문제와 연관이 없을 수 있습니다. 장치가 요청 (**D2C**)을 처리하는데 소요되는 시간이 긴 경우 장치는 과부하되거나 장치에 전송된 워크로드가 최적의 것이 아님을 의미하는 것일 수 있습니다. 블록 I/O가 요청 (**Q2G**)을 지정하기 전 오랜 시간 동안 대기 중인 경우 사용 중인 스토리지가 I/O 로드를 처리할 수 없음을 의미합니다.

이 도구에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man btt
```

5.2.2.3. seekwatcher로 blktrace 출력 분석

seekwatcher 도구는 **blktrace** 출력을 사용하여 I/O 시간 경과를 그래프화할 수 있습니다. 이는 디스크 I/O의 LBA (Logical Block Address), 초당 처리 능력 (MB 단위), 초당 검색 수, 초당 I/O 작업 수에 중점을 둡니다. 이는 초당 작업 수가 장치의 한계에 도달했는지 확인하는데 유용합니다.

이 도구에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man seekwatcher
```

5.2.3. SystemTap을 사용한 스토리지 모니터링

*Red Hat Enterprise Linux 7 SystemTap Beginner's Guide*에는 스토리지 성능을 모니터링 및 프로파일링하는데 유용한 여러 스크립트 예제가 들어 있습니다.

다음의 **SystemTap** 스크립트 예제는 스토리지 성능과 관련되어 있으며 스토리지나 파일 시스템 성능 문제를 진단하는데 유용합니다. 이는 기본적으로 **/usr/share/doc/systemtap-client/examples/io** 디렉토리에 설치됩니다.

disktop.stp

5 초 마다 디스크 읽기/쓰기 상태를 확인하고 상위 10 개 항목을 출력합니다.

iotime.stp

읽기 및 쓰기 동작에 소요된 시간과 읽기 및 쓰기 바이트 수를 출력합니다.

traceio.stp

매 초 마다 누적된 I/O 트랙픽에 따라 상위 10 개의 실행 파일을 출력합니다.

`traceio2.stp`

지정한 장치에 읽기 및 쓰기가 발생하면 실행 파일 이름 및 프로세스 ID를 출력합니다.

`inodewatch.stp`

지정된 메이저/마이너 장치에 지정된 inode에 읽기 또는 쓰기가 발생할 때 마다 실행 파일 이름 및 프로세스 ID를 출력합니다.

`inodewatch2.stp`

지정된 메이저/마이너 장치에 지정된 inode에 속성이 변경할 때마다 실행 파일 이름, 프로세스 ID, 속성을 출력합니다.

*Red Hat Enterprise Linux 7 SystemTap Beginner's Guide*는 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에서 확인하실 수 있습니다.

5.3. 설정 도구

Red Hat Enterprise Linux에는 스토리지 및 파일 시스템을 설정할 때 관리자에게 유용한 여러 도구가 있습니다. 다음 부분에서는 이러한 도구에 대해 설명하고 Red Hat Enterprise Linux 7에서 I/O 및 파일 시스템 관련 성능 문제를 해결하는 방법에 대해 설명합니다.

5.3.1. 스토리지 성능의 프로파일 튜닝 설정

Tuned 및 **tuned-adm**은 특정 사용 경우에서 성능을 개선하기 위한 여러 프로파일을 제공합니다. 특히 다음 프로파일은 스토리지 성능을 개선하는데 유용합니다.

- latency-performance
- throughput-performance (기본)

시스템에 프로파일을 설정하려면 다음 명령을 실행합니다. 여기서 *name*은 사용하고자 하는 프로파일 이름으로 변경합니다.

```
$ tuned-adm profile name
```

`tuned-adm recommend` 명령은 시스템에 적합한 프로파일을 권장합니다. 또한 이는 설치 시 시스템에 기본 프로파일을 설정하여 기본 프로파일로 돌아갈 때 사용할 수 있습니다.

이러한 프로파일이나 추가 설정 옵션에 대한 보다 자세한 내용은 A.6절. “tuned-adm”에서 참조하십시오.

5.3.2. 기본 I/O 스케줄러 설정

기본 I/O 스케줄러는 장치의 마운트 옵션에 지정된 스케줄러가 없을 경우 사용되는 스케줄러입니다.

기본 I/O 스케줄러를 설정하려면 부팅시 커널 명령행에 엘리베이터 매개 변수를 추가하거나 **`/etc/grub2.conf`** 파일을 편집하여 사용하려는 스케줄러를 설정합니다.

```
elevator=scheduler_name
```

5.3.3. 장치의 I/O 스케줄러 설정

특정 스토리지 장치의 스케줄러나 스케줄러 우선 순위를 설정하려면 **/sys/block/*devname*/queue/scheduler** 파일을 편집합니다. 여기서 *devname*은 설정하려는 장치 이름으로 변경합니다.

```
# echo cfq > /sys/block/hda/queue/scheduler
```

5.3.4. deadline 스케줄러 튜닝

deadline이 사용 중일 때 대기 큐에 있는 I/O 요청은 읽기 또는 쓰기 배치로 나뉘어 LBA 오름 차순으로 실행이 예약됩니다. 애플리케이션은 읽기 I/O에서 블록될 가능성이 높기 때문에 기본적으로 읽기 배치가 쓰기 배치보다 우선합니다. 배치 처리 후 **deadline**은 쓰기 작업의 프로세서 대기 시간을 확인하고 다음의 읽기 또는 쓰기 배치를 적절히 예약합니다.

다음 매개 변수는 **deadline** 스케줄러 동작에 영향을 미칠 수 있습니다.

fifo_batch

단일 배치에서 발생하는 읽기 또는 쓰기 작업 수입니다. 기본값은 **16**입니다. 값이 높을 수록 처리 능력도 증가하지만 대기 시간도 늘어납니다.

front_merges

워크로드가 전면 병합을 생성하지 않은경우 이 매개 변수를 **0**으로 설정할 수 있습니다. 하지만 이러한 검사의 오버헤드를 측정하지 않는 한 Red Hat은 기본 설정 **1**을 그대로 사용할 것을 권장합니다.

read_expire

읽기 요청을 실행할 시간을 밀리초 단위로 예약 설정할 수 있습니다. 기본값은 **500** (0.5 초)입니다.

write_expire

쓰기 요청을 실행할 시간을 밀리초 단위로 예약 설정할 수 있습니다. 기본값은 **5000** (5 초)입니다.

writes_starved

쓰기 배치를 처리하기 전 처리할 수 있는 읽기 배치 수를 지정합니다. 이 값을 높게 설정할 수록 읽기 배치가 우선적으로 처리됩니다.

5.3.5. cfq 스케줄러 튜닝

cfq를 사용하고 있을 경우 프로세스는 실시간, 최선형, 유휴 상태의 3 가지 다른 클래스에 배치됩니다. 실시간 프로세스는 최선형 프로세스보다 먼저 스케줄 예약되고 최선형 프로세스는 유휴 상태 프로세스보다 먼저 스케줄 예약됩니다. 기본값으로 프로세스는 최선형 클래스에 배치됩니다. **ionice** 명령을 사용하여 프로세스를 수동으로 조정할 수 있습니다.

다음 매개 변수를 사용하여 **cfq** 스케줄러 동작을 추가로 설정할 수 있습니다. 이러한 매개 변수는 **/sys/block/*devname*/queue/iosched** 디렉토리 아래에 지정된 파일에서 장치 별로 설정을 변경합니다.

back_seek_max

cfq는 역방향 검색을 수행하는 최대 거리를 킬로바이트 단위로 지정합니다. 기본값은 **16** KB입니다. 일반적으로 역방향 검색은 성능에 지장이 될 수 있으므로 큰 값을 사용하는 것을 권장하지 않습니다.

back_seek_penalty

디스크 헤드를 앞 또는 뒤로 이동할 지를 지정할 때 역방향 검색에 적용할 승수를 지정합니다. 기본값은 **2**입니다. 디스크 헤드 위치가 1024 KB이고 시스템에 등거리 요청 (예: 1008 KB 및 1040 KB)이 있을 경우 ***back_seek_penalty***는 역박향 탐색거리에 적용되며 디스크는 앞으로 이동합니다.

fifo_expire_async

비동기 (버퍼 쓰기) 요청이 처리되지 않은 상태로 남아 있을 수 있는 시간을 밀리초 단위로 지정합니다. 지정된 시간이 경과하면 단일 비동기 처리 요청은 디스패치 목록으로 이동합니다. 기본값은 **250** 밀리초입니다.

fifo_expire_sync

동기화 (읽기 또는 **O_DIRECT** 쓰기) 요청이 최리되지 않은 상태로 남아 있을 수 있는 시간을 밀리초 단위로 지정합니다. 지정된 시간이 경과하면 단일 동기화 처리 요청은 디스패치 목록으로 이동합니다. 기본값은 **125** 밀리초입니다.

group_idle

이 매개 변수는 기본적으로 **0** (비활성화)로 설정됩니다. **1** (활성화)로 설정되어 있을 경우 **cfq** 스케줄러는 제어 그룹에서 I/O를 발행하는 마지막 프로세스에서 유휴 상태가 됩니다. 비례 가중 I/O 제어 그룹을 사용하고 ***slice_idle***이 **0**으로 (고속 스토리지) 설정되어 있을 때 유용합니다.

group_isolation

이 매개 변수는 기본적으로 **0** (비활성화)로 설정되어 있습니다. **1** (활성화)로 설정되어 있을 경우 그룹 간의 분리를 강화하지만 연속 및 임의의 워크로드 모두에 공정성이 적용되므로 처리 능력이 감소됩니다. ***group_isolation***이 비활성화 (**0** 설정)되어 있을 경우 공정성은 연속 워크로드에만 제공됩니다. 보다 자세한 내용은 **/usr/share/doc/kernel-doc-version/Documentation/cgroups/blkio-controller.txt**에 설치된 문서에서 참조하십시오.

low_latency

이 매개 변수는 기본적으로 **1** (활성화)로 설정되어 있습니다. 활성화되어 있을 경우 **cfq**는 장치에 I/O를 발행하는 각 프로세스에 대해 최대 **300** ms로 대기 시간을 제공하여 처리량 보다 공정성을 우선으로 합니다. 이 매개 변수가 **0** (비활성화)로 설정되어 있을 경우 대상 지연 시간은 무시되고 각 프로세스는 전체 타임 슬라이스를 받습니다.

quantum

이 매개 변수는 **cfq**가 한 번에 하나의 장치에 보낼 수 있는 I/O 요청 수를 지정하며, 기본적으로 큐 깊이를 제한합니다. 기본값은 **8**입니다. 사용 장치는 더 깊은 큐 깊이를 지원할 수 있지만 quantum을 증가시키면 대기 시간이 늘어나며 특히 대량의 연속 쓰기 워크로드가 있을 경우 더욱 그러합니다.

slice_async

이 매개 변수는 비동기 I/O 요청을 발행하는 각 프로세스에 할당할 타임 슬라이스 (밀리초 단위) 길이를 지정합니다. 기본값은 **40** 밀리초입니다.

slice_idle

이는 추가 요청을 기다리는 동안 cfq의 유휴 상태 시간을 밀리초 단위로 지정합니다. 기본값은 **0** (큐 또는 서비스 트리 레벨에서 유휴 상태가 아님)입니다. 외부 RAID 스토리지에서의 처리량의 경우 기본값을 사용하는 것이 좋지만 이는 전체 검색 수를 증가시키므로 내부의 비 RAID 스토리지에서의 처리량을 저하시킬 수 있습니다.

slice_sync

이 매개 변수는 비동기 I/O 요청을 발행하는 각 프로세스에 할당할 타임 슬라이스 (밀리초 단위) 길이를 지정합니다. 기본값은 **100** 밀리초입니다.

5.3.5.1. 고속 스토리지에 cfq 튜닝

외장형 스토리지 어레이나 SSD와 같은 대형 탐색 패널티의 영향을 받지 않는 하드웨어의 경우 **cfq** 스케줄러 사용을 권장하지 않습니다. 이러한 스토리지에서 **cfq**를 사용해야 할 경우 다음 설정 파일을 편집해야 합니다:

- **/sys/block/*devname*/queue/ionice/slice_idle**을 **0**로 설정
- **/sys/block/*devname*/queue/ionice/quantum**을 **64**로 설정
- **/sys/block/*devname*/queue/ionice/group_idle**을 **1**로 설정

5.3.6. noop 스케줄러 튜닝

noop I/O 스케줄러는 주로 고속 스토리지를 사용하는 CPU 바인딩 시스템에 유용합니다. 요청은 블록층에서 병합되므로 **noop** 동작을 수정하려면 **/sys/block/sdX/queue/** 디렉토리 아래에 있는 파일에서 블록 층 매개 변수를 편집합니다.

add_random

일부 I/O 이벤트는 **/dev/random**의 엔트로피 풀에 적용됩니다. 이러한 적용의 오버헤드가 측정 가능할 경우 이 매개 변수를 **0**로 설정할 수 있습니다.

max_sectors_kb

최대 I/O 요청 크기를 KB 단위로 지정합니다. 기본값은 **512** KB입니다. 이 매개 변수의 최소값은 스토리지 장치의 논리적 블록 크기에 따라 지정됩니다. 이 매개 변수의 최대값은 ***max_hw_sectors_kb*** 값에 따라 지정됩니다.

I/O 요청이 내부 소거 블록 크기 보다 크면 일부 SSD 성능이 저하됩니다. 이러한 경우 Red Hat은 ***max_hw_sectors_kb***를 내부 소거 블록 크기로 줄일 것을 권장합니다.

nomerges

요청 병합은 대부분의 워크로드에 효과적이지만 디버깅 목적의 경우 병합을 비활성화하는것이 유용합니다. 이 매개 변수를 **0**로 설정하여 병합을 비활성화합니다. 기본값으로 이는 비활성화 (**1**으로 설정)되어 있습니다.

nr_requests

한 번에 대기 큐에 둘 수 있는 최대 읽기 및 쓰기 요청 수를 지정합니다. 기본값은 **128**이며 이는 다음의 읽기 또는 쓰기 요청 프로세스를 수면 상태로 두기 전 128 개의 읽기 요청과 128 개의 쓰기 요청을 대기 큐에 둘 수 있음을 의미합니다.

