Red Hat Enterprise Linux 9

시스템 상태 및 성능 모니터링 및 관리

시스템 처리량, 대기 시간 및 전력 소비 최적화

Last Updated: 2022-08-29

Red Hat Enterprise Linux 9 시스템 상태 및 성능 모니터링 및 관리

시스템 처리량, 대기 시간 및 전력 소비 최적화

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

법적 공지

초록

이 문서 컬렉션은 다양한 시나리오에서 Red Hat Enterprise Linux 9의 처리량, 대기 시간 및 전력 소비를 모니터링하고 최적화하는 방법에 대한 지침을 제공합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체 9

RED HAT 문서에 관한 피드백 제공 10

1장. TUNED 시작하기 11
1.1. TUNED의 목적 11
1.2. TUNED 프로필 11
프로필 구성의 구문 11
1.3. 기본 TUNED 프로필 12
1.4. 병합된 TUNED 프로필 12
1.5. TUNED 프로필의 위치 12
1.6. RHEL을 사용하여 배포된 TUNED 프로필 13
1.7. TUNED CPU-PARTITIONING 프로필 16
1.8. 대기 시간이 짧은 튜닝을 위해 TUNED CPU-PARTITIONING 프로파일 사용 18
1.9. CPU-PARTITIONING TUNED 프로파일 사용자 정의 19
1.10. RHEL을 사용하여 실시간 TUNED 프로필 배포 20
1.11. TUNED의 정적 및 동적 튜닝 21
1.12. TUNED NO-DAEMON 모드 22
1.13. TUNED 설치 및 활성화 22
1.14. 사용 가능한 TUNED 프로필 나열 23
1.15. TUNED 프로필 설정 24
1.16. TUNED 비활성화 25

2장. TUNED 프로필 사용자 정의 27
2.1. TUNED 프로필 27
프로필 구성의 구문 27
2.2. 기본 TUNED 프로필 28
2.3. 병합된 TUNED 프로필 28
2.4. TUNED 프로필의 위치 29
2.5. TUNED 프로필 간 상속 29
2.6. TUNED의 정적 및 동적 튜닝 30
2.7. TUNED 플러그인 31
TuneD 프로필의 플러그인 구문 32
짧은 플러그인 구문 33
프로필에서 플러그인 정의 충돌 34
2.8. 사용 가능한 TUNED 플러그인 34
모니터링 플러그인 34
플러그인 튜닝 35
2.9. TUNED 프로필의 변수 40
2.10. TUNED 프로파일의 기본 제공 함수 41
2.11. TUNED 프로필에서 사용할 수 있는 기본 제공 함수 42
2.12. 새 TUNED 프로필 생성 44
2.13. 기존 TUNED 프로필 수정 45
2.14. TUNED를 사용하여 디스크 스케줄러 설정 47

3장. TUNA 인터페이스를 사용하여 시스템 검토 51
3.1. TUNA 툴 설치 51
3.2. TUNA 툴을 사용하여 시스템 상태 보기 52
3.3. TUNA 툴을 사용하여 CPU 튜닝 53
3.4. TUNA 툴을 사용하여 IRQ 튜닝 56

4장. RHEL 시스템 역할을 사용하여 성능 모니터링 59

4.1. RHEL 시스템 역할 소개 59
4.2. RHEL 시스템 역할 용어 60
4.3. 시스템에서 RHEL 시스템 역할 설치 60
4.4. 역할 적용 61
4.5. 지표 시스템 역할 소개 64
4.6. 시각화로 로컬 시스템을 모니터링하기 위해 메트릭 시스템 역할을 사용 65
4.7. 지표 시스템 역할을 사용하여 자신을 모니터링하기 위해 개별 시스템 세트를 설정 66
4.8. 메트릭 시스템 역할을 사용하여 로컬 시스템을 통해 중앙 집중식으로 시스템 그룹을 모니터링할 수 있습니다. 67
4.9. METRICS 시스템 역할을 사용하여 시스템을 모니터링하는 동안 인증 설정 68
4.10. METRICS 시스템 역할을 사용하여 SQL SERVER에 대한 메트릭 컬렉션을 구성하고 활성화합니다. 69

5장. PCP 설정 72
5.1. PCP 개요 72
5.2. PCP 설치 및 활성화 73
5.3. 최소 PCP 설정 배포 74
5.4. PCP로 배포된 시스템 서비스 76
5.5. PCP와 함께 배포된 툴 76
5.6. PCP 배포 아키텍처 79
5.7. 권장되는 배포 아키텍처 82
5.8. 크기 조정 요인 82
5.9. PCP 스케일링을 위한 구성 옵션 83
5.10. 예: 중앙 집중식 로깅 배포 분석 84
5.11. 예: 페더레이션 설정 배포 분석 85
5.12. 높은 메모리 사용량 문제 해결 86

6장. PMLOGGER로 성능 데이터 로깅 89
6.1. PMLOGCONF로 PMLOGGER 구성 파일 수정 89
6.2. PMLOGGER 구성 파일을 수동으로 편집 90
6.3. PMLOGGER 서비스 활성화 91
6.4. 메트릭 컬렉션에 대한 클라이언트 시스템 설정 92
6.5. 데이터 수집을 위해 중앙 서버 설정 94
6.6. PMREP로 PCP 로그 아카이브 재생 96

7장. PERFORMANCE CO-INSPECTOR를 사용하여 성능 모니터링 98
7.1. PMDA-JOURNALD로 POSTFIX 모니터링 98
7.2. PCP 차트 애플리케이션을 사용하여 PCP 로그 아카이브를 시각적으로 추적 100
7.3. PCP를 사용하여 SQL 서버에서 데이터 수집 102
7.4. WEC 아카이브에서 PCP 아카이브 생성 105

8장. PCP를 사용한 XFS의 성능 분석 107
8.1. XFS PMDA를 수동으로 설치 107
8.2. PMINFO를 사용하여 XFS 성능 지표 검사 108
8.3. PMSTORE를 사용하여 XFS 성능 지표 재설정 109
8.4. XFS의 PCP 메트릭 그룹 111
8.5. XFS용 장치별 PCP 지표 그룹 112