대기 시간에 민감한 애플리케이션의 경우 이 매개 변수의 값을 낮추고 스토리지의 명령 큐 깊이를 제한하면 다시 쓰기 I/O는 쓰기 요청으로 장치 큐를 채울 수 없습니다. 장치 큐가 채워지면 I/O 동작을 실행하려고 하는 다른 프로세스는 큐를 사용할 수 있을 때 까지 수면 상태로 됩니다. 요청은 라운드 로빈 방식으로 할당되기 때문에 하나의 프로세스가 큐에서 지속적으로 소비되지 않게 합니다.

optimal_io_size

일부 스토리지 장치는 이러한 매개 변수를 통해 최적의 I/O 크기를 보고합니다. 이러한 값이 보고되면 Red Hat은 애플리케이션이 가능한 최적의 I/O 크기로 조정되어 그 배수의 I/O를 실행할 것을 권장합니다.

read_ahead_kb

페이지 캐시에서 곧 필요한 정보를 저장하기 위해 연속적인 읽기 동작 동안 운영 체제가 미리 읽기 할 용량 (KB)을 지정합니다. 장치 매퍼는 ***read_ahead_kb*** 값을 높게 지정하여 이점을 갖는 경우가 종종 있습니다. 각 장치의 매핑 값을 128 KB로 시작하는 것이 좋습니다.

rotational

일부 SSD는 솔리드 스테이트 상태를 올바르게 보고하지 않아 기존 회전 디스크로 마운트됩니다. SSD 장치가 이 값을 자동으로 **0**으로 설정하지 않으면 이를 수동으로 설정하여 스케줄러에서 필요하지 않은 검색 시간 단축 설정을 비활성화합니다.

rq_affinity

기본적으로 I/O 요청을 발행한 프로세서가 아닌 다른 프로세서에서 I/O 완료를 처리할 수 있습니다. ***rq_affinity***를 **1**로 설정하여 이 기능을 해제하고 I/O 요청을 발행하는 프로세서에서만 I/O 완료를 처리하도록 합니다. 이는 프로세서 데이터 캐시의 효율성을 향상시킬 수 있습니다.

5.3.7. 성능 개선을 위한 파일 시스템 설정

다음 부분에서는 Red Hat Enterprise Linux 7에서 지원되는 각 파일 시스템의 특정 매개 변수 튜닝에 대해 설명합니다. 매개 변수는 스토리지 장치를 포맷할 때 매개 변수 값을 설정하는 경우와 포맷된 장치에 마운트할 때 매개 변수를 설정하는 경우로 나뉘어집니다.

파일 조각화 또는 리소스 경합으로 인해 성능 저하가 발생하는 경우 파일 시스템을 재설정하여 성능을 향상시킬 수 있습니다. 하지만 일부 경우 애플리케이션에 변경을 해야 하는 경우가 있습니다. 이러한 경우 Red Hat은 고객 지원팀에 문의할 것을 권장합니다.

5.3.7.1. XFS 튜닝

다음 부분에서는 XFS 파일 시스템 포맷이나 마운트 시 사용할 수 있는 일부 튜닝 매개 변수에 대해 설명합니다.

대부분의 워크로드에서 기본 XFS 포맷과 마운트 설정을 사용할 수 있습니다. Red Hat은 워크로드에 특정 설정 변경이 필요한 경우에만 변경할 것을 권장합니다.

5.3.7.1.1. 포맷 옵션

포맷 옵션에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man mkfs.xfs
```

디렉토리 블록 크기

디렉토리 블록 크기는 I/O 동작 마다 검색 또는 수정할 수 있는 디렉토리 정보 양에 영향을 미칩니다. 디렉토리 블록 크기의 최소 값은 파일 시스템 블록 크기 (기본4 KB)입니다. 최대 디렉토리 블록 크기 값은 **64** KB입니다.

지정된 디렉토리 블록 크기에서 큰 디렉토리는 작은 디렉토리보다 많은 I/O를 필요로 합니다. 큰 디렉토리 블록 크기의 시스템이 작은 디렉토리 블록 크기의 시스템 보다 I/O 동작 당 처리량이 높습니다. 따라서 워크로그에서 디렉토리 및 디렉토리 블록 크기를 가능한 작게 할 것을 권장합니다.

Red Hat은 대량의 쓰기 작업이 및 읽기 작업이 있는 워크로드의 항목 수를 초과하지 않도록 파일 시스템에 대해 표 5.1. “디렉토리 블록 크기에 권장하는 최대 디렉토리 항목 수 ”에 나열된 디렉토리 블록 크기를 사용할 것을 권장하고 있습니다.

표 5.1. 디렉토리 블록 크기에 권장하는 최대 디렉토리 항목 수

디렉토리 블록 크기	최대 항목 수 (대량의 읽기 작업)	최대 항목 수 (대량의 쓰기 작업)
4 KB	100000–200000	1000000–2000000
16 KB	100000–1000000	1000000–10000000
64 KB	>1000000	>10000000

다른 크기의 파일 시스템에 있는 읽기 및 쓰기 워크로드에서 디렉토리 블록 크기에 미치는 영향에 대한 보다 자세한 내용은 XFS 문서에서 참조하십시오.

디렉토리 블록 크기를 설정하려면 **`mkfs.xfs -l`** 옵션을 사용합니다. 보다 자세한 내용은 **`mkfs.xfs`** man 페이지에서 참조하십시오.

할당 그룹

할당 그룹은 독립적인 구조로 파일 시스템 섹션에 할당된 inode 및 여유 공간의 인덱스를 생성합니다. 각 할당 그룹은 독립적으로 수정할 수 있기 때문에 XFS는 동시에 작업을 수행하는것이 다른 할당 그룹에 영향을 미치는 경우 할당 및 할당 해제 작업을 동시에 수행할 수 있습니다. 따라서 파일 시스템에서 실행할 수 있는 동시 작업 수는 할당 그룹 수와 동일합니다. 하지만 동시 작업을 실행할 수 있는 기능은 작업을 실행할 수 있는 프로세서 수에 의해 제한되므로 Red Hat은 할당 그룹 수를 시스템의 프로세서 수와 동일하게 또는 그 이상으로 할 것을 권장합니다.

단일 디렉토리는 여러 할당 그룹에서 동시에 변경할 수 없습니다. 따라서 Red Hat은 대량의 파일을 생성 및 삭제하는 애플리케이션은 단일 디렉토리에 모든 파일을 저장하지 못하게 할 것을 권장합니다.

할당 그룹을 설정하려면 **`mkfs.xfs -d`** 옵션을 사용합니다. 보다 자세한 내용은 **`mkfs.xfs`** man 페이지에서 참조하십시오.

증가 제약 조건

포맷 후 파일 시스템 크기를 증가시켜야 할 경우 (더 많은 하드웨어를 추가하거나 씬 프로비저닝을 통해), 포맷 완료 후 할당 그룹 크기를 변경할 수 없으므로 초기 파일 레이아웃을 주의 깊게 살펴보아야 합니다.

할당 그룹 크기는 초기 파일 시스템 용량이 아니라 최종 파일 시스템 용량에 따라 지정해야 합니다. 완전히 증가한 파일 시스템에 있는 할당 그룹 수는 할당 그룹이 최대 크기 (1 TB)가 아니면 수백을 초과해서는 안됩니다. 따라서 대부분의 파일 시스템에서 증가할 수 있는 최대 권장 크기는 초기 크기의 10배입니다.

RAID 어레이에서 파일 시스템을 증가시킬 때 장치 크기는 할당 그룹 크기의 배수로 새로운 할당 그룹 헤더가 새로 추가된 스토리지에 정렬되어야 하므로 추가적으로 주의를 기울여야 합니다. 새로운 스토리지는 기존 스토리지와 동일한 배열을 갖게 해야 하므로 포맷 후 배열을 변경할 수 없기 때문에 동일한 블록 장치에서 다른 배열의 스토리지를 최적화할 수 없습니다.

Inode 크기 및 인라인 속성

Inode에 사용 가능한 공간이 충분할 경우 XFS는 속성 이름과 값을 inode에 직접 쓸 수 있습니다. 이러한 인라인 속성은 추가 I/O가 필요하지 않기 때문에 별도의 속성 블록을 검색하는것 보다 신속하게 검색 및 변경할 수 있습니다.

기본 inode 크기는 256 바이트입니다. 여기서 약 100 바이트만 inode에 저장되는 데이터 범위 포인터 수에 따라 속성 스토리지에 사용할 수 있습니다. 파일 시스템 포맷 시 inode 크기를 증가시키면 스토리지 속성에 사용할 수 있는 공간 크기도 증가됩니다.

속성 이름 및 속성 값 모두 최대 254 바이트로 제한되어 있습니다. 속성 이름이나 속성 값 중 하나의 길이가 254 바이트를 초과하면 속성은 인라인에 저장되지 않고 별도의 속성 블록에 푸시됩니다.

inode 매개 변수를 설정하려면 **mkfs.xfs -i** 옵션을 사용합니다. 보다 자세한 내용은 **mkfs.xfs** man 페이지에서 참조하십시오.

RAID

소프트웨어 RAID를 사용하고 있는 경우 **mkfs.xfs**는 적절한 스트라이프 단위 및 너비로 기본 하드웨어를 자동으로 설정합니다. 하지만 스트라이프 단위 및 너비는 하드웨어 RAID가 사용 중일 경우 모든 하드웨어 RAID 장치가 이러한 정보를 내보내지 않으므로 수동으로 설정해야 할 필요가 있을 수 있습니다. 스트라이프 단위 및 너비를 설정하려면 **mkfs.xfs -d** 옵션을 사용합니다. 보다 자세한 내용은 **mkfs.xfs** man 페이지에서 참조하십시오.

로그 크기

보류중인 변경 사항은 로그에 기록될 부분에 동기화 이벤트가 발생할 때 까지 메모리에 집계됩니다. 로그 크기에 따라 한번에 실행할 수 있는 동시 수정 수를 지정합니다. 또한 메모리에 집계할 수 있는 최대 변경 크기를 지정하여 디스크에 데이터가 기록되는 빈도를 지정합니다. 로그가 작을 수록 데이터가 디스크에 기록되는 빈도가 많아집니다. 하지만 로그가 큰 경우 보류 중인 변경 내용을 기록하기 위해 큰 메모리를 사용하기 때문에 메모리 제한이 있는 시스템은 큰 로그를 사용해도 이점이 없습니다.

로그는 기본 스트라이프 단위로 정렬될 경우 성능이 향상됩니다. 즉 로그 스트라이프 단위의 경계에서 시작하고 종료하는 것입니다. 로그를 스트라이프 단위로 정렬하려면 **mkfs.xfs -d** 옵션을 사용합니다. 보다 자세한 내용은 **mkfs.xfs** man 페이지에서 참조하십시오.

로그 크기를 설정하려면 다음의 **mkfs.xfs** 옵션을 사용합니다. 여기서 *logsize*는 로그 크기로 변경합니다:

```
# mkfs.xfs -l size=logsize
```

보다 자세한 내용은 **mkfs.xfs** man 페이지에서 참조하십시오:

```
$ man mkfs.xfs
```

로그 스트라이프 단위

RAID5 또는 RAID6 레이아웃을 사용하여 스토리지 장치에 기록할 로그는 스트라이프 단위 경계에서 시작하고 종료 (기본 스트라이프에 정렬)할 때 성능이 향상됩니다. **mkfs.xfs**는 적절한 로그 스트라이프 단위로 자동 설정 시도하지만 이러한 정보를 내보내는 RAID 장치에 따라 다릅니다.

동기화 이벤트가 자주 발생하는 워크로드의 경우 로그 스트라이프 단위를 크게 설정하면 성능에 문제가 될 수 있습니다. 기록이 작으면 로그 스트라이프 단위 크기를 채워야 하기 때문에 대기 시간이 증가될 수 있기 때문입니다. 워크로드가 로그 기록 대기 시간에 제약을 받는 경우 Red Hat은 로그 스트라이프 단위를 1 블록에 설정하여 정렬되지 않은 로그 기록을 시작하게 할 것을 권장합니다.

지원되는 로그 스트라이프 단위의 최대 크기는 최대 로그 버퍼 크기 (256 KB)입니다. 따라서 기본 스토리지는 로그에 설정할 수 있는 스트라이프 단위 보다 큰 스트라이프 단위를 갖게 할 수 있습니다. 이러한 경우 **mkfs.xfs**는 경고를 발행하고 32 KB의 로그 스트라이프 단위가 설정됩니다.

로그 스트라이프 단위를 설정하려면 다음 옵션 중 하나를 사용합니다. 여기서 *N*은 스트라이프 단위로 사용할 블록 수로 변경하고 *size*는 스트라이프 단위 크기를 KB로 입력합니다.

```
mkfs.xfs -l sunit=Nb
mkfs.xfs -l su=size
```

보다 자세한 내용은 **mkfs.xfs** man 페이지에서 참조하십시오:

```
$ man mkfs.xfs
```

5.3.7.1.2. 마운트 옵션

Inode 할당

- compress=zlib - 더 나은 압축 비율입니다. 이전 커널에 대해 기본값이며 사용 안전합니다. (기본값)
- compress=lzo - 압축 속도는 빠르지만 zlib 만큼 압축하지 않습니다.
- compress=no - 압축을 비활성화합니다. (커널 3.6부터 사용 가능)
- compress-force=method - 비디오나 dd 디스크 이미지와 같이 압축이 잘 되지 않는 파일의 경우 이러한 압축을 사용합니다. 사용 가능한 옵션은 compress-force=zlib 및 compress-force=lzo입니다.

1 TB 보다 큰 파일 시스템을 사용할 것을 권장합니다. ***inode64*** 매개 변수는 XFS가 전체 파일 시스템에 걸쳐 inode와 데이터를 할당하도록 설정합니다. 이는 파일 시스템 시작 부분에 inode가 대량 할당되지 않게하고 파일 시스템의 마지막 부분에 데이터가 대량으로 할당되지 않게하여 대량 파일 시스템의 성능을 향상시킬 수 있습니다.

로그 버퍼 크기 및 수

로그 버퍼가 클 수록 로그에 모든 변경 사항을 기록할 때 소요되는 I/O 작업 수가 줄어듭니다. 대량의 로그 버퍼는 비휘발성 쓰기 캐시가 없는 I/O 집중 워크로드에서 시스템 성능을 향상시킬 수 있습니다.

로그 버퍼 크기는 ***logbsize*** 마운트 옵션으로 설정되며 로그 버퍼에 저장할 수 있는 최대 정보량을 정의합니다. 로그 스트라이프 단위가 설정되어 있지 않을 경우 버퍼 쓰기는 최대 값보다 작아지기 때문에 동기화 작업이 많은 워크로드에 대해 로그 버퍼 크기를 작게할 필요가 없습니다. 기본 로그 버퍼 크기는 32 KB입니다. 최대 크기는 256 KB이며 기타 지원되는 크기는 64 KB, 128 KB 또는 32 KB와 256 KB 사이에서 로그 스트라이프 단위의 2 제곱 배수입니다.

로그 버퍼 수는 ***logbufs*** 마운트 옵션에 의해 지정됩니다. 기본값은 (최대) 8 로그 버퍼이지만 최소 2 로그 버퍼를 지정할 수 있습니다. 추가 로그 버퍼에 메모리를 할당할 여유가 없는 메모리 바인딩 시스템을 제외하고 로그 버퍼 수를 줄일 필요가 없습니다. 로그 버퍼 수를 줄이면 로그 성능이 저하되는 경향이 있으며 특히 로그 I/O 대기 시간에 제약이 있는 작업에서 더욱 그러합니다.

변경 기록 지연

XFS에는 변경 사항을 로그에 기록하기 전 메모리에 변경 사항을 집계해 둘 수 있는 옵션이 있습니다. ***delaylog*** 매개 변수를 사용하여 자주 변경되는 메타 데이터를 변경 사항이 있을 때 마다 기록하지 않고 주기적으로 로그에 기록할 수 있습니다. 이러한 옵션을 사용하면 크래시에서 손실될 수 있는 동작 수가 증가하고 메타 테이터를 추적하기 위해 사용되는 메모리 양도 증가합니다. 하지만 예상 작업량에 따라 메타데이터 변경 속도와 확장성을 증가시킬 수 있고 **fsync**, **fdatasync**, **sync**를 사용하여 디스크에 기록되는 데이터와 메타데이터를 확인하는 경우 데이터 또는 메타테이터의 무결성을 감소시킬 수 없습니다.

5.3.7.2. ext4 튜닝

다음 부분에서는 ext4 파일 시스템 포맷이나 마운트 시 사용할 수 있는 일부 튜닝 매개 변수에 대해 설명합니다.

5.3.7.2.1. 포맷 옵션

Inode 테이블 초기화

대형 파일 시스템에서 파일 시스템에 있는 모든 inode를 초기화하는 데는 시간이 오래 걸릴 수 있습니다. 기본적으로 초기화 프로세스는 미룰 수 있습니다 (이는 inode 테이블 초기화 지연이 활성화되어 있는 경우에 가능합니다만 시스템에 ext4 드라이버가 없을 경우 inode 테이블 초기화 지연은 기본적으로 비활성화되어 있습니다. 이는 **lazy_itable_init**을 1로 설정하여 활성화할 수 있습니다.) 이러한 경우 커널 프로세스는 마운트된 후 파일 시스템을 계속 초기화합니다.

다음 부분에서는 포맷 시 사용가능한 일부 옵션에 대해서만 설명합니다. 자세한 포맷 관련 매개 변수는 **mkfs.ext4** man 페이지에서 참조하십시오:

```
$ man mkfs.ext4
```

5.3.7.2.2. 마운트 옵션

Inode 테이블 초기화 비율

inode 테이블 초기화 지연이 활성화되면 ***init_itable*** 매개 변수 값을 지정하여 초기화가 발생하는 비율을 제어할 수 있습니다. 백그라운드 초기화 실행에 걸리는 시간은 대략 이러한 매개 변수 값을 1로 나눈 값입니다. 기본값은 **10**입니다.

자동 파일 동기화

기존 파일 이름을 변경하거나 잘라내기 및 재작성 후 일부 애플리케이션에서 **fsync**가 제대로 작동하지 않는 경우가 있습니다. 기본적으로 ext4는 이러한 동작 후 파일 동기화가 자동으로 수행됩니다. 하지만 이러한 작업을 수행하는데는 시간이 걸릴 수 있습니다.

이러한 수준의 동기화 작업이 필요하지 않은 경우 마운트시 **noauto_da_alloc** 옵션을 지정하여 이러한 동작을 비활성화할 수 있습니다. **noauto_da_alloc** 옵션이 설정되면 애플리케이션은 데이터 지속성을 보장하기 위해 명시적으로 fsync를 사용해야 합니다.

저널 I/O 우선 순위

기본적으로 저널 I/O우 우선 순위는 일반적인 I/O 우선 순위 보다 약간 높은 **3**입니다. 마운트 시 ***journal_ioprio*** 매개 변수로 저널 I/O의 우선 순위를 제어할 수 있습니다. ***journal_ioprio*** 값으로 사용할 수 있는 범위는 **0**에서 **7**이며 **0**이 가장 우선 순위가 높은 I/O가 됩니다.

다음 부분에서는 마운트 시 사용할 수 있는 일부 옵션에 대해서만 설명합니다. 마운트 옵션에 대한 보다 자세한 내용은 **mount** man 페이지에서 참조하십시오:

```
$ man mount
```

5.3.7.3. BTRFS 튜닝

Red Hat Enterprise Linux 7.0에서 BTRFS는 기술 프리뷰로 제공됩니다. 항상 튜닝은 시스템의 현재 워크로드에 기반하여 시스템을 최적화하기 위해 수행됩니다. 다음 부분에서는 BTRFS를 최적화하는 방법만을 다루며 생성 및 마운트 옵션의 경우 https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Storage_Administration_Guide/ch-btrfs.html의 *Red Hat Enterprise Linux 7 Storage Administration Guide*에서 참조하십시오.

5.3.7.3.1. 데이터 압축

기본 압축 알고리즘은 zlib이지만 워크로드에 따라 이를 변경할 수 있습니다. 예를 들어 대량의 파일 I/O가 있는 단일 스레드가 있는 경우 lzo 알고리즘을 사용할 수 있습니다. 마운트시 옵션은 다음과 같습니다:

- compress=zlib - 더 나은 압축 비율입니다. 이전 커널에 대해 기본값이며 사용 안전합니다. (기본값)
- compress=lzo - 압축 속도는 빠르지만 zlib 만큼 압축하지 않습니다.

- compress=no - 압축을 비활성화합니다. (커널 3.6부터 사용 가능)
- compress-force=method - 비디오나 디스크 이미지와 같이 압축이 잘 되지 않는 파일의 경우 이러한 압축을 사용합니다. 사용 가능한 옵션은 compress-force=zlib 및 compress-force=lzo입니다.

마운트 옵션 추가 후 생성 또는 변경된 파일만 압축합니다. 기존 파일을 압축하려면 다음 명령을 사용합니다:

```
$ btrfs filesystem defragment -calg
```

alg는 **zlib** 또는 **lzo**가 됩니다

lzo를 사용하여 파일을 다시 압축하려면 다음 명령을 사용합니다:

```
$ btrfs filesystem defragment -r -v -clzo /
```

5.3.7.4. GFS2 튜닝

다음 부분에서는 GFS2 파일 시스템 포맷이나 마운트 시 사용할 수 있는 일부 튜닝 매개 변수에 대해 설명합니다.

디렉토리 간격

GFS2 마운트 지점의 최상위 디렉토리에 생성된 모든 디렉토리는 자동으로 간격을 두어 단편화를 줄이고 디렉토리에 쓰기 속도를 가속화합니다. 최상위 디렉토리와 같이 다른 디렉토리에 간격을 두려면 다음과 같이 ***T*** 속성과 함께 디렉토리를 표시합니다. 여기서 *dirname*은 간격을 두려는 디렉토리 경로로 변경합니다.

```
# chattr +T dirname
```

chattr는 *e2fsprogs* 패키지 일부로 제공됩니다.

경합 감소

GFS2는 클러스터 노드 간 통신을 필요로 하는 글로벌 잠금 메카니즘을 사용합니다. 여러 노드에서 파일 및 디렉토리 경합이 발생하면 성능이 저하됩니다. 여러 노드에서 공유되는 파일 시스템 공간을 최소화하여 캐시 무효화 위험을 감소시킬 수 있습니다.

6장. 네트워킹

네트워킹 서브 시스템은 여러 다른 부분의 민감한 연결로 형성되어 있습니다. 따라서 Red Hat Enterprise Linux 7 네트워킹은 대부분의 작업에 최적의 성능을 제공하고 이러한 성능을 자동으로 최적화하도록 고안되어 있습니다. 이로 인해 수동으로 네트워크 성능을 튜닝할 필요가 없습니다. 다음 부분에서는 네트워킹 시스템에 실질적으로 적용 가능한 최적화에 대해 설명합니다.

일부 경우 네트워크 성능 문제는 하드웨어 결함이나 인프라 장애로 인한 것이 원인이 될 수 있습니다. 이러한 문제를 해결하는 방법은 이 문서의 범위에서 벗어나기 때문에 다루지 않습니다.

6.1. 고려 사항

튜닝 관련 적절한 설정을 하려면 Red Hat Enterprise Linux의 패킷 수신에 대해 충분히 숙지하고 있어야 합니다. 다음 부분에서는 네트워크 패킷 수신 및 처리, 병목 현상이 발생할 수 있는 부분에 대해 설명합니다.

Red Hat Enterprise Linux 시스템에 전송된 패킷은 NIC (network interface card)에서 수신되어 내부 하드웨어 버퍼 또는 링 버퍼 중 하나에 배치됩니다. 그 후 NIC는 하드웨어 인터럽트 요청을 전송하고 이러한 인터럽트 요청을 처리하기 위해 소프트웨어 인터럽트 작동을 생성할 것을 요청합니다.

소프트웨어 인터럽트 동작의 일부로 패킷은 버퍼에서 네트워크 스택에 전달됩니다. 패킷 및 네트워크 설정에 따라 패킷은 애플리케이션 소켓 전송 큐에 전달, 전송, 삭제된 후 네트워크 스택에서 제거됩니다. 이러한 절차는 NIC 하드웨어 버퍼에 패킷이 남아 있지 않거나 특정 패킷 수 (**/proc/sys/net/core/dev_weight**에 지정됨)가 전송될 때 까지 계속됩니다.

6.1.1. 튜닝 전

네트워크 성능 문제는 주로 하드웨어 기능 장애나 인프라 장애로 인한 경우가 많습니다. Red Hat은 네트워크 스택을 튜닝하기 전 하드웨어 및 인프라가 제대로 작동하는지를 확인할 것을 권장합니다.

6.1.2. 패킷 수신 병목 현상

네트워크 스택은 대부분 자체 최적화되지만 네트워크 패킷 프로세스 동안 병목 현상 및 성능 저하를 일으킬 수 있는 몇 가지 부분이 있습니다.

NIC 하드웨어 버퍼 또는 링 버퍼

대량의 패킷이 드롭되면 하드웨어 버퍼가 병목되는 현상이 발생할 수 있습니다. 드롭된 패킷을 모니터링하는 방법에 대한 자세한 내용은 6.2.4절. “ethtool”에서 참조하십시오.

하드웨어 또는 소프트웨어 인터럽트 큐

인터럽트로 인해 대기 시간이 늘어나고 프로세서 경합이 발생할 수 있습니다. 프로세서에 의해 인터럽트가 처리되는 방법에 대한 자세한 내용은 3.1.3절. “IRQ (Interrupt Request) 처리”에서 참조하십시오. 시스템에서 인터럽트 처리를 모니터링하는 방법에 대한 자세한 내용은 3.2.3절. “/proc/interrupts”에서 참조하십시오. 인터럽트 처리에 영향을 미치는 설정 옵션에 대한 자세한 내용은 3.3.7절. “인터럽트 친화도 설정 ”에서 참조하십시오.

애플리케이션의 소켓 수신 큐

애플리케이션의 수신 큐에서의 병목 현상은 요청 애플리케이션에 대해 대량의 소켓이 복사되지 않은 경우 또는 **/proc/net/snmp**에 UDP 입력 오류 (**InErrors**)가 증가한 경우 나타납니다. 이러한 오류를 모니터링하는 방법에 대한 자세한 내용은 6.2.1절. “ss” 및 6.2.5절. “/proc/net/snmp”에서 참조하십시오.

6.2. 성능 관련 문제 모니터링 및 진단

Red Hat Enterprise Linux 7에는 시스템 성능을 모니터링하고 네트워크를 구성하는 서브 시스템과 관련된 성능 문제를 진단하는데 유용한 여러 도구가 있습니다. 다음 부분에서는 네트워크 관련 성능 문제를 모니터링하고 진단하기 위해 사용 가능한 도구 및 사용 방법에 대해 설명합니다.

6.2.1. ss

ss는 소켓에 대한 통계 정보를 출력하는 명령행 유틸리티로 이를 통해 관리자는 장치 성능을 평가할 수 있습니다. 기본적으로 **ss**는 연결된 오픈 비수신 TCP 소켓을 나열하지만 특정 소켓에 대한 통계를 필터링하는데 관리자에게 유용한 여러 옵션도 제공합니다.

Red Hat은 Red Hat Enterprise Linux 7에서 **netstat** 보다 **ss**를 사용할 것을 권장합니다.

ss는 *iproute* 패키지에서 제공됩니다. 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man ss
```

6.2.2. ip

ip 유틸리티를 사용하여 관리자는 경로, 장치, 라우팅 정책, 터널을 관리 및 모니터링할 수 있습니다. **`ip monitor`** 명령을 사용하여 주소, 경로, 장치 상태 등을 지속적으로 모니터링할 수 있습니다.

ip는 *iproute* 패키지에서 제공됩니다. ip 사용에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man ip
```

6.2.3. dropwatch

Dropwatch는 커널에서 드롭된 패킷을 모니터링 및 기록할 수 있는 대화형 도구입니다.

보다 자세한 내용은 **dropwatch** man 페이지에서 참조하십시오:

```
$ man dropwatch
```

6.2.4. ethtool

ethtool 유틸리티를 통해 관리자는 네트워크 인터페이스 카드 설정을 확인 및 편집할 수 있습니다. 이는 특정 장치에서 드롭된 패킷 수와 같이 특정 장치의 통계를 확인하는데 유용합니다.

`ethtool -S` 및 모니터할 장치 이름을 사용하여 특정 장치의 카운터 상태를 확인할 수 있습니다.