9장. PCP 메트릭의 그래픽 표현 설정 114
9.1. PCP-ZEROCONF로 PCP 설정 114
9.2. GRAFANA-SERVER 설정 114
9.3. GRAFANA 웹 UI 액세스 116
9.4. PCP REDIS 구성 118
9.5. PCP REDIS 데이터 소스에서 패널 생성 및 경고 120
9.6. 경고에 대한 알림 채널 추가 123

9.7. PCP 구성 요소 간 인증 설정 125
9.8. PCP BPFTRACE 설치 126
9.9. PCP BPFTRACE 시스템 분석 대시보드 보기 128
9.10. PCP 벡터 설치 130
9.11. PCP 벡터 체크리스트 보기 131
9.12. GRAFANA 문제 해결 133

10장. 웹 콘솔을 사용하여 시스템 성능 최적화 136
10.1. 웹 콘솔의 성능 튜닝 옵션 136
10.2. 웹 콘솔에서 성능 프로파일 설정 136
10.3. 웹 콘솔을 사용하여 성능 모니터링 138

11장. 디스크 스케줄러 설정 140
11.1. 사용 가능한 디스크 스케줄러 140
11.2. 다른 사용 사례에 대한 다양한 디스크 스케줄러 141
11.3. 기본 디스크 스케줄러 142
11.4. 활성 디스크 스케줄러 확인 142
11.5. TUNED를 사용하여 디스크 스케줄러 설정 143
11.6. UDEV 규칙을 사용하여 디스크 스케줄러 설정 145
11.7. 특정 디스크에 대한 스케줄러를 임시로 설정 147

12장. SAMBA 서버의 성능 튜닝 148
12.1. SMB 프로토콜 버전 설정 148
12.2. 많은 수의 파일이 포함된 디렉터리와의 공유 튜닝 149
12.3. 성능에 부정적인 영향을 줄 수 있는 설정 150

13장. 가상 머신 성능 최적화 151
13.1. 가상 머신 성능에 미치는 영향 151
가상화가 시스템 성능에 미치는 영향 151
VM 성능 손실 감소 152
13.2. TUNED를 사용하여 가상 머신 성능 최적화 152
13.3. LIBVIRT 데몬 최적화 154
13.3.1. libvirt 데몬 유형 154
13.3.2. 모듈식 libvirt 데몬 활성화 155
13.4. 가상 머신 메모리 구성 157
13.4.1. 웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거 157
13.4.2. 명령줄 인터페이스를 사용하여 가상 머신 메모리 추가 및 제거 159
13.4.3. 추가 리소스 162
13.5. 가상 머신 I/O 성능 최적화 162
13.5.1. 가상 머신에서 블록 I/O 튜닝 162
13.5.2. 가상 머신에서 디스크 I/O 제한 163
13.5.3. 다중 대기열 virtio-scsi 활성화 165
13.6. 가상 머신 CPU 성능 최적화 165
13.6.1. 명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거 166
13.6.2. 웹 콘솔을 사용하여 가상 CPU 관리 167
13.6.3. 가상 머신에서 NUMA 구성 169
13.6.4. 샘플 vCPU 성능 튜닝 시나리오 172
13.6.5. 커널 동일한 페이지 병합 관리 178
13.7. 가상 머신 네트워크 성능 최적화 180
13.8. 가상 머신 성능 모니터링 툴 182
13.9. 추가 리소스 185

14장. POWERTOP를 사용하여 전력 소비 관리 186
14.1. POWERTOP의 목적 186

14.2. POWERTOP 사용 186
14.2.1. PowerTOP 시작 186
14.2.2. PowerTOP 교정 186
14.2.3. 측정 간격 설정 187
14.2.4. 추가 리소스 187
14.3. POWERTOP 통계 188
14.3.1. 개요 탭 188
14.3.2. Idle 통계 탭 189
14.3.3. 장치 통계 탭 189
14.3.4. Tunables 탭 189
14.3.5. WakeUp 탭 190
14.4. POWERTOP에서 일부 인스턴스에서 FREQUENCY STATS 값을 표시하지 않는 이유 190
14.5. HTML 출력 생성 191
14.6. 전력 소비 최적화 192
14.6.1. powertop 서비스를 사용하여 전원 소비 최적화 192
14.6.2. powertop2tuned 유틸리티 192
14.6.3. powertop2tuned 유틸리티를 사용하여 전원 소비 최적화 193
14.6.4. powertop.service 및 powertop2tuned 비교 193

15장. PERF 시작하기 195
15.1. PERF 소개 195
15.2. PERF 설치 195
15.3. 일반적인 PERF 명령 195

16장. PERF를 사용하여 CPU 사용량을 실시간으로 프로파일링 197
16.1. PERF의 목적 197
16.2. PERF를 사용하여 CPU 사용량 프로파일링 197
16.3. 상위 출력의 해석 198
16.4. PERF가 일부 함수 이름을 원시 함수 주소로 표시하는 이유 198
16.5. 디버그 및 소스 리포지토리 활성화 199
16.6. GDB를 사용하여 애플리케이션 또는 라이브러리를 위한 DEBUGINFO 패키지 가져오기 199

17장. PERF STAT를 사용하여 프로세스 실행 중 이벤트 계산 202
17.1. PERF 통계의 목적 202
17.2. 통계로 이벤트 계산 202
17.3. 통계 출력의 해석 204
17.4. 실행 중인 프로세스에 PERF STAT 연결 204

18장. PERF를 사용하여 성능 프로파일 기록 및 분석 206
18.1. PERF 레코드의 목적 206
18.2. 루트 액세스없이 성능 프로파일 기록 206
18.3. 루트 액세스 권한으로 성능 프로파일 기록 207
18.4. CPU 모드에서 성능 프로파일 기록 207
18.5. PERF 레코드를 사용하여 호출 그래프 데이터 캡처 208
18.6. PERF 보고서를 사용하여 PERF.DATA 분석 209
18.7. PERF 보고서 출력 해석 210
18.8. 다른 장치에서 읽을 수 있는 PERF.DATA 파일 생성 210
18.9. 다른 장치에서 생성된 PERF.DATA 파일 분석 212
18.10. PERF가 일부 함수 이름을 원시 함수 주소로 표시하는 이유 213
18.11. 디버그 및 소스 리포지토리 활성화 213
18.12. GDB를 사용하여 애플리케이션 또는 라이브러리를 위한 DEBUGINFO 패키지 가져오기 213

19장. PERF로 사용 중인 CPU 조사 216
19.1. PERF 통계로 계산된 CPU 이벤트 표시 216

19.2. PERF 보고서에서 가져온 CPU 샘플 표시 216
19.3. PERF를 사용하여 프로파일링하는 동안 특정 CPU 표시 217
19.4. PERF 레코드 및 PERF 보고서를 사용하여 특정 CPU 모니터링 218

20장. PERF로 애플리케이션 성능 모니터링 220
20.1. 실행 중인 프로세스에 PERF 레코드 연결 220
20.2. PERF 레코드를 사용하여 호출 그래프 데이터 캡처 220
20.3. PERF 보고서를 사용하여 PERF.DATA 분석 221

21장. PERF를 사용하여 UPROBES 생성 223
21.1. PERF를 사용하여 함수 수준에서 UPROBES 생성 223
21.2. PERF를 사용하여 함수 내에서 줄에 UPROBES 생성 223
21.3. UPROBES에서 기록된 데이터의 PERF 스크립트 출력 225

22장. PERF MEM를 사용하여 메모리 액세스 프로파일링 226
22.1. PERF MEM의 목적 226
22.2. PERF MEM를 사용하여 샘플링 메모리 액세스 226
22.3. PERF MEM 보고서 출력 해석 228

23장. 잘못된 공유 감지 230
23.1. PERF C2C의 목적 230
23.2. PERF C2C를 사용하여 캐시 라인 경합 감지 230
23.3. PERF C2C 레코드로 기록된 PERF.DATA 파일 시각화 231
23.4. PERF C2C 보고서 출력의 해석 234
23.5. PERF C2C로 잘못된 공유 탐지 235

24장. FIREAMEGRAPHS 시작하기 238
24.1. FIREAMEGRAPHS 설치 238
24.2. 전체 시스템에 불AMEGRAPHS 생성 238
24.3. 특정 프로세스에 대한 불AMEGRAPHS 생성 239
24.4. 불AMEGRAPHS 해석 241

25장. PERF 순환 버퍼를 사용하여 성능 병목 현상을 위한 프로세스 모니터링 243
25.1. PERF를 사용하여 순환 버퍼 및 이벤트별 스냅샷 243
25.2. PERF CIRCULAR BUFFERS를 사용하여 성능 병목 현상을 모니터링하기 위해 특정 데이터를 수집합니다. 243

26장. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에서 추적 지점 추가 및 제거 245
26.1. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에 추적 지점 추가 245
26.2. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에서 추적 지점 제거 246

27장. NUMASTAT로 메모리 할당 프로파일링 248
27.1. 기본 NUMASTAT 통계 248
27.2. NUMASTAT로 메모리 할당 보기 249

28장. CPU 사용률을 최적화하도록 운영 체제 구성 250
28.1. 프로세서 문제를 모니터링 및 진단하기 위한 툴 250
28.2. 시스템 토폴로지 유형 251
28.2.1. 시스템 토폴로지 표시 252
28.3. 커널 틱 시간 구성 254
28.4. 인터럽트 요청 개요 256
28.4.1. 인터럽트 수동 밸런싱 257
28.4.2. smp_affinity mask 설정 258

29장. 스케줄링 정책 튜닝 260
29.1. 스케줄링 정책의 범주 260

29.2. NFSNOBODY_DPDK를 사용한 정적 우선 순위 예약 261
29.3. DASD_RR을 사용한 라운드 로빈 우선순위 스케줄링 261
29.4. NFSNOBODY_OTHER를 사용한 일반 스케줄링 262
29.5. 스케줄러 정책 설정 262
29.6. CHRT 명령의 정책 옵션 264
29.7. 부팅 프로세스 중 서비스 우선 순위 변경 264
29.8. 우선순위 맵 266
29.9. TUNED CPU-PARTITIONING 프로필 267
29.10. 대기 시간이 짧은 튜닝을 위해 TUNED CPU-PARTITIONING 프로파일 사용 268
29.11. CPU-PARTITIONING TUNED 프로파일 사용자 정의 270

30장. I/O 및 파일 시스템 성능에 영향을 미치는 요소 271
30.1. I/O 및 파일 시스템 문제를 모니터링 및 진단하기 위한 툴 272
30.2. 파일 시스템 포맷에 사용 가능한 튜닝 옵션 274
30.3. 파일 시스템 마운트에 사용 가능한 튜닝 옵션 276
30.4. 사용되지 않는 블록 삭제 유형 277
30.5. 솔리드 스테이트 디스크 튜닝 고려 사항 278
30.6. 일반 블록 장치 튜닝 매개변수 279

31장. SYSTEMD를 사용하여 애플리케이션에서 사용하는 리소스 관리 281
31.1. SYSTEMD를 사용하여 시스템 리소스 할당 281
31.2. 리소스 관리에서 SYSTEMD의 역할 283
31.3. CGROUPS의 SYSTEMD 계층 구조 개요 283
31.4. SYSTEMD 단위 나열 285
31.5. SYSTEMD 제어 그룹 계층 구조 보기 287
31.6. 프로세스 CGROUPS 보기 289
31.7. 리소스 사용 모니터링 290
31.8. SYSTEMD 장치 파일을 사용하여 애플리케이션 제한 설정 291
31.9. SYSTEMCTL 명령을 사용하여 애플리케이션 제한 설정 292
31.10. 관리자 구성을 통해 글로벌 기본 CPU 선호도 설정 294
31.11. SYSTEMD를 사용하여 NUMA 정책 구성 294
31.12. SYSTEMD에 대한 NUMA 정책 구성 옵션 296
31.13. SYSTEMD-RUN 명령을 사용하여 일시적인 CGROUPS 생성 297
31.14. 일시적인 제어 그룹 제거 298