```
$ ethtool -S devname
```

보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man ethtool
```

6.2.5. /proc/net/snmp

/proc/net/snmp 파일은 IP, ICMP, TCP, UDP 모니터링 및 관리 용으로 snmp 에이전트에 의해 사용되는 데이터를 표시합니다. 정기적으로 이러한 파일을 검사하면 관리자는 비정상 값이 표시되는지 확인할 수 있으므로 잠재적 성능 관련 문제를 확인할 수 있습니다. 예를 들어 **/proc/net/snmp** 에서 UDP 입력 오류 (**InErrors**)가 증가하는 경우 소켓 수신 큐에 병목 현상을 표시할 수 있습니다.

6.2.6. SystemTap을 사용한 네트워크 모니터링

*Red Hat Enterprise Linux 7 SystemTap Beginner's Guide*에는 네트워크 성능을 모니터링 및 프로파일링하는데 유용한 여러 스크립트 예제가 들어 있습니다.

다음의 **SystemTap** 스크립트 예제는 네트워킹과 관련되어 있으며 네트워크 성능 문제를 진단하는데 유용합니다. 이는 기본적으로 **/usr/share/doc/systemtap-client/examples/network** 디렉토리에 설치됩니다.

nettop.stp

5 초 마다 프로세스 목록 (프로세스 ID 및 명령)을 송수신된 패킷 수와 송수신 간격에서 프로세스에 의해 송수신된 데이터 양과 함께 출력합니다.

socket-trace.stp

Linux 커널의 **net/socket.c** 파일에서 각 함수를 계측하고 추적 데이터를 출력합니다.

dropwatch.stp

5 초 마다 커널에 해제된 소켓 버퍼 수를 출력합니다. **--all-modules** 옵션을 사용하여 기호 이름을 표시합니다.

latencytap.stp 스크립트는 하나 이상의 프로세스에 있는 다른 종류의 대기 시간이 미치는 영향을 기록합니다. 이는 30 초 마다 대기 유형 목록을 총 프로세스 대기 시간에 따라 내림 차순으로 분류하여 출력합니다. 이는 스토리지 및 네트워크 지연 원인을 파악하는데 유용합니다. Red Hat은 대기 시간 이벤트 맵핑을 보다 효과적으로 활성화하기 위해 이 스크립트와 함께 **--all-modules** 옵션을 사용할 것을 권장합니다. 기본적으로 이 스크립트는 **/usr/share/doc/systemtap-client-*version*/examples/profiling** 디렉토리에 설치됩니다.

보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에 있는 *Red Hat Enterprise Linux 7 SystemTap Beginner's Guide*에서 참조하십시오.

6.3. 설정 도구

Red Hat Enterprise Linux에는 시스템을 설정할 때 관리자에게 유용한 여러 도구가 있습니다. 다음 부분에서는 이러한 도구에 대해 설명하고 Red Hat Enterprise Linux 7에서 네트워크 관련 성능 문제를 해결하는 방법에 대해 설명합니다.

일부 네트워크 성능 문제 하드웨어 기능 장애나 인프라 장애로 인한 경우가 많다는 것을 염두해 두셔야 합니다. Red Hat은 네트워크 스택을 튜닝하기 위해 이러한 도구를 사용하기 전 하드웨어 및 인프라가 제대로 작동하는지를 확인하는 것이 좋습니다.

이에 더하여 일부 네트워크 성능 관련 문제는 네트워크 서브 시스템을 다시 설정하는것 보다 애플리케이션을 변경하는 것이 문제 해결을 위해 더 나은 방법이 될 수 있습니다. 애플리케이션 영역에 데이터 큐가 있어도 데이터를 유연하게 저장할 수 있고 필요에 따라 메모리를 스왑할 수 있기 때문에 일반적으로 애플리케이션이 자주 posix 호출을 실행하도록 설정하는 것도 좋은 방법일 수 있습니다.

6.3.1. 네트워크 성능을 위한 Tuned-adm 프로파일

tuned-adm은 다수의 특정 사용 경우에서 성능을 개선하기 위한 여러 다른 프로파일을 제공합니다. 다음 프로파일은 네트워크 성능을 개선하는데 유용합니다.

- latency-performance
- network-latency
- network-throughput

이러한 프로파일에 대한 보다 자세한 내용은 A.6절. “tuned-adm”에서 참조하십시오.

6.3.2. 하드웨어 버퍼 설정

다수의 패킷이 하드웨어 버퍼에 의해 드롭되는 경우 몇 가지 사용 가능한 솔루션이 있습니다.

입력 트래픽 속도 감소

들어오는 트래픽을 필터링하면 참여한 멀티 캐스트 그룹 수를 감소시키거나 큐가 채워지는 비율을 줄이기 위해 브로드캐스트 트래픽 수를 감소시킵니다. 들어오는 트래픽을 필터링하는 방법에 대한 자세한 내용은 *Red Hat Enterprise Linux 7 Security Guide*에서 참조하십시오. 멀티캐스트 그룹에 대한 자세한 내용은 Red Hat Enterprise Linux 7 클러스터링 문서에서 참조하십시오. 브로드캐스트 트래픽에 대한 자세한 내용은 *Red Hat Enterprise Linux 7 System Administrator's Reference Guide*에서 참조하거나 설정하고자 하는 장치와 관련된 문서에서 참조하십시오. 모든 Red Hat Enterprise Linux 7 문서는 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에서 확인하실 수 있습니다.

하드웨어 버퍼 큐 크기 변경

큐 크기를 증가시키기 위해 드롭되는 패킷 수를 줄여 쉽게 오버플로우되지 않게 합니다. ethtool 명령을 사용하여 네트워크 장치의 rx/tx 매개 변수를 수정할 수 있습니다.

```
# ethtool --set-ring devname value
```

큐 배출 비율 변경

장치 가중치는 한 번에 (한 번의 스케줄된 프로세서 에 액세스) 장치가 수신할 수 있는 패킷 수를 말합니다. ***dev_weight*** 매개 변수에 의해 관리되는 장치 가중치를 증가시켜 큐 배출 비율을 증가시킬 수 있습니다. 이러한 매개 변수는 **/proc/sys/net/core/dev_weight** 파일 내용을 변경하여 일시적으로 변경하거나 *procps-ng* 패키지에서 제공하는 **sysctl**를 사용하여 영구적으로 변경할 수 있습니다.

일반적으로 큐 배출 비율을 변경하는 것이 네트워크 성능 저하를 완화하는 가장 간단한 방법입니다. 하지만 한 번에 장치가 수신할 수 있는 패킷 수를 증가시켜 추가 프로세서 시간을 사용하게 되므로 그동안 다른 프로세서 스케줄에 예약될 수 없기 때문에 다른 성능 관련 문제가 발생할 수 있습니다.

6.3.3. 인터럽트 큐 설정

분석 결과에 높은 대기 시간이 나타날 경우 시스템은 인터럽트 기반 패킷 수신 대신 폴링 기반 수신을 통해 개선될 수 있습니다.

6.3.3.1. 비지 폴링 설정

비지 폴링을 통해 소켓 레이어 코드가 네트워크 장치의 수신 큐를 폴링시켜 네트워크 인터럽트를 비활성화시키므로 네트워크 수신 경로의 대기 시간을 줄일 수 있습니다. 이는 인터럽트 및 결과 컨텍스트 스위치로 인한 지연을 감소시킬 수 있지만 CPU 사용률은 증가합니다. 비지 폴링은 CPU의 수면 상태가 되는 것을 방지하므로 추가 전력 소비가 발생할 수 있습니다.

기본적으로 비지 폴링은 비활성화되어 있으며 특정 소켓에서 폴링을 활성화하려면 다음을 실행합니다.

- ***sysctl.net.core.busy_poll***을 **0** 이외의 값으로 설정합니다. 이 매개 변수는 소켓 폴링 및 선택 용 장치 큐에서 패킷을 기다리는 시간을 마이크로 초 단위로 관리합니다. Red Hat은 **50**으로 설정할 것을 권장합니다.
- 소켓에 **SO_BUSY_POLL** 소켓 옵션을 추가합니다.

글로벌 비지 폴링을 사용하는 경우 ***sysctl.net.core.busy_read***를 **0** 이외의 값으로 설정해야 합니다. 이 매개 변수는 소켓의 읽기 용 장치 큐에서 패킷을 기다리는 시간을 마이크로 초 단위로 관리합니다. 또한 **SO_BUSY_POLL** 옵션의 기본값이 설정됩니다. Red Hat은 소켓 수가 적은 경우 **50**으로 소켓 수가 많은 경우 **100**으로 설정할 것을 권장합니다. 소켓 수가 너무 많은 경우 (수백 이상) 대신 **epoll**을 사용합니다.

비지 폴링 동작은 다음 드라이버에 의해 지원됩니다. 이러한 드라이버는 Red Hat Enterprise Linux 7에서 지원됩니다.

- bnx2x
- be2net
- ixgbe
- mlx4
- myri10ge

Red Hat Enterprise Linux 7.1에서 다음 명령을 실행하여 특정 장치가 비지 폴링을 지원하는지에 대한 여부를 확인할 수 있습니다.

```
# ethtool -k device | grep "busy-poll"
```

busy-poll: on [fixed]을 반환할 경우 장치에서 비지 폴링을 사용할 수 있습니다.

6.3.4. 소켓 수신 큐 설정

분석 결과에서 소켓 큐 배출 비율이 너무 낮아 패킷이 드롭되는 현상이 나타날 경우 이러한 성능 문제를 완화시킬 수 있는 몇 가지 방법이 있습니다.

들어오는 트래픽 속도 감소

패킷이 큐에 도달하기전 필터링 또는 드롭되거나 장치 가중치를 낮춰 큐가 채워지는 비율을 감소시킵니다.

애플리케이션의 소켓 큐 깊이 증가

제한된 트래픽을 연속적으로 수신하는 소켓 큐의 경우 연속적 트랙픽 크기에 맞게 소켓큐의 깊이를 증가시키면 패킷이 드롭되지 않게 할 수 있습니다.

6.3.4.1. 들어오는 트래픽 속도 감소

들어오는 트래픽을 필터링하거나 네트워크 인터페이스 카드의 장치 가중치를 낮게하여 들어 오는 트랙픽 속도를 늦출 수 있습니다. 들어오는 트래픽을 필터링하는 방법에 대한 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 *Red Hat Enterprise Linux 7 Security Guide*에서 참조하십시오.

장치 가중치는 한 번에 (한 번의 스케줄된 프로세서에 액세스) 장치가 수신할 수 있는 패킷 수를 말합니다. 장치 가중치는 ***dev_weight*** 매개 변수에 의해 관리됩니다. 이러한 매개 변수는 **/proc/sys/net/core/dev_weight** 파일 내용을 변경하여 일시적으로 변경하거나 *procps-ng* 패키지에서 제공하는 **sysctl**를 사용하여 영구적으로 변경할 수 있습니다.

6.3.4.2. 큐 깊이 증가

애플리케이션 소켓 큐 깊이를 증가시키는 것이 소켓 큐 배출 비율을 개선하기 위한 가장 쉬운 방법이지만 이는 장기적 솔루션이 될 수 없습니다.

큐의 깊이를 증가시키려면 다음과 같이 변경하여 소켓 수신 버퍼 크기를 증가시킵니다:

/proc/sys/net/core/rmem_default 값을 증가

이 매개 변수는 소켓에서 사용되는 기본 수신 버퍼 크기를 관리합니다. 이 값은 **/proc/sys/net/core/rmem_max** 값보다 작아야 합니다.

setsockopt를 사용하여 SO_RCVBUF 값을 크게 설정

이 매개 변수는 소켓 수신 버퍼의 최대 크기를 바이트 단위로 관리합니다. **getsockopt** 시스템 호출을 사용하여 현재 버퍼 값을 확인합니다. 이 매개 변수에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man 7 socket
```

6.3.5. RSS (Receive-Side Scaling) 설정

멀티 큐 수신이라고 하는 RSS (Receive-Side Scaling)은 여러 하드웨어 기반 수신 큐에 걸쳐 네트워크 수신 프로세스를 분산시켜 인바운드 네트워크 트래픽을 여러 CPU에서 처리할 수 있습니다. RSS는 단일 CPU 오버로드에 의한 수신 인터럽트 처리에 있어서 병목 현상을 완화하고 네트워크 대기 시간을 감소시키는데 사용될 수 있습니다.

네트워크 인터페이스 카드가 RSS를 지원하는지 확인하려면 여러 인터럽트 요청 큐가 **/proc/interrupts**에 있는 인터페이스와 관련되어 있는지 확인합니다. 예를 들어 **p1p1** 인터페이스를 확인하려면 다음을 수행합니다:

```
# 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
```

위의 출력에서는 NIC 드라이버가 **p1p1** 인터페이스 (**p1p1-5**를 통한 **p1p1-0**)에 대해 6 수신 큐가 생성되었음을 보여주고 있습니다. 또한 각 큐에서 얼마나 많은 인터럽트가 프로세스되었는지와 어떤 CPU가 인터럽트를 처리했는지도 보여주고 있습니다. 이러한 경우 이러한 특정 NIC 드라이버가 CPU 당 하나의 큐를 생성하기 때문에 기본적으로 6 큐가 생성되어 있고 이러한 시스템에는 6 CPU가 있게 됩니다. 이는 NIC 드라이버에서 가장

일반적인 패턴입니다.

다른 방법으로 네트워크 드라이버가 로딩된 후 **ls -1 /sys/devices/*/*/*device_pci_address*/msi_irqs** 출력 결과를 확인할 수 있습니다. 예를 들어 PCI 주소가 **0000:01:00.0**인 장치를 확인하려면 다음 명령을 사용하여 해당 장치의 인터럽트 요청 큐를 나열할 수 있습니다:

```
# ls -1 /sys/devices/*/*/0000:01:00.0/msi_irqs
101
102
103
104
105
106
107
108
109
```

RSS는 기본적으로 활성화되어 있습니다. RSS 큐 수 (또는 네트워크 활동을 처리하는 CPU 수)는 해당 네트워크 장치 드라이버에서 설정됩니다. **bnx2x** 드라이버의 경우 ***num_queues***에서 설정됩니다. **sfc** 드라이버의 경우 ***rss_cpus*** 매개 변수에서 설정됩니다. 이와 관계 없이 일반적으로 **/sys/class/net/*device*/queues/*rx-queue*/**에 설정됩니다. 여기서 *device*는 네트워크 장치 이름 (예: **eth1**)이고 *rx-queue*는 해당 장치 큐 이름입니다.

RSS 설정 시 Red Hat은 실제 CPU 코어 당 하나로 큐의 수를 제한할 것을 권장하고 있습니다. 분석 도구에서 하이퍼 스레드가 별도의 코어로 표시될 수 있지만 하이퍼 스레드와 같은 논리 코어를 포함한 모든 코어의 큐를 설정하는것은 네트워크 성능 향상에 도움이 되는지는 입증되어 있지 않습니다.

RSS가 활성화되어 있으면 이는 각 CPU가 큐에 등록된 처리 양에 따라 CPU 사이에서 네트워크 처리를 균등하게 분산합니다. 하지만 **ethtool *--show-rxfh-indir*** 및 ***--set-rxfh-indir*** 매개 변수를 사용하여 네트워크 활동 분산 방법을 수정하거나 특정 네트워크 활동을 다른 활동 보다 더 중요한 것으로 가중치 부여할 수 있습니다.

irqbalance 데몬은 RSS와 함께 사용하여 노드간 메모리 전송 및 캐시 라인 반송 가능성을 감소시킬 수 있습니다. 이는 네트워크 패킷 처리 지연 시간을 줄일 수 있습니다.

6.3.6. RPS (Receive Packet Steering) 설정

RPS (Receive Packet Steering)은 특정 CPU 처리에 직접 패킷을 사용한다는 점에서 RSS와 유사합니다. 하지만 RPS는 소프트웨어 레벨에서 구현되며 단일 네트워크 인터페이스 카드의 하드웨어 큐의 네트워크 트래픽에서 병목현상이 발생하지 않도록 합니다.

RPS는 하드웨어 기반 RSS를 통해 다음과 같은 장점을 갖습니다:

- RPS는 네트워크 인터페이스 카드와 함께 사용할 수 있습니다.
- 새로운 프로토콜을 처리하기 위해 RPS에 소프트웨어 필터를 쉽게 추가할 수 있습니다.
- RPS는 네트워크 장치의 하드웨어 인터럽트 비율을 증가시킬 수 없습니다. 하지만 내부 프로세서 인터럽트가 발생합니다.

RPS는 **/sys/class/net/*device*/queues/*rx-queue*/rps_cpus** 파일에서 네트워크 장치 및 수신 큐 마다 설정됩니다. 여기서 *device*는 네트워크 장치 (예: **eth0**) 이름이며 *rx-queue*는 해당 수신 큐 (예: **rx-0**) 이름입니다.

rps_cpus 파일의 기본값은 **0**입니다. 이는 RPS를 비활성화하여 네트워크 인터럽트를 처리하는 CPU가 패킷을 처리합니다.

RPS를 활성화하려면 특정 네트워크 장치 및 수신 큐에서 패킷을 처리하는 CPU에서 해당 **rps_cpus** 파일을 설정합니다.

rps_cpus 파일은 쉼표로 구분된 CPU 비트맵을 사용합니다. 따라서 CPU를 통해 인터페이스에서 수신 큐의 인터럽트를 처리하려면 비트맵을 1로 설정합니다. 예를 들어 CPU 0, 1, 2, 3의 인터럽트를 처리하려면 **rps_cpus**를 **00001111** (1+2+4+8), 또는 **f** (15의 16 진수 값)로/으로 설정합니다.

단일 전송 큐가 있는 네트워크 장치의 경우 동일한 메모리 도메인에 있는 CPU를 사용하도록 RPS를 설정하여 최상의 성능이 나타나게 할 수 있습니다. 이는 비 NUMA 시스템에서 모든 사용 가능한 CPU를 사용할 수 있음을 의미합니다. 네트워크 인터럽트 비율이 너무 높을 경우 네트워크 인터럽트를 처리하는 CPU를 제외시켜 성능을 개선할 수 있습니다.

여러 큐가 있는 네트워크 장치의 경우 일반적으로 RPS 및 RSS 모두를 설정할 경우 장점이 없습니다. RSS는 기본적으로각 수신 큐에 CPU를 맵핑하도록 설정되기 때문입니다. 하지만 CPU 보다 하드웨어 큐가 적고 동일한 메모리 도메인에서 CPU를 사용하도록 RPS가 설정되어 있는 경우 RPS는 장점이 될 수 있습니다.

6.3.7. RFS (Receive Flow Steering) 설정

RFS (Receive Flow Steering)는 RPS 동작을 확장한 것으로 CPU 캐시 적중률을 높이고 네트워크 대기 시간을 줄이기 위한 것입니다. RPS는 큐 길이에 따라 패킷을 전송하지만 RFS는 최적의 CPU를 계산하기 위해 RPS 백엔드를 사용하고 패킷을 소비하는 애플리케이션의 위치에 따라 패킷을 전송합니다. 이로 인해 CPU 캐시 효율성이 증가합니다.

RFS는 기본적으로 비활성화되어 있습니다. RFS를 활성화하려면 다음의 두 파일을 편집해야 합니다:

/proc/sys/net/core/rps_sock_flow_entries

이 파일의 값을 동시에 활성화할 최대 연결 예상 수로 설정합니다. 일반적인 서버 부하의 경우 **32768**로 설정하는 것이 좋습니다. 입력된 모든 값은 가장 가까운 2의 거듭 제곱으로 반올림됩니다.

/sys/class/net/*device*/queues/*rx-queue*/rps_flow_cnt

*device*는 설정하고자 하는 네트워크 장치 이름 (예: **eth0**)으로 *rx-queue*는 설정하고자 하는 수신 큐 (예: **rx-0**)로 변경합니다.

이 파일 값을 **N**으로 나눈 **rps_sock_flow_entries** 값으로 설정합니다. 여기서 **N**은 장치의 수신 큐 수입니다. 예를 들어 **rps_flow_entries**가 **32768**로 설정되어 있고 16 개의 설정된 수신 큐가 있을 경우 **rps_flow_cnt**는 **2048**로 설정해야 합니다. 단일 큐 장치의 경우 **rps_flow_cnt** 값은 **rps_sock_flow_entries** 값과 동일합니다.

단일 전송자로 부터 수신된 데이터는 여러 CPU에 전송되지 않습니다. 단일 전송자로 부터 수신된 데이터 양이 단일 CPU가 처리할 수 있는 양 보다 많을 경우 프레임 크기를 크기 설정하여 인터럽트 수를 줄이고 결국 CPU 프로셋스 양을 줄일 수 있습니다. 다른 방법으로 NIC 오프로드 옵션 이나 빠른 CPU 사용을 고려해 볼 수 있습니다.

RFS와 함께 **numactl** 또는 **taskset**을 사용하여 애플리케이션을 특정 코어, 소켓 또는 NUMA 노드에 고정할 수 있습니다. 이를 통해 패킷이 잘못 처리되지 않게 할 수 있습니다.

6.3.8. 가속 RFS 설정

가속 RFS는 하드웨어 지원을 추가하여 RFS 속도를 높일 수 있습니다. RFS 처럼 패킷은 패킷을 소비하는 애플리케이션 위치에 따라 전송됩니다. 하지만 기존 RFS와는 다르게 패킷은 데이터를 소비하는 스레드 로컬인 CPU (애플리케이션을 실행하는 CPU 또는 캐시 계층에 있는 CPU의 CPU 로컬 중 하나)에 직접 전송됩니다.

가속 RFS는 다음과 같은 조건이 충족되는 경우에만 사용할 수 있습니다:

- 가속 RFS는 네트워크 인터페이스 카드에 의해 지원되어야 합니다. 가속 RFS는 **`ndo_rx_flow_steer()`** **`netdevice`** 함수를 내보내는 카드에 의해 지원됩니다.
- **`ntuple`** 필터링이 활성화되어 있어야 합니다.

이러한 조건이 충족되면 큐 맵핑에서의 CPU는 기존 RFS 설정에 따라 자동으로 추론됩니다. 즉 큐 맵핑에서의 CPU는 각 수신 큐의 드라이버에 의해 설정되는 IRQ 친화도에 따라 추론됩니다. 기존 RFS 설정에 대한 자세한 내용은 6.3.7절. “RFS (Receive Flow Steering) 설정 ”에서 참조하십시오.

Red Hat은 RFS 사용이 적합한 상황에서 네트워크 인터페이스 카드가 하드웨어 가속을 지원하는 경우 가속 RFS 사용을 권장하고 있습니다.

부록 A. 도구 참조

다음 부분에서는 성능 조정에 사용할 수 있는 Red Hat Enterprise Linux 7에서의 다양한 도구에 대해 간략하게 설명합니다. 툴에 대한 자세한 내용 및 최신 정보는 해당 도구의 man 페이지에서 참조하십시오.

A.1. irqbalance

irqbalance는 명령행 도구로 프로세서간 하드웨어 인터럽트를 분산시켜 시스템 성능을 개선합니다. 이는 기본적으로 데몬으로 실행되지만 **--oneshot** 옵션을 사용하여 한 번만 실행할 수 있습니다.

다음과 같은 매개 변수는 성능 개선에 유용합니다.

--powerthresh

CPU가 절전 모드로 변경되기 전 유휴 상태가 될 수 있는 CPU 수를 설정합니다. 임계값을 초과하는 CPU 수가 평균 softirq 워크로드 보다 1 표준 편차 아래에 있고 평균 보다 높은 1 표준 편차를 초과하는 CPU가 없으며 하나 이상의 irq가 할당된 경우 CPU는 절전 모드에 들어갑니다. 절전 모드에서 CPU는 irq 밸런싱의 일부가 아니므로 불필요하게 재개되지 않습니다.

--hintpolicy

irq 커널 친화도 힌트의 처리 방법을 지정합니다. 사용 가능한 값은 **exact** (irq 친화도 힌트를 항상 적용), **subset** (irq가 균형 조정되지만 할당 개체는 친화도 힌트의 일부가 됨), **ignore** (irq 친화도 힌트를 완전히 무시)입니다.

--policyscript

각 인터럽트 요청을 실행하는 스크립트의 위치를 정의합니다. 인수로 전달된 장치 경로 및 irq 번호, **irqbalance**에서 예상되는 제로 종료 코드를 사용합니다. 정의된 스크립트는 전달된 irq 관리에서 **irqbalance**를 가이드하기 위해 제로 또는 그 이상 키 값 쌍을 지정할 수 있습니다.

유효한 키 값 쌍으로 인식되는 것은 다음과 같습니다.

ban

유효한 값은 **true** (전달된 irq를 밸런싱에서 제외) 또는 **false** (이 irq에서 밸런싱 실행)입니다.

balance_level

사용자는 전달된 irq 균형 수준을 덮어쓰기할 수 있습니다. 기본적으로 균형 수준은 irq를 소유하는 장치의 PCI 장치 클래스를 기반으로 합니다. 사용 가능한 값은 **none**, **package**, **cache**, **core**입니다.

numa_node

전달된 irq는 로컬로 간주되는 NUMA 노드를 사용자가 덮어쓰기할 수 있습니다. 로컬 노드에 대한 정보가 ACPI에 지정되어 있지 않을 경우 장치는 모든 노드에서 등거리로 간주됩니다. 사용 가능한 값은 특정 NUMA 노드를 식별하는 정수 (0에서 시작) 및 irq가 모든 노드에서 등거리로 간주되도록 지정하는 **-1**입니다.

--banirq

지정된 인터럽트 요청 번호가 있는 인터럽트는 인터럽트 금지 목록에 추가됩니다.

IRQBALANCE_BANNED_CPUS 환경 변수를 사용하여 **irqbalance**에 의해 무시되는 CPU 마스크를 지정할 수 있습니다.

보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man irqbalance
```

A.2. Tuna

Tuna를 통해 프로세스 및 스케줄링 친화도를 제어할 수 있습니다. 다음 부분에서는 명령행 인터페이스에 대해 설명하지만 동일한 기능을 갖춘 그래픽 인터페이스도 사용 가능합니다. 명령행에서 **tuna**를 실행하여 그래픽 유틸리티를 시작합니다.

Tuna는 순서대로 처리되는 여러 명령행 매개 변수를 사용합니다. 다음 명령은 4 개의 소켓 시스템에 걸쳐 로드를 시스템에 분산하고 있습니다.

```
tuna --socket 0 --isolate \
   --thread my_real_time_app --move \
   --irq serial --socket 1 --move \
   --irq eth* --socket 2 --spread \
   --show_threads --show_irqs
```

--gui

그래픽 사용자 인터페이스를 시작합니다.

--cpus

Tuna에서 관리하는 CPU 목록을 콤마로 구분하여 가져옵니다. 새로운 목록이 지정될 때 까지 이 목록을 사용할 수 있습니다.

--config_file_apply

시스템에 적용할 프로파일 이름을 가져옵니다.

--config_file_list

미리 로드한 프로파일을 나열합니다.

--cgroup

--show_threads와 함께 사용됩니다. 제어 그룹이 활성화되어 있는 경우 ***--show_threads***가 속한 프로세스를 표시하는 제어 그룹 유형을 표시합니다. -P가 필요합니다.