32장. CGROUPS 이해 300
32.1. 제어 그룹 이해 300
32.2. 커널 리소스 컨트롤러란? 301
32.3. 네임스페이스란? 303

33장. ZSWAP으로 시스템 성능 개선 305
33.1. ZSWAP이란 무엇입니까? 305
33.2. 런타임 시 ZSWAP 활성화 305
33.3. ZSWAP을 영구적으로 활성화 306

34장. CGROUP을 사용하여 CGROUP 수동 관리 308
34.1. CGROUP 생성 및 CGROUPS-V2 파일 시스템에서 컨트롤러 활성화 308
34.2. CPU 가중치를 조정하여 애플리케이션의 CPU 시간 분배 제어 311
34.3. CGROUPS-V1 마운트 314
34.4. CGROUPS-V1을 사용하여 애플리케이션에 CPU 제한 설정 317

35장. BPF COMPILER COLLECTION을 사용하여 시스템 성능 분석 322
35.1. BCC 소개 322
35.2. BCC-TOOLS 패키지 설치 322
35.3. 성능 분석에 선택된 BCC-TOOLS 사용 323

execsnoop를 사용하여 시스템 프로세스 검사 323
opennoop를 사용하여 명령에서 열리는 파일을 추적할 수 있습니다. 324
BIOStop를 사용하여 디스크의 I/O 작업 검사 326
xfsslower를 사용하여 예기치 않게 느린 파일 시스템 작업 노출 327

36장. 대규모 페이지 구성 330
36.1. 사용 가능한 대규모 페이지 기능 330
36.2. 부팅 시 HUGETLB 페이지 예약 매개변수 331
36.3. 부팅 시 HUGETLB 구성 332
36.4. 런타임 시 HUGETLB 페이지 예약 매개변수 334
36.5. 런타임에 HUGETLB 구성 335
36.6. 투명 HUGEPAGES 활성화 336
36.7. 투명 HUGEPAGES 비활성화 337
36.8. 번역 버퍼 크기에 대한 페이지 크기의 영향 338

37장. SYSTEMTAP 시작하기 339
37.1. SYSTEMTAP의 목적 339
37.2. SYSTEMTAP 설치 339
37.3. SYSTEMTAP을 실행하는 권한 341
37.4. SYSTEMTAP 스크립트 실행 341

38장. SYSTEMTAP의 교차 계측 343
38.1. SYSTEMTAP 교차 계측 343
38.2. SYSTEMTAP의 교차 계측 초기화 344

39장. SYSTEMTAP으로 네트워크 활동 모니터링 347
39.1. SYSTEMTAP을 사용한 네트워크 활동 프로파일링 347
39.2. SYSTEMTAP을 사용하여 네트워크 소켓 코드에서 호출되는 함수 추적 348
39.3. SYSTEMTAP으로 네트워크 패킷 삭제 모니터링 349

40장. SYSTEMTAP을 사용하여 커널 활동 프로파일링 351
40.1. SYSTEMTAP으로 함수 호출 계산 351
40.2. SYSTEMTAP을 사용하여 함수 호출 추적 352
40.3. SYSTEMTAP을 사용하여 커널 및 사용자 공간에 소요되는 시간 확인 354
40.4. SYSTEMTAP으로 폴링 애플리케이션 모니터링 355
40.5. SYSTEMTAP으로 가장 자주 사용되는 시스템 호출 추적 356
40.6. SYSTEMTAP으로 프로세스당 시스템 호출 볼륨 추적 357

41장. SYSTEMTAP을 사용하여 디스크 및 I/O 활동 모니터링 359
41.1. SYSTEMTAP으로 디스크 읽기/쓰기 트래픽 요약 359
41.2. SYSTEMTAP으로 읽기 또는 쓰기 각 파일의 I/O 시간 추적 360
41.3. SYSTEMTAP을 사용하여 누적 I/O 추적 361
41.4. SYSTEMTAP을 사용하여 특정 장치에서 I/O 활동 모니터링 362
41.5. SYSTEMTAP으로 파일 읽기 및 쓰기 모니터링 363

보다 포괄적 수용을 위한 오픈 소스 용어 교체

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

RED HAT 문서에 관한 피드백 제공

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

- 특정 문구에 대한 간단한 의견 작성 방법은 다음과 같습니다.
  1. 문서가 *Multi-page HTML* 형식으로 표시되는지 확인합니다. 또한 문서 오른쪽 상단에 **피드백** 버튼이 있는지 확인합니다.
  2. 마우스 커서를 사용하여 주석 처리하려는 텍스트 부분을 강조 표시합니다.
  3. 강조 표시된 텍스트 아래에 표시되는 **피드백 추가** 팝업을 클릭합니다.
  4. 표시된 지침을 따릅니다.
- Bugzilla를 통해 피드백을 제출하려면 새 티켓을 생성합니다.
  1. Bugzilla 웹 사이트로 이동합니다.
  2. 구성 요소로 **문서**를 사용합니다.
  3. **설명** 필드에 문서 개선을 위한 제안 사항을 기입하십시오. 관련된 문서의 해당 부분 링크를 알려주십시오.
  4. **버그 제출**을 클릭합니다.

1장. TUNED 시작하기

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

1.1. TUNED의 목적

tuned 는 시스템을 모니터링하고 특정 워크로드에서 성능을 최적화하는 서비스입니다. TuneD 의 코어는 다른 사용 사례에 맞게 시스템을 조정하는 *프로필* 입니다.

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

- 높은 처리량
- 짧은 대기 시간
- 전원 저장

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

또한 장치 사용 변경에 대응하고 활성 장치의 성능을 개선하고 비활성 장치의 전력 소비를 줄이기 위해 설정을 조정할 수도 있습니다.

1.2. TUNED 프로필

시스템의 상세한 분석은 매우 오래 걸릴 수 있습니다. tuned 는 일반적인 사용 사례에 대해 사전 정의된 여러 프로필을 제공합니다. 프로필을 생성, 수정 및 삭제할 수도 있습니다.

TuneD 로 제공되는 프로필은 다음 범주로 나뉩니다.

- **절전 프로필**
- **performance-boosting 프로필**

performance-boosting 프로필에는 다음과 같은 측면에 중점을 둔 프로필이 포함됩니다.

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

프로필 구성의 구문
tuned.conf 파일에는 하나의 [main] 섹션과 플러그인 인스턴스를 구성하기 위한 다른 섹션을 포함할 수 있습니다. 그러나 모든 섹션은 선택 사항입니다.

해시 기호(#)로 시작하는 행은 주석입니다.

추가 리소스

- **tuned.conf(5)** man page.

1.3. 기본 TUNED 프로필

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

환경	기본 프로필	목표
컴퓨팅 노드	**throughput-performance**	최대 처리량 성능
가상 머신	**virtual-guest**	최상의 성능. 최상의 성능에 관심이 없는 경우 **균형 잡힌** 또는 **절전** 프로필로 변경할 수 있습니다.
기타 사례	**balanced**	균형 있는 성능 및 전력 소비

추가 리소스

- **tuned.conf(5)** man page.

1.4. 병합된 TUNED 프로필

실험적 기능으로는 한 번에 더 많은 프로필을 선택할 수 있습니다. tuned는 로드 중에 병합하려고 합니다.

충돌이 발생하면 마지막으로 지정된 프로필의 설정이 우선합니다.

예 1.1. 가상 게스트의 전력 소비 감소

다음 예제에서는 가상 머신에서 실행하도록 시스템을 최적화하여 최상의 성능을 발휘하고 전력 소비가 낮고 전력 소비가 낮을 때 동시에 조정할 수 있습니다.

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

주의

결과 매개변수 조합이 의미가 있는지 확인하지 않고 병합이 자동으로 수행됩니다. 결과적으로 이 기능은 일부 매개 변수를 반대로 튜닝할 수 있습니다. 예를 들어, counterproductive일 수 있습니다. 예를 들어 throughput-performance 프로필을 사용하여 높은 처리량에 대해 디스크를 설정하고, 디스크 회전을 spindown-disk 프로필을 통해 낮은 값으로 디스크를 설정할 수 있습니다.

추가 리소스

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

1.5. TUNED 프로필의 위치

tuned는 프로필을 다음 디렉터리에 저장합니다.

/usr/lib/tuned/

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

/etc/tuned/

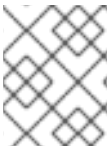
프로필을 사용자 지정해야 하는 경우 사용자 지정 프로필에 사용되는 디렉터리에 프로필 디렉터리를 복사합니다. 동일한 이름의 프로필이 두 개 있는 경우 **/etc/tuned/** 에 있는 사용자 지정 프로필이 사용됩니다.

추가 리소스

- **tuned.conf(5)** man page.

1.6. RHEL을 사용하여 배포된 TUNED 프로필

다음은 Red Hat Enterprise Linux에서 TuneD 와 함께 설치된 프로필 목록입니다.



참고

제품별 또는 타사 TuneD 프로필이 더 있을 수 있습니다. 이러한 프로필은 일반적으로 별도의 RPM 패키지로 제공됩니다.

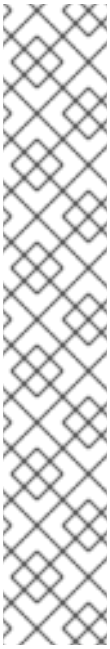
balanced

기본 power- saving 프로필. 이는 성능과 전력 소비 간의 절충을 위한 것입니다. 가능한 경우 자동 확장 및 자동 튜닝을 사용합니다. 유일한 단점은 대기 시간이 증가하는 것입니다. 현재 TuneD 릴리스에서는 CPU, 디스크, 오디오 및 비디오 플러그인을 활성화하고 보존 CPU governor를 활성화합니다. **radeon_powersave** 옵션은 지원되는 경우**dpm-balanced** 값을 사용합니다. 그러지 않으면**auto** 로 설정됩니다.
energy_performance_preference 특성을 일반적인 에너지 설정으로 변경합니다. 또한 **scaling_governor** policy 속성을보존 하거나 CPU governor로 변경합니다.

powersave

최대 성능 절약을 위한 프로필입니다. 실제 전력 소비를 최소화하기 위해 성능을 제한할 수 있습니다. 현재 **TuneD** 릴리스에서는 **SATA** 호스트 어댑터의 **USB** 자동 일시 중지, **WiFi** 전원 저장 및 **Aggressive Link Power Management (ALPM)** 전원 절약 기능을 사용할 수 있습니다. 또한 정전 비율이 낮은 시스템의 멀티 코어 전력 절감을 예약하고 온디맨드 **governor**를 활성화합니다. 이는 **AC97** 오디오 전력 절약을 가능하게 하거나, 시스템에 따라 **10**초의 시간 초과로**HDA-Intel**의 전력 절약을 가능하게 합니다. 시스템에 **KMS**가 활성화된 지원되는 **Radeon** 그래픽 카드가 포함된 경우 프로필은 자동 전원 저장으로 구성합니다. **ASUS Eee PC**에서는 동적 슈퍼 하이브리드 엔진을 사용할 수 있습니다.

energy_performance_preference 속성을 **powersave** 또는 **power energy setting**로 변경합니다. 또한 **scaling_governor** 정책 속성을 온 디맨드 또는 절전 **CPU governor**로 변경합니다.



참고

경우에 따라 **balanced** 프로파일은 **powersave** 프로필에 비해 더 효율적입니다.

예를 들어 트랜스코딩이 필요한 비디오 파일 등 수행해야 하는 정의된 작업 양을 고려하십시오. 작업이 빠르게 완료되기 때문에 트랜스코딩이 전체 전원에서 수행되면 컴퓨터가 더 적은 전력을 소비할 수 있으며 머신이 유휴 상태로 시작되고 매우 효율적인 전력 절약 모드로 자동 전환될 수 있습니다. 반면, 스로틀링된 시스템으로 파일을 트랜스코딩하는 경우 시스템은 트랜스코딩 중에 전력을 적게 소비하지만 프로세스는 더 오래 걸리고 전반적으로 소비된 전력이 더 높을 수 있습니다.