--affect_children

지정되어 있을 경우 **Tuna**는 자식 스레드와 부모 스레드 모두에 영향을 미칩니다.

--filter

--gui에서 선택된 CPU 표시를 비활성화합니다. -c가 필요합니다.

--isolate

콤마로 구분된 CPU 목록을 가져옵니다. **Tuna**는 지정된 CPU에서 모든 스레드를 마이그레이션합니다. -c 또는 -s가 필요합니다.

--include

콤마로 구분된 CPU 목록을 가져옵니다. Tuna는 모든 스레드가 지정된 CPU에서 실행되도록 합니다. -c 또는 -s가 필요합니다.

--no_kthreads

이 매개 변수가 지정되어 있을 경우 Tuna는 커널 스레드에 영향을 미치지 않습니다.

--move

선택한 엔티티를 CPU 목록으로 이동합니다. -c 또는 -s가 필요합니다.

--priority

스케줄러 정책 및 스레드 우선 순위를 지정합니다. 사용 가능한 스케줄러 정책은 **OTHER**, **FIFO**, **RR**, **BATCH**, **IDLE**입니다.

정책이 **FIFO** 또는 **RR**인 경우 우선 순위 값은 1 (최저)부터 99 (최고) 사이의 정수입니다. 기본값은 **1**입니다. 예를 들어, **tuna --threads 7861 --priority=RR:40**은 **RR** (라운드 로빈) 정책을 설정하고 스레드 **7861**에 대해 우선 순위를 **40**으로 설정합니다.

정책이 **OTHER**, **BATCH**, **IDLE**일 경우 사용 가능한 우선 순위 값은 **0**이며 이는 기본값이 됩니다. -t가 필요합니다.

--show_threads

스레드 목록을 표시합니다.

--show_irqs

IRQ 목록을 표시합니다.

--irqs

Tuna에 영향을 미치는 콤마로 구분된 IRQ 목록을 가져옵니다. **+**를 사용하여 목록에 IRQ를 추가할 수 있으며 **-**를 사용하여 목록에서 삭제할 수 있습니다.

--save

지정된 파일에 커널 스레드 일정을 저장합니다.

--sockets

Tuna에서 관리하는 콤마로 구분된 CPU 소켓 목록을 가져옵니다. 이 옵션은 단일 프로세서 캐시를 공유하는 코어와 동일한 물리적 칩에 있는 코어와 같은 시스템 토폴로지 기능을 고려합니다.

--threads

Tuna에서 관리하는 콤마로 구분된 스레드 목록을 가져옵니다. 새로운 목록을 지정할 때 까지 이 목록을 사용할 수 있습니다. **+**를 사용하여 목록에 스레드를 추가할 수 있으며 **-**를 사용하여 목록에서 삭제할 수 있습니다.

--no_uthreads

동작이 사용자 스레드에 영향을 미치지 않도록 합니다.

--what_is

선택한 엔티티에 대한 상세한 정보를 확인하려면 -t가 필요합니다.

--spread

--cpus로 지정된 CPU 간 **--threads**로 지정된 스레드를 균등하게 분배합니다. -t 또는 -q가 필요합니다.

--help

옵션 목록을 인쇄합니다. 인쇄 후 tuna는 명령행의 나머지 부분을 무시하고 종료됩니다.

--version

버전을 표시합니다.

A.3. ethtool

ethtool 유틸리티를 통해 관리자는 네트워크 인터페이스 카드 설정을 확인 및 편집할 수 있습니다. 이는 특정 장치에서 드롭된 패킷 수와 같이 특정 장치의 통계를 확인하는데 유용합니다.

ethtool, 옵션, 사용법 등에 대한 자세한 내용은 man 페이지에 종합적으로 설명되어 있습니다.

```
$ man ethtool
```

A.4. ss

ss는 소켓에 대한 통계 정보를 출력하는 명령행 유틸리티로 이를 통해 관리자는 장치 성능을 평가할 수 있습니다. 기본적으로 **ss**는 연결된 오픈 비수신 TCP 소켓을 나열하지만 특정 소켓에 대한 통계를 필터링하는데 관리자에게 유용한 여러 옵션도 제공합니다.

일반적으로 사용되는 명령 중 하나로 **`ss -tmpie`**가 있습니다. 이는 모든 TCP 소켓 (**`t`**), 내부 TCP 정보 (**`i`**), 소켓 메모리 사용량 (**`m`**), 소켓을 사용하는 프로세스 (**`p`**), 자세한 소켓 정보 (**`i`**) 등을 표시합니다.

Red Hat은 Red Hat Enterprise Linux 7에서 **netstat** 보다 **ss**를 사용할 것을 권장합니다.

ss는 *iproute* 패키지에서 제공됩니다. 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man ss
```

A.5. tuned

Tuned는 튜닝 프로파일을 설정하여 특정 워크로드에서 성능을 개선하기 위해 운영 체제를 적합하게 튜닝하는 데몬입니다. CPU 및 네트워크 사용 변경에 반응하도록 설정하고 작동하는 장치에서 성능을 개선하고 작동하지 않는 장치에서 전력 소비를 감소하도록 설정 조정할 수 있습니다.

동적 튜닝 동작을 설정하려면 **`/etc/tuned/tuned-main.conf`** 파일의 ***`dynamic_tuning`*** 매개 변수를 편집합니다. 또한 ***`update_interval`*** 매개 변수로 tuned 사용량 체크와 튜닝 세부 사항 업데이트간의 간격을 초 단위로 설정할 수 있습니다.

tuned에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man tuned
```

A.6. tuned-adm

tuned-adm은 명령행 도구로 특정 사용 경우에 성능을 개선하기 위한 여러 다른 프로파일을 제공합니다. 또한 시스템을 평가하고 권장 튜닝 프로파일을 출력하는 하위 명령 (**`tuned-adm recommend`**)을 제공합니다. 이러한 하위 명령은 설치 시 시스템에 기본 프로파일을 설정하여 기본 프로파일로 돌아갈 때 사용할 수 있습니다.

Red Hat Enterprise Linux 7에는 **tuned-adm**이 포함되어 있어 튜닝 프로파일을 활성화 또는 비활성화의 일부로 명령을 실행할 수 있습니다. 이를 통해 적용할 튜닝 프로파일을 선택하기 전 **tuned-adm**에서 사용할 수 없는 시스템이 마스터베이스 노드인지를 확인하는 것과 같은 환경 별 확인 사항을 추가할 수 있습니다.

또한 Red Hat Enterprise Linux 7에서는 프로파일 정의 파일에 ***include*** 매개 변수를 제공하여 기존 프로파일에 자신의 **tuned-adm** 프로파일을 기반하도록 할 수 있습니다.

다음의 튜닝 프로파일이 **tuned-adm**에서 제공되며 Red Hat Enterprise Linux 7에서 지원됩니다.

throughput-performance

처리량 개선에 중점을 둔 서버 프로파일입니다. 이는 기본 프로파일로 대부분의 시스템에 사용 권장됩니다.

이 프로파일은 **intel_pstate** 및 **min_perf_pct=100**을 설정하여 절전 보다 성능을 우선시합니다. 이는 투명한 huge 페이지를 활성화하고 **cpupower**를 사용하여 **performance** cpufreq 관리자를 설정하고 입/출력 스케줄러를 **deadline**으로 설정합니다. 또한 이는 ***kernel.sched_min_granularity_ns***를 **10** µs으로 ***kernel.sched_wakeup_granularity_ns***를 **15** µs로 ***vm.dirty_ratio***를 **40**%로 설정합니다.

latency-performance

대기 시간 단축에 중점을 둔 서버 프로파일입니다. 이 프로파일은 c-state 튜닝이나 투명한 huge 페이지의 TLB 효율성을 증가시키기 위해 대기 시간 제약이 있는 워크로드에 권장되는 프로파일입니다.

이 프로파일은 **intel_pstate** 및 **max_perf_pct=100**을 설정하여 절전 보다 성능을 우선시합니다. 이는 투명한 huge 페이지를 활성화하고 **cpupower**를 사용하여 **performance** cpufreq 관리자를 설정하고 ***cpu_dma_latency*** 값으로 **1**을 요청합니다.

network-latency

네트워크 대기 시간 단축에 중점을 둔 서버 프로파일입니다.

이 프로파일은 **intel_pstate** 및 **min_perf_pct=100**을 설정하여 절전 보다 성능을 우선시합니다. 이는 투명한 huge 페이지 및 자동 NUMA 밸런싱을 비활성화합니다. **cpupower**를 사용하여 **performance** cpufreq 관리자를 설정하고 ***cpu_dma_latency*** 값으로 **1**을 요청합니다. 이는 ***busy_read*** 및 ***busy_poll*** 시간을 **50** µs으로 설정하고 ***tcp_fastopen***을 **3**으로 설정합니다.

network-throughput

네트워크 처리량 개선에 중점을 둔 서버 프로파일입니다.

이 프로파일은 **intel_pstate** 및 **max_perf_pct=100**을 설정하고 커널 네트워크 버퍼 크기를 늘려 절전 보다 성능을 우선시합니다. 이는 투명한 huge 페이지를 활성화하고 **cpupower**를 사용하여 **performance** cpufreq 관리자를 설정합니다. 또한 이는 ***kernel.sched_min_granularity_ns***를 **10** µs으로 ***kernel.sched_wakeup_granularity_ns***를 15 µs로 ***vm.dirty_ratio***를 **40**%로 설정합니다.

virtual-guest

Red Hat Enterprise Linux 7 가상 머신에서 성능 최적화에 중점을 둔 프로파일입니다.

이 프로파일은 **intel_pstate** 및 **max_perf_pct=100**을 설정하여 절전 보다 성능을 우선시합니다. 또한 이는 가상 머신의 스왑 공간을 줄일 수 있습니다. 투명한 huge 페이지를 활성화하고 **cpupower**를 사용하여 **performance** cpufreq 관리자를 설정합니다. 또한 이는 ***kernel.sched_min_granularity_ns***를 **10** µs으로 ***kernel.sched_wakeup_granularity_ns***를 15 µs로 ***vm.dirty_ratio***를 **40**%로 설정합니다.

virtual-host

Red Hat Enterprise Linux 7 가상 호스트에서 성능 최적화에 중점을 둔 프로파일입니다.

이 프로파일은 **intel_pstate** 및 **max_perf_pct=100**을 설정하여 절전 보다 성능을 우선시합니다. 또한 이는 가상 머신의 스왑 공간을 줄일 수 있습니다. 이러한 프로파일은 투명한 huge 페이지를 활성화하고 더티 페이지를 보다 자주 디스크에 기록합니다. **cpupower**를 사용하여 **performance** cpufreq 관리자를 설정합니다. 또한 이는 ***kernel.sched_min_granularity_ns***를 **10** µs으로 ***kernel.sched_wakeup_granularity_ns***를 15 µs로 ***kernel.sched_migration_cost***를 **5** µs로 ***vm.dirty_ratio***를 **40**%로 설정합니다.

tuned-adm에서 제공되는 절전 프로파일에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 *Red Hat Enterprise Linux 7 Power Management Guide*에서 참조하십시오.

tuned-adm 사용에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man tuned-adm
```

A.7. perf

perf 도구는 여러 유용한 명령을 제공하며 이러한 명령 중 일부는 본 섹션에 설명되어 있습니다. **perf**에 대한 보다 자세한 내용은 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/의 *Red Hat Enterprise Linux 7 Developer Guide* 또는 man 페이지에서 참조하십시오.

perf stat

이 명령은 실행된 지시 사항 및 소비된 클럭 사이클을 포함하여 일반적인 성능 이벤트의 전체 통계를 제공합니다. 옵션 플래그를 사용하여 기본 측정 이벤트 이외에 이벤트의 통계를 수집할 수 있습니다. Red Hat Enterprise Linux 6.4에서 **perf stat**를 사용하여 하나 이상의 지정된 컨트롤 그룹(cgroups)에 기반하여 모니터링을 필터링할 수 있습니다.

보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man perf-stat
```

perf record

이 명령은 성능 데이터를 **perf report**를 사용하여 나중에 분석할 수 있는 파일에 기록합니다. 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man perf-record
```

perf report

이 명령은 파일에서 성능 데이터를 읽고 기록된 데이터를 분석합니다. 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man perf-report
```

perf list

이 명령은 특정 컴퓨터에서 사용할 수 있는 이벤트를 나열합니다. 이러한 이벤트는 성능 모니터링 하드웨어 및 시스템의 소프트웨어 설정에 따라 다릅니다. 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man perf-list
```

perf top

이 명령은 **top** 도구와 유사한 기능을 수행합니다. 실시간으로 성능 카운트 프로파일을 생성 및 표시합니다. 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man perf-top
```

perf trace

이 명령은 **strace** 도구와 유사한 기능을 수행합니다. 이는 지정된 스레드 또는 프로세스에서 사용되는 시스템 호출 및 애플리케이션에 의해 수신되는 모든 신호를 모니터링합니다. 추적 대상을 추가할 수 있으며 전체 목록은 man 페이지에서 참조하십시오:

```
$ man perf-trace
```

A.8. PCP (Performance Co-Pilot)

PCP (Performance Co-Pilot)는 여러 명령행 도구, 그래픽 도구 및 라이브러리를 제공합니다. 이러한 도구에 대한 자세한 내용은 man 페이지에서 참조하십시오: 명령행에서 **man *toolname***을 입력하고 *toolname*을 도구 이름으로 변경합니다.

pcp-doc 패키지는 기본적으로 자세한 설명서를 **/usr/share/doc/pcp-doc** 디렉토리에 설치합니다.

Red Hat 고객 포털에는 PCP 사용 방법에 대한 몇몇 유용한 문서가 있습니다. 이러한 문서의 인덱스는 https://access.redhat.com/articles/1145953에서 참조하십시오.

A.9. vmstat

Vmstat는 시스템 프로세스, 메모리, 페이징, 입/출력 차단, 인터럽트, CPU 동작에 대한 보고를 출력합니다. 마지막으로 컴퓨터를 부팅한 시간 또는 마지막 보고 시간에서 이벤트 평균을 바로 보고합니다.