따라서 균형 잡힌 프로필은 일반적으로 더 나은 옵션이 될 수 있습니다.

throughput-performance

높은 처리량에 최적화된 서버 프로필입니다. 전원 절감 메커니즘을 비활성화하고 디스크 및 네트워크 IO의 처리량 성능을 향상시키는 sysctl 설정을 활성화합니다. CPU governor가 성능 으로 설정됩니다.

energy_performance_preference 및 scaling_governor 속성을 성능 프로필로 변경합니다.

accelerator-performance

액셀러레이터-성능 프로파일에 는 throughput-performance 프로필과 동일한 튜닝이 포함되어 있습니다. 또한 CPU가 낮은 C 상태로 잠금되므로 대기 시간이 100us 미만입니다. 이를 통해 GPU와 같은 특정 가속기의 성능이 향상됩니다.

latency-performance

짧은 대기 시간을 위해 최적화된 서버 프로필입니다. 전원 절감 메커니즘을 비활성화하고 대기 시간을 개선하는 sysctl 설정을 활성화합니다. CPU governor가 성능 으로 설정되어 CPU가 낮은 C 상태(PM QoS에 의해)에 고정됩니다.

energy_performance_preference 및 scaling_governor 속성을 성능 프로필로 변경합니다.

network-latency

대기 시간이 짧은 네트워크 튜닝을 위한 프로필입니다. latency-performance 프로필을 기반으로 합니다. 또한 투명한 대규모 페이지 및 NUMA 분산을 비활성화하고 다른 여러 네트워크 관련 sysctl 매개변수를 튜닝합니다.

latency-performance 프로필을 상속하여 energy_performance_preference 및

scaling_governor 속성을 성능 프로필로 변경합니다.

hpc-compute

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

network-throughput

처리량 네트워크 튜닝을 위한 프로필입니다. **throughput-performance** 프로필을 기반으로 합니다. 커널 네트워크 버퍼를 추가로 늘립니다.

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

virtual-guest

Red Hat Enterprise Linux 9 가상 시스템용으로 설계된 프로필과 **throughput-performance** 프로필 중 다른 작업 중 하나를 기반으로 하는 **VMWare** 게스트로, 가상 메모리 스왑**piness** 감소 및 디스크 **readahead** 값이 증가합니다. 디스크 장벽을 비활성화하지 않습니다.

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

virtual-host

throughput-performance 프로필을 기반으로 하는 가상 호스트를 위해 설계된 프로필로, 다른 작업 중에 가상 메모리 스왑**piness**를 줄이며, 디스크 **readahead** 값을 늘리며, 더 공격적인 더티 페이지 쓰기 값을 활성화합니다.

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

Oracle

Oracle 데이터베이스에 최적화된 프로필은 **throughput-performance** 프로필을 기반으로 로드됩니다. 또한 투명한 대규모 페이지를 비활성화하고 기타 성능 관련 커널 매개변수를 수정합니다. 이 프로필은 **tuned-profiles-oracle** 패키지에서 제공합니다.

desktop

balanced 프로필을 기반으로 하는 데스크탑에 최적화된 프로필입니다. 또한 스케줄러 자동 그룹을 활성화하여 대화형 애플리케이션의 대응을 개선할 수 있습니다.

optimize-serial-console

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

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

mssql

Microsoft SQL Server에 제공된 프로필입니다. 이는 **thoguhput-performance** 프로필을 기반으로 합니다.

intel-sst

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

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

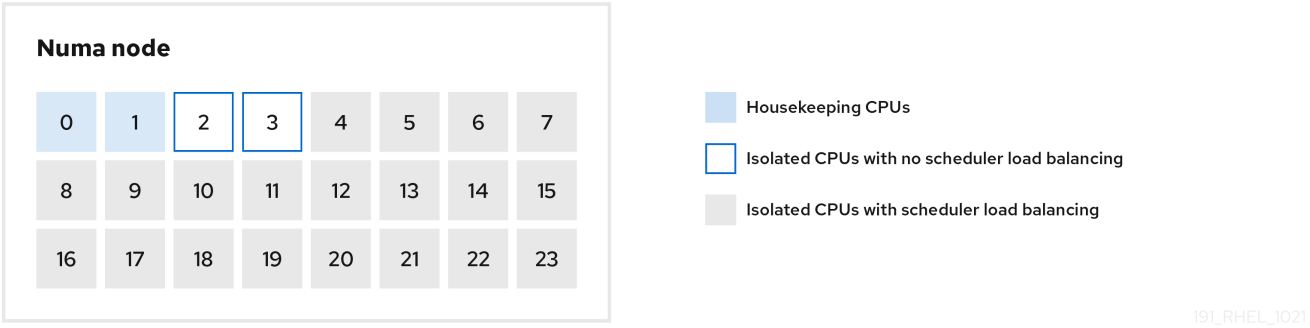
1.7. TUNED CPU-PARTITIONING 프로필

대기 시간에 민감한 워크로드에 맞게 **Red Hat Enterprise Linux 9**를 튜닝하려면 **cpu-partitioning TuneD** 프로필을 사용하는 것이 좋습니다.

Red Hat Enterprise Linux 9 이전에는 대기 시간이 짧은 **Red Hat** 문서에서는 대기 시간이 짧은 튜닝을 수행하는 데 필요한 수많은 낮은 수준의 단계를 설명했습니다. **Red Hat Enterprise Linux 9**에서는 **cpu-partitioning TuneD** 프로필을 사용하여 대기 시간이 짧은 튜닝을 보다 효율적으로 수행할 수 있습니다. 이 프로필은 대기 시간이 짧은 개별 애플리케이션의 요구 사항에 따라 쉽게 사용자 지정할 수 있습니다.

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

그림 1.1. figure cpu-partitioning



다음 구성 옵션을 사용하여 **/etc/tuned/cpu-partitioning-ECDHEs.conf** 파일에서 **cpu-partitioning** 프로필을 구성할 수 있습니다.

로드 밸런싱이 있는 격리된 **CPU**

cpu-partitioning figure에서 **4**에서 **23**으로 번호가 매겨진 블록은 기본 분리된 **CPU**입니다. 커널 스케줄러의 프로세스 부하 분산이 이러한 **CPU**에서 활성화됩니다. 커널 스케줄러 부하 분산이 필요한 여러 스레드가 있는 대기 시간이 짧은 프로세스를 위해 설계되었습니다.

kernel 스케줄러 로드 밸런싱을 사용할 **CPU**를 나열하는 **isolated_cores= cpu-list** 옵션을 사용하여 **/etc/tuned/cpu-partitions.conf** 파일에서 **cpu-partitioning** 프로필을 구성할 수 있습니다.

분리된 **CPU** 목록은 쉼표로 구분하거나 **dash(예:)**를 사용하여 범위를 지정할 수 있습니다. 이 옵션은 필수입니다. 이 목록에서 누락된 **CPU**는 자동으로 하우스키핑 **CPU**로 간주됩니다.

로드 밸런싱이 없는 격리된 **CPU**

cpu-partitioning 그림에서 번호가 지정된 블록 **2**와 **3**은 추가 커널 스케줄러 프로세스 부하 분산을 제공하지 않는 격리된 **CPU**입니다.

커널 스케줄러 로드 밸런싱을 사용하지 않는 **CPU**를 나열하는 **no_balance_cores= cpu-list** 옵션을 사용하여 **/etc/tuned/cpu-partitions.conf** 파일에서 **cpu-partitioning** 프로필을 구성할 수 있습니다.

no_balance_cores 옵션을 지정하는 것은 선택 사항이지만 이 목록의 모든 **CPU**는 **isolated_cores** 목록에 나열된 **CPU**의 하위 집합이어야 합니다.

이러한 **CPU**를 사용하는 애플리케이션 스레드를 각 **CPU**에 개별적으로 고정해야 합니다.

하우스키핑 **CPU**

cpu-partitioning-postgresqls.conf 파일에서 분리된 CPU는 자동으로 하우스키핑 CPU로 간주됩니다. 하우스키핑 CPU에서 모든 서비스, 데몬, 사용자 프로세스, 이동 가능한 커널 스레드, 인터럽트 처리기, 커널 타이머를 실행할 수 있습니다.

추가 리소스

- tuned-profiles-cpu-partitioning(7) man page

1.8. 대기 시간이 짧은 튜닝을 위해 TUNED CPU-PARTITIONING 프로파일 사용

이 프로세스에서는 TuneD의 cpu-partitioning 프로필을 사용하여 대기 시간이 짧은 시스템을 조정하는 방법을 설명합니다. cpu-partitioning 및 cpu-partitioning 수치에 언급된 CPU 레이아웃을 사용할 수 있는 대기 시간이 짧은 애플리케이션의 예를 사용합니다.

이 경우 애플리케이션은 다음을 사용합니다.

- 네트워크에서 데이터를 읽는 하나의 전용 리더 스레드가 CPU 2에 고정되어 있습니다.
- 이 네트워크 데이터를 처리하는 많은 수의 스레드가 CPU 4-23에 고정되어 있습니다.
- 처리된 데이터를 네트워크에 쓰는 전용 작성기 스레드는 CPU 3에 고정되어 있습니다.

사전 요구 사항

- dnf install tuned-profiles- cpu-partitioning 명령을 root로 사용하여 cpu-partitioning TuneD 프로필 을 설치했습니다.

절차

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

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

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

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

3. **reboot**

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

추가 리소스

- **tuned-profiles-cpu-partitioning(7) man page**

1.9. CPU-PARTITIONING TUNED 프로파일 사용자 정의

TuneD 프로필을 확장하여 추가 튜닝을 변경할 수 있습니다.

예를 들어 cpu-partitioning 프로필은 CPU가 cstate=1 을 사용하도록 설정합니다. cpu-partitioning 프로필을 사용하지만 CPU cstate를 cstate1에서 cstate0으로 추가로 변경하려면 다음 절차에서는 cpu-partitioning 프로필을 상속하고 C state-0을 설정하는 *my_profile* 이라는 새 TuneD 프로필을 설명합니다.

절차

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

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

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

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

3. 새 프로필을 사용합니다.

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

참고

공유 예에서는 재부팅이 필요하지 않습니다. 그러나 ***my_profile*** 프로필의 변경 사항을 적용하려면 재부팅을 수행한 다음 시스템을 재부팅합니다.

추가 리소스

- **tuned-profiles-cpu-partitioning(7) man page**

1.10. RHEL을 사용하여 실시간 TUNED 프로필 배포

실시간 프로필은 실시간 커널을 실행하는 시스템을 위한 것입니다. 특별한 커널 빌드가 없으면 시스템을 실시간으로 구성하지 않습니다. RHEL에서는 추가 리포지토리에서 프로필을 사용할 수 있습니다.

다음 실시간 프로필을 사용할 수 있습니다.

realtime

베어메탈 실시간 시스템에서 를 사용합니다.

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

realtime-virtual-host

실시간으로 구성된 가상화 호스트에서 를 사용합니다.

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

realtime-virtual-guest

실시간으로 구성된 가상화 게스트에서 를 사용합니다.

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

1.11. TUNED의 정적 및 동적 튜닝

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

정적 튜닝

주로 사전 정의된 **sysctl** 및 **sysfs** 설정 애플리케이션과 함께 **ethtool** 과 같은 여러 구성 툴의 일회성 활성화로 구성됩니다.

동적 튜닝

시스템의 가동 시간 동안 다양한 시스템 구성 요소가 사용되는 방식을 감시합니다. **tuned** 는 모니터링 정보를 기반으로 시스템 설정을 동적으로 조정합니다.

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

동적 튜닝은 기본적으로 비활성화되어 있습니다. 이를 활성화하려면 **/etc/tuned/tuned-main.conf** 파일을 편집하고 **dynamic_tuning** 옵션을 **1** 로 변경합니다. 그런 다음 **tuned**는 정기적으로 시스템 통계를 분석하고 이를 사용하여 시스템 튜닝 설정을 업데이트합니다. 이러한 업데이트 사이의 시간 간격을 초 단위로 구성하려면 **update_interval** 옵션을 사용합니다.

현재 동적 튜닝 알고리즘은 성능 및 절전의 균형을 유지하려고 시도하므로 성능 프로필에서 비활성화됩니다. 개별 플러그인에 대한 동적 튜닝은 **TuneD** 프로필에서 활성화하거나 비활성화할 수 있습니다.