-a

활성 메모리 및 비활성 메모리를 표시합니다.

-f

부팅에서 포크 수를 표시합니다. 이에는 **fork**, **vfork**, **clone** 시스템 호출이 포함되며 생성된 작업 수 합계와 동일합니다. 각 프로세스는 스레드 사용에 따라 여러 작업에서 표시됩니다. 이는 반복 표시되지 않습니다.

-m

슬래브 정보를 표시합니다.

-n

헤더가 주기적이 아니라 한 번만 나타나도록 지정합니다.

-s

다양한 이벤트 카운터 및 메모리 통계 테이블을 표시합니다. 반복 표시되지 않습니다.

delay

리포트 사이의 지연 시간을 초 단위로 지정합니다. 지연 시간이 지정되어 있지 않은 경우 시스템이 마지막으로 시작된 시점에서 평균적으로 하나의 리포트만 출력됩니다.

count

시스템에 보고할 횟수를 지정합니다. 횟수가 지정되어 있지 않고 지연 시작이 지정되어 있을 경우 **vmstat**는 무기한으로 보고합니다.

-d

디스크 통계를 표시합니다.

-p

파티션 이름을 값으로 지정하고 해당 파티션에 대한 자세한 통계를 보고합니다.

-S

보고서에 의해 출력되는 단위를 지정합니다. 사용 가능한 값은 **k** (1000 바이트), **K** (1024 바이트), **m** (1000000 바이트), **M** (1048576 바이트)입니다.

-D

디스크 활동에 대한 통계 요약을 보고합니다.

각 출력 모드에서 제공하는 출력에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man vmstat
```

A.10. x86_energy_perf_policy

x86_energy_perf_policy 도구를 사용하여 관리자는 성능 및 전력 효율성의 상대적 중요성을 지정할 수 있습니다. 이는 *kernel-tools* 패키지에서 제공됩니다.

현재 정책을 확인하려면 다음 명령을 실행합니다:

```
# x86_energy_perf_policy -r
```

새로운 정책을 설정하려면 다음 명령을 실행합니다:

```
# x86_energy_perf_policy profile_name
```

profile_name을 다음 프로파일 중 하나로 변경합니다.

performance

프로세서는 에너지 절약을 위해 성능을 낭비하지 않습니다. 이는 기본값입니다.

normal

절전 가능성이 있는 경우 프로세서는 최소한의 성능 저하를 감수해야 합니다. 이는 대부분의 서버 및 데스크톱에 적절한 설정입니다.

powersave

에너지 효율성을 최대화하기 위해 프로세서 성능이 저하될 가능성을 감수해야 합니다.

x86_energy_perf_policy 사용 방법에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man x86_energy_perf_policy
```

A.11. turbostat

turbostat 도구는 시스템이 다른 상태에서 소요되는 시간에 대한 자세한 정보를 제공합니다. **Turbostat**는 *kernel-tools* 패키지에서 제공됩니다.

기본적으로 **turbostat**는 전체 시스템의 카운터 결과 요약을 출력하고 제목 아래에 각 카운터 결과를 5 초마다 출력합니다.

pkg

프로세서 패키지 번호입니다.

core

프로세서 코어 번호입니다.

CPU

Linux CPU (논리 프로세서) 번호입니다.

%c0

CPU가 명령을 사용하지 않는 간격 (백분율)입니다.

GHz

이 번호가 TSC에 있는 값보다 높을 경우 CPU는 터보 모드가 됩니다.

TSC

전체 기간 동안 평균 클럭 속도입니다.

%c1, %c3, %c6

프로세서가 각각 c1, c3, c6 상태였던 간격 (백분율)입니다.

%pc3 또는 %pc6

프로세서가 각각 pc3 또는 pc6 상태였던 간격 (백분율)입니다.

`-i` 옵션으로 카운터 결과 출력 간격을 지정합니다. 예를 들어 10 초 마다 결과를 출력하려면 **`turbostat -i 10`**을 실행합니다.

참고

차후 Intel 프로세서에는 c 상태가 추가될 수 있습니다. Red Hat Enterprise Linux 7.0에서 **turbostat**는 c7, c8, c9, c10 상태를 지원합니다.

A.12. numastat

numastat 도구는 *numactl* 패키지를 통해 제공되며 NUMA 노드 기반에서 프로세서 및 운영 체제의 메모리 통계 (할당 히트 및 누락 사항)를 표시합니다. **`numastat`** 명령의 기본 추적 카테고리는 다음에서 설명합니다:

numa_hit

이 노드에 성공적으로 할당된 페이지 수입니다.

numa_miss

원래 의도된 노드에 메모리가 부족하여 이 노드에 할당된 페이지 수입니다. 각각의 **numa_miss** 이벤트는 해당하는 **numa_foreign** 이벤트가 다른 노드에 있습니다.

numa_foreign

다른 노드에 할당하려 했으나 처음에 의도된 노드에 할당된 페이지 수입니다. 각각의 **numa_foreign** 이벤트에 해당하는 **numa_miss** 이벤트가 다른 노드에 있습니다.

interleave_hit

이 노드에 성공적으로 할당된 인터리브 정책 페이지 수입니다.

local_node

이 노드의 프로세스에 의해 이 노드에 성공적으로 할당된 페이지 수입니다.

other_node

다른 노드의 프로세스에 의해 이 노드에 할당된 페이지 수입니다.

다음 옵션 중 하나를 적용하면 메모리의 표시 단위가 메가바이트로 변경됩니다. (소수점 두 자리 까지 반올림됨) 다음에서 설명하고 있듯이 다른 특정 **numastat** 동작을 변경합니다.

-c

표시된 정보 테이블을 가로로 축소합니다. 이는 NUMA 노드 수가 많은 시스템에서 유용하지만 열의 폭과 열 사이의 간격은 예측 불가능합니다. 이 옵션을 사용하면 메모리 양은 가장 가까운 메가바이트로 반올림됩니다.

-m

노드 당 시스템 전체 메모리 사용량을 표시합니다. 이는 **/proc/meminfo**에 있는 정보와 유사합니다.

-n

업데이트된 형식으로 측정 단위로 메가바이트를 사용하여 원래의 umastat 명령 (**numa_hit**, **numa_miss**, **numa_foreign**, **interleave_hit**, **local_node**, **other_node**)과 동일한 정보가 표시됩니다.

-p pattern

지정된 패턴의 노드 당 메모리 정보를 표시합니다. 패턴 값이 숫자로 구성되면 **numastat**는 이를 숫자 프로세스 식별자로 간주합니다. 그렇지 않을 경우 **numastat**는 지정된 패턴의 프로세스 명령행을 검색합니다.

-p 옵션 값 다음에 입력되는 명령행 인수는 필터링 추가 패턴으로 간주됩니다. 추가 패턴은 필터를 좁히는 것이 아니라 확장합니다.

-s

표시된 데이터를 내림 차순으로 정렬하므로 (전체 칼럼의) 메모리 사용량이 많은 것이 먼저 나열됩니다.

옵션으로 node를 지정하면 표는 node 칼럼에 정렬됩니다. 이 옵션을 사용할 때 다음과 같이 node 값은 **-s** 옵션 바로 뒤에 와야 합니다.

```
numastat -s2
```

옵션과 값 사이에 공백을 넣지 마십시오.

-v

자세한 정보를 표시합니다. 즉 여러 프로세스의 프로세스 정보는 각 프로세스에 대해 자세한 정보를 표시합니다.

-V

numastat 버전 정보를 표시합니다.

-z

표시된 정보에서 제로 값을 갖는 표의 행과 열을 생략합니다. 표시 목적으로 제로로 반올림된 제로에 가까운 값은 표시된 출력에서 생략되지 않음에 유의합니다.

A.13. numactl

관리자는 **Numactl**을 사용하여 지정된 스케줄링 또는 메모리 배치 정책에 따라 프로세스를 실행할 수 있습니다. **Numactl**은 공유 메모리 세그먼트나 파일에 영구 정책을 설정하고 프로세서 친화도 및 메모리 친화도를 설정할 수 있습니다.

Numactl은 여러 유용한 옵션을 제공합니다. 다음 부분에서는 이러한 옵션 중 일부만을 소개하고 사용 방법에 대해 설명합니다.

--hardware

노드 간의 상대 거리를 포함하여 시스템에서 사용 가능한 노드의 인벤토리를 표시합니다.

--membind

특정 노드에서만 메모리를 할당하도록 합니다. 지정된 위치에 사용 가능한 메모리가 충분하지 않을 경우 할당 실패합니다.

--cpunodebind

지정된 노드에서만 지정된 명령 및 하위 프로세스를 실행하도록 합니다.

--phycpubind

지정된 프로세서에서만 지정된 명령 및 하위 프로세스를 실행하도록 합니다.

--localalloc

로컬 노드에서 항상 할당되어야 하는 메모리를 지정합니다.

--preferred

메모리 할당에서 기본 설정 노드를 지정합니다. 지정된 노드에서 메모리를 할당할 수 없는 경우 다른 노드가 대체로 사용됩니다.

이러한 매개 변수 및 기타 다른 매개 변수에 대한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man numactl
```

A.14. numad

numad는 NUMA 친화도 자동 관리 데몬입니다. 이는NUMA 리소스 할당 및 관리 개선을 위해 NUMA 토폴로지 및 시스템 리소스 사용을 모니터링합니다.

numad가 활성화되어 있을 때 기본 동작은 자동 NUMA 밸런싱 동작을 무시함에 유의하십시오.

A.14.1. 명령행에서 numad 사용

numad를 실행 파일로 사용하려면 다음 명령을 실행합니다:

```
# numad
```

numad가 실행되는 동안 실행 동작은 **/var/log/numad.log**에 기록됩니다. 다음 명령으로 동작이 중지될 때 까지 계속됩니다.

```
# numad -i 0
```

numad를 중지해도 NUMA 친화도를 개선하기 위해 변경한 내용이 삭제되지 않습니다. 시스템 사용이 크게 변경되는 경우 **numad**를 다시 실행하여 친화도를 조정하고 새로운 조건 하에서 성능이 향상됩니다.

numad 관리를 특정 프로세스로 제한하려면 다음과 같은 옵션을 사용하여 시작합니다.

```
# numad -S 0 -p pid
```

-p *pid*

이 옵션은 지정된 *pid*를 명시적 대상 목록에 추가합니다. 지정된 프로세스는 **numad** 프로세스 중요도 임계값에 도달할 때 까지 관리되지 않습니다.

-S 0

이는 프로세스 스캔 유형을 **0**으로 설정하여 **numad** 관리를 명시적으로 대상 프로세스에 한정합니다.

사용 가능한 **numad** 옵션에 대한 보다 자세한 내용은 **numad** man 페이지에서 참조하십시오:

```
$ man numad
```

A.14.2. numad를 서비스로 사용

numad가 서비스로 실행되는 동안 현재 시스템 워크로드에 따라 시스템을 동적으로 튜닝합니다. 실행 동작은 **/var/log/numad.log**에 기록됩니다.

서비스를 시작하려면 다음 명령을 실행합니다:

```
# systemctl start numad.service
```

재부팅 후에도 서비스를 유지하려면 다음 명령을 실행합니다:

```
# chkconfig numad on
```

사용 가능한 **numad** 옵션에 대한 보다 자세한 내용은 **numad** man 페이지에서 참조하십시오:

```
$ man numad
```

A.14.3. 사전 배포 컨설팅 서비스

numad는 다양한 작업 관리 시스템을 쿼리할 수 있는 사전 배포 컨설팅 서비스도 제공하고 있으며 프로세스의 CPU 및 메모리 리소스의 초기 바인딩을 제공합니다. 이러한 사전 배포 컨설팅 서비스는 시스템에서 **numad**가 실행 가능한 파일로 또는 서비스로 실행되고 있는지에 대한 여부와 상관 없이 사용 가능합니다.

A.14.4. KSM으로 numad 사용

KSM이 NUMA 시스템에서 사용되고 있을 경우 **/sys/kernel/mm/ksm/merge_nodes** 매개 변수 값을 **0**으로 변경하여 NUMA 노드에서 페이지가 병합되지 않도록 합니다. 변경하지 않을 경우 KSM은 노드에서 페이지를 병합하기 때문에 원격 메모리 액세스가 증가합니다. 또한 노드에서 대량 병합된 후 결과적으로 커널 메모리 계정 통계가 서로 상반될 수 있습니다. 이러한 경우 KSM 데몬이 여러 메모리 페이지를 병합한 후 **numad**는 사용 가능한 정확한 메모리 양과 위치에 대해 혼동될 수 있습니다. 시스템에서 메모리를 오버커밋하는 경우에만 KSM이 유용합니다. 시스템에 여유 메모리가 충분할 경우 KSM 데몬을 해제 및 종료하여 시스템 성능을 높일 수 있습니다.

A.15. OProfile

OProfile은 *oprofile* 패키지를 통해 제공되는 오버헤드가 낮은 시스템 전역 성능 모니터링 도구입니다. 이는 프로세서에서 성능 모니터링 하드웨어를 사용하여 메모리 참조 시기, 두 번째 레벨 캐시 요청 수, 하드웨어 인터럽트 수신 수 등과 같은 시스템의 커널 및 실행 파일에 대한 정보를 검색합니다. OProfile은 JVM (Java Virtual Machine)에서 실행되는 애플리케이션을 프로파일링할 수 있습니다.

OProfile은 다음과 같은 도구를 제공합니다. 기존의 **opcontrol** 도구 및 새로운 **operf** 도구는 상호 배타적임에 유의합니다.

ophelp

시스템 프로세서에 대한 간략한 설명과 함께 사용 가능한 이벤트를 표시합니다.

opimport

샘플 데이터베이스 파일을 외부 바이너리 형식에서 네이티브 형식으로 전환합니다. 이 옵션은 다른 아키텍처에서 샘플 데이터베이스를 분석할 때에만 사용합니다.

opannotate

애플리케이션이 디버깅 심볼로 컴파일되어 있을 경우 실행 파일의 주석 소스를 생성합니다.

opcontrol

프로파일링 실행에서 수집될 데이터를 설정합니다.

operf

opcontrol을 변경합니다. **operf** 도구는 Linux Performance Events 서브 시스템을 사용하기 때문에 단일 프로세스나 시스템 전역에 걸쳐 보다 정확하게 프로파일링할 수 있으며 OProfile이 시스템에서 성능 모니터링 하드웨어를 사용하는 다른 도구와 보다 더 잘 공존할 수 있게 합니다. **opcontrol**과는 다르게 초기 설정이 필요하지 않고 **--system-wide** 옵션을 사용하고 있지 않은 경우 root 권한 없이 사용할 수 있습니다.

opreport

프로파일 데이터를 검색합니다.

oprofiled

데몬으로 실행하고 디스크에 샘플 데이터를 주기적으로 기록합니다.

레거시 모드 (**opcontrol**, **oprofiled**, 사후 처리 도구)는 여전히 사용 가능하지만 더이상 프로파일링 방법으로 권장되지 않습니다.

이러한 명령에 대한 보다 자세한 내용은 OProfile man 페이지에서 참조하십시오:

```
$ man oprofile
```

A.16. taskset

taskset 도구는 *util-linux* 패키지로 제공됩니다. 이를 사용하여 관리자는 실행 중인 프로세스의 친화도를 검색 및 설정하거나 지정된 프로세스 친화도로 프로세스를 시작할 수 있습니다.

중요

taskset은 로컬 메모리 할당을 보장하지 않습니다. 로컬 메모리 할당을 통해 추가적으로 성능을 향상시켜야 할 경우 Red Hat은 taskset 대신 **numactl**을 사용할 것을 권장합니다.

실행 중인 프로세스의 CPU 친화도를 설정하려면 다음 명령을 실행합니다:

```
# taskset -c processors pid
```

*processors*를 콤마로 구분된 프로세서 목록 또는 프로세서 범위 (예: **1,3,5-7**)로 변경합니다. *pid*는 재설정하려는 프로세스의 프로세스 ID로 변경합니다.

지정된 친화도로 프로세스를 시작하려면 다음 명령을 실행합니다:

```
# taskset -c processors -- application
```

*processors*를 콤마로 구분된 프로세서 목록 또는 프로세서 범위로 변경합니다. *application*은 실행하려는 애플리케이션의 명령, 옵션, 인수로 변경합니다.

taskset에 관한 보다 자세한 내용은 man 페이지에서 참조하십시오:

```
$ man taskset
```

A.17. SystemTap

SystemTap은 사용 설명서에 자세히 설명되어 있습니다. Red Hat Enterprise Linux 7 버전의 *SystemTap Beginner's Guide* 및 *SystemTap TapSet Reference*는 모두 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/에서 확인하실 수 있습니다.

부록 B. 고친 과정

고침 10.08-39.2 **Thu Nov 17 2016** **Eun-Ju Kim**
한국어 번역 완료

고침 10.08-39.1 **Thu Nov 17 2016** **Eun-Ju Kim**
XML 소스 10.08-39 버전과 번역 파일을 동기화

고침 10.08-39 **Wed Nov 11 2015** **Jana Heves**
7.2GA 릴리즈 버전

고침 10.08-36 **Mon Nov 9 2015** **Jana Heves**
RH 교육 코스 관련 링크 추가

고침 10.08-35 **Wed Aug 6 2015** **Charlie Boyle**
5.3.7.3에 BTRFS 정보 추가.

고침 04.7-33 **Wed Aug 5 2015** **Charlie Boyle**
5.1.3에 BTRFS 압축 정보 추가.

고침 03.6-32 **Mon Aug 3 2015** **Charlie Boyle**
5.1.3 파일 시스템에서 파일 시스템의 제목 수준 변경.

고침 03.4-32 **Wed Jun 5 2015** **Charlie Boyle**
1189350의 SSD 정보 5.5.1 부분을 업데이트.

고침 03.4-31 **Wed Apr 8 2015** **Charlie Boyle**
man 페이지를 반영하기 위해 5.2.1에 VMSTAT 매개 변수 업데이트. (BZ1131829)

고침 02.4-30 **Wed Apr 8 2015** **Charlie Boyle**
man 페이지를 반영하기 위해 VMSTAT 매개 변수 업데이트. (BZ1131829)

고침 02.4-29 **Tue Apr 7 2015** **Charlie Boyle**
Tuna 도움말 업데이트. (BZ1131829)

고침 02.4-28 **Tue Apr 7 2015** **Charlie Boyle**
Turbostat Ghz 설명 변경. (BZ992461)

고침 02.4-27 **Fri Mar 27 2015** **Charlie Boyle**
매개 변수 설정 수정. (BZ1145906)

고침 02.4-26 **Thu Mar 26 2015** **Charlie Boyle**
섹션 4.3 설정 도구에 대한 내용 수정. (BZ1122400)

고침 0.3-25 **Thu Mar 26 2015** **Charlie Boyle**
처리 성능에 대한 내용 수정. (BZ1122132)

고침 0.3-24 **Mon Feb 23 2015** **Laura Bailey**
비지 폴링 지원 확인에 대한 상세 정보 수정. (BZ1080703)

고침 0.3-23 **Tue Feb 17 2015** **Laura Bailey**

RHEL 7.1 GA 용으로 빌드.

고침 0.3-22 **Tue Feb 17 2015** **Laura Bailey**

비지 폴링 지원 확인 추가. (BZ1080703)

고침 0.3-21 **Thu Feb 05 2015** **Laura Bailey**

새로운 조정 프로파일 매개 변수 cmdline에 대해 설명. (BZ1130818)
'7.1에서 새로운 기능' 부분에 swapon 명령의 새로운 discard 매개 변수에 대한 주의 사항을 추가. (BZ113082

고침 0.3-20 **Fri Jan 09 2015** **Charles Boyle**

pgrep 명령에서 오류 수정. (BZ1155253)
vm.swappiness에 대한 설명 수정. (BZ1148419)

고침 0.3-19 **Fri Dec 05 2014** **Laura Bailey**

스플래쉬 페이지 (시작 페이지)의 sort_order 업데이트.

고침 0.3-18 **Wed Nov 26 2014** **Laura Bailey**

7.1 Beta에 PCP 기사 색인 링크 추가. (BZ1083387)

고침 0.3-15 **Mon Nov 24 2014** **Laura Bailey**

유휴 상태 밸런싱 변경에 대한 내용 추가. (BZ1131851)

고침 0.3-14 **Wed Nov 19 2014** **Laura Bailey**

Huge 페이지 할당에 대한 상세 내용 업데이트. (BZ1131367)
런타임 시 Huge 페이지 할당하는 방법에 대한 예시 추가. (BZ1131367)

고침 0.3-12 **Thu Nov 13 2014** **Laura Bailey**

RHEL 7.1에서 SHMALL 및 SHMMAX의 새 기본값 추가. (BZ1131848)

고침 0.3-11 **Mon Nov 10 2014** **Laura Bailey**

ext4의 클러스터 할당 / bigalloc 상태 지원에 대한 설명 추가. (BZ794607)

고침 0.3-10 **Fri Oct 31 2014** **Laura Bailey**

노드 당 정적 huge 페이지에 대해 문서화. (BZ1131832)

고침 0.3-9 **Tue Jul 22 2014** **Laura Bailey**

latencytap.stp 스크립트에 대한 설명 추가. (BZ988155)

고침 0.3-7 **Thu Jun 26 2014** **Laura Bailey**

CPU 장에서 오타 수정. (Jiri Hladky님에게 감사)
I/O 스케줄러 조정 대안에 대한 내용 삭제. (Jiri Hladky님에게 감사)

고침 0.3-5 **Wed Jun 11 2014** **Laura Bailey**

리디렉션되지 않은 access.redhat.com 링크 뒤에 슬래시 추가.

고침 0.3-4 **Tue Jun 10 2014** **Laura Bailey**

irqbalance 부록에 인터럽트 및 CPU 금지 내용 추가. (BZ852981)

고침 0.3-3 **Mon Apr 07 2014** **Laura Bailey**

RHEL 7.0 GA 용으로 다시 빌드.

고침 0.3-2 **Mon Apr 07 2014** **Laura Bailey**

RT#294949 용으로 책 구성을 업데이트.

고침 0.2-38 **Mon Apr 07 2014** **Laura Bailey**

업데이트된 OProfile 데이터 추가. (BZ955882)
오래된 코멘트 삭제.

고침 0.2-34 **Fri Apr 04 2014** **Laura Bailey**

lstopo 출력 이미지 스타일 수정. (BZ1042800)
irqbalance 데몬에 대한 세부 사항 추가. (BZ955890)
제어 그룹에 대한 세부 사항 추가 및 수정. (BZ794624)
PCP에 대한 세부 사항 추가 (BZ955883)
XFS 튜닝에 대한 자세한 내용 업데이트. (BZ794616)
업데이트된 OProfile 데이터 추가. (BZ955882)

고침 0.2-27 **Fri Mar 28 2014** **Laura Bailey**

Jeremy Eder의 피드백에 따라 busy_poll 섹션 수정. (RT276607)
Jeremy Eder의 피드백에 따라 nohz_full 섹션 수정 및 세부 정보 추가. (RT284423)
SystemTap 섹션에 세부 정보 추가. (BZ955884)
SSD 섹션에 세부 정보 추가. (BZ955900)
tuned-adm recommend 명령에 세부 정보 추가. (BZ794623)
기능 섹션에서 자동 NUMA 밸런싱에 대한 내용 수정. (BZ794612)
용어 문제 및 새 이미지가 포함된 NUMA 관련 출력 문제 예제 수정. (BZ1042800)
Jeremy Eder의 피드백에 따라 RSS에 연결된 irqbalance에 대한 세부 정보 수정.

고침 0.2-19 **Fri Mar 21 2014** **Laura Bailey**

메모리 장에 Transparent Huge Transparent Huge Pages에 대한 세부 정보 추가. (BZ794621)
NUMA 노드 관련 용어 사용 수정. (BZ1042800)
커널 제한 업데이트. (BZ955894)
틱리스 커널 섹션 초안 작성. (RT284423)
비지 폴링 섹션 초안 작성. (RT276607)
파일 시스템 장애에 대한 정보 업데이트.
노드 당 huge 페이지 할당에 대한 명확하지 않은 내용 삭제. 차후 보다 유용한 정보를 추가하기 위해 BZ1079079 생성.
SSD에 대한 세부 정보 추가. (BZ955900)
리뷰 표시 삭제.

고침 0.2-14 **Thu Mar 13 2014** **Laura Bailey**

Jeremy Eder 및 Joe Mario의 피드백 적용.
Tuna GUI에 대한 업데이트 공지. (BZ955872)
네트워크 장 및 도구 참조 부록에서 SystemTap에 대한 세부 사항 추가. (BZ955884)

고침 0.2-12 **Fri Mar 07 2014** **Laura Bailey**

자동 NUMA 마이그레이션에 대해 설명. (BZ794612)
Jeremy Eder의 추가 피드백 적용.

고침 0.2-11 **Fri Mar 07 2014** **Laura Bailey**

Jeremy Eder의 피드백 적용.

고침 0.2-10 **Mon Feb 24 2014** **Laura Bailey**

Lukáš Czerner의 피드백에 따라 Ext4 정보 수정. (BZ#794607)

고침 0.2-9 **Mon Feb 17 2014** **Laura Bailey**

Bill Gray의 피드백에 따라 CPU 장 수정.
Bill Gray의 피드백에 따라 메모리 장 및 도구 참조에 내용 추가 및 수정.

고침 0.2-8 **Mon Feb 10 2014** **Laura Bailey**

CPU 장에 isolcpus 부팅 매개 변수 추가. (RT276607)
SME 피드백: 매개 변수 설명 수정 및 새 매개 변수 추가. (BZ#970844)
네트워크 장에 권장 tuned-adm 프로파일 추가.
검토를 위해 플래그 섹션에 설명 추가.

고침 0.2-4 **Mon Feb 03 2014** **Laura Bailey**

`numactl --membind` 매개 변수가 문서화되어 있는지 확인. (BZ#922070)
도구 소개, CPU 장, 도구 참조 부록에 Tuna에 대한 세부 사항 추가. (BZ#970844)
스토리지 및 파일 시스템 장에 구성 오류 수정.
누락된 상호 참조 추가.

고침 0.2-2 **Fri Jan 31 2014** **Laura Bailey**

재작성 및 재구성 완료.
가이드에서 언급된 모든 도구가 이를 제공하는 패키지와 함께 나열되어 있는지 확인.

고침 0.1-11 **Thu Dec 05 2013** **Laura Bailey**

RHEL 7.0 베타 버전 용 가이드 재구성.

고침 0.1-10 **Wed Nov 27 2013** **Laura Bailey**

베타 이전 사용자 정의 빌드.

고침 0.1-9 **Tue Oct 15 2013** **Laura Bailey**

고객 피드백에 따라 약간의 내용 수정. (BZ#1011676)

고침 0.1-7 **Mon Sep 09 2013** **Laura Bailey**

RHEL 6.5에서 새로운 컨텐츠를 병합.
편집자 피드백 적용.

고침 0.1-6 **Wed May 29 2013** **Laura Bailey**

ext4 파일 시스템 제한을 업데이트. (BZ#794607)
64 비트 파일 시스템의 이론적 최대 값을 수정.
성능 관련 변경 사항을 추적하기 위해 새로운 기능 섹션 추가.
기본적 I/O 스케줄러를 **`cfq`**에서 **`deadline`**으로 변경. (BZ#794602)
BTRFS 튜닝 초안을 추가. (BZ#794604)
XFS 섹션을 업데이트하고 디렉토리 블록 크기에 대한 보다 명확한 권장 사항 제공 및 XFS 지원 제한 업데이트. (BZ#794616)

고침 0.1-2 **Thurs Jan 31 2013** **Tahlia Richardson**

RHEL 7 초안으로 업데이트 및 공개.

고침 0.1-1 **Wed Jan 16 2013** **Laura Bailey**

RHEL 6.4 버전에서 분기됨.