> **예 1.2.** 워크스테이션의 정적 및 동적 튜닝
>
> 일반적인 사무실 워크스테이션에서 이더넷 네트워크 인터페이스는 대부분의 시간 동안 비활성화됩니다. 일부 이메일만 들어오거나 나가거나 일부 웹 페이지가 로드될 수 있습니다.
>
> 이러한 종류의 로드의 경우 네트워크 인터페이스는 기본적으로와 같이 항상 전체 속도로 실행할 필요가 없습니다. **tuned** 에는 이 낮은 활동을 감지한 다음 해당 인터페이스의 속도를 자동으로 낮추어 일반적으로 전력 사용량을 줄일 수 있는 네트워크 장치에 대한 모니터링 및 튜닝 플러그인이 있습니다.
>
> 인터페이스의 활동이 더 긴 기간 동안 증가하는 경우, 예를 들어 **DVD** 이미지가 다운로드되고 있는 이메일이나 큰 첨부 파일이 있는 이메일이 열리기 때문에, **TuneD** 는 이를 감지하고 활동 수준이 높은

동안 최상의 성능을 제공하기 위해 인터페이스 속도를 최대로 설정합니다.

이 원칙은 CPU 및 디스크의 다른 플러그인에도 사용됩니다.

1.12. TUNED NO-DAEMON 모드

no-daemon 모드에서 TuneD 를 실행하면 상주 메모리가 필요하지 않습니다. 이 모드에서 TuneD 는 설정을 적용하고 종료합니다.

이 모드에서는 다음을 포함하여 많은 TuneD 기능이 없기 때문에 기본적으로 no-daemon 모드가 비활성화되어 있습니다.

- D-Bus 지원
- 핫플러그 지원
- 설정 롤백 지원

no-daemon 모드를 활성화하려면 /etc/tuned/tuned-main.conf 파일에 다음 행을 포함합니다.

```
daemon = 0
```

1.13. TUNED 설치 및 활성화

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

절차

1. TuneD 패키지를 설치합니다.

```
# dnf install tuned
```

2. **TuneD 서비스를 활성화하고 시작합니다.**

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

3. **필요한 경우 실시간 시스템에 대한 TuneD 프로필을 설치합니다.**

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

4. **TuneD 프로필이 활성화되어 적용되는지 확인합니다.**

```
$ tuned-adm active

Current active profile: balanced
```

```
$ tuned-adm verify

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

1.14. 사용 가능한 TUNED 프로필 나열

다음 절차에서는 시스템에서 현재 사용 가능한 모든 TuneD 프로필을 나열합니다.

절차

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

```
$ tuned-adm list

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

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

- 현재 활성화된 프로필만 표시하려면 다음을 사용합니다.

```
$ tuned-adm active

Current active profile: balanced
```

추가 리소스

- **tuned-adm(8) man page.**

1.15. TUNED 프로필 설정

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

사전 요구 사항

- **TuneD** 서비스가 실행 중입니다. 자세한 내용은 **TuneD** 설치 및 활성화를 참조하십시오.

절차

1. 필요한 경우 **TuneD** 가 시스템에 가장 적합한 프로필을 권장하도록 할 수 있습니다.

```
# tuned-adm recommend

balanced
```

2. 프로필을 활성화합니다.

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

또는 여러 프로필의 조합을 활성화할 수도 있습니다.

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

예 1.3. 낮은 전력 사용을 위해 최적화된 가상 머신

다음 예제에서는 시스템을 최적화하여 최상의 성능을 갖춘 가상 시스템에서 실행하고 전력 소비가 낮은 전력을 위해 동시에 튜닝하는 것이 우선 순위입니다.

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

3. 시스템에서 현재 활성화된 **TuneD** 프로필을 확인합니다.

```
# tuned-adm active

Current active profile: selected-profile
```

4. 시스템을 재부팅합니다.

```
# reboot
```

검증 단계

- **TuneD** 프로필이 활성화되어 적용되는지 확인합니다.

```
$ tuned-adm verify

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

추가 리소스

- **tuned-adm(8) man page**

1.16. TUNED 비활성화

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

절차

- 모든 튜닝을 일시적으로 비활성화하려면 다음을 수행합니다.

```
# tuned-adm off
```

TuneD 서비스를 다시 시작한 후 튜닝이 다시 적용됩니다.

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

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

추가 리소스

- **tuned-adm(8) man page**

2장. TUNED 프로필 사용자 정의

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

사전 요구 사항

- 자세한 내용은 TuneD 설치 및 Enabling TuneD에 설명된 대로 TuneD 를 설치하고 활성화합니다.

2.1. TUNED 프로필

시스템의 상세한 분석은 매우 오래 걸릴 수 있습니다. tuned 는 일반적인 사용 사례에 대해 사전 정의된 여러 프로필을 제공합니다. 프로필을 생성, 수정 및 삭제할 수도 있습니다.

TuneD 로 제공되는 프로필은 다음 범주로 나뉩니다.

- 절전 프로필
- performance-boosting 프로필

performance-boosting 프로필에는 다음과 같은 측면에 중점을 둔 프로필이 포함됩니다.

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

프로필 구성의 구문

tuned.conf 파일에는 하나의 [main] 섹션과 플러그인 인스턴스를 구성하기 위한 다른 섹션을 포함할

수 있습니다. 그러나 모든 섹션은 선택 사항입니다.

해시 기호(#)로 시작하는 행은 주석입니다.

추가 리소스

- tuned.conf(5) man page.

2.2. 기본 TUNED 프로필

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

환경	기본 프로필	목표
컴퓨팅 노드	**throughput-performance**	최대 처리량 성능
가상 머신	**virtual-guest**	최상의 성능. 최상의 성능에 관심이 없는 경우 **균형 잡힌** 또는 **절전** 프로필로 변경할 수 있습니다.
기타 사례	**balanced**	균형 있는 성능 및 전력 소비

추가 리소스

- tuned.conf(5) man page.

2.3. 병합된 TUNED 프로필

실험적 기능으로는 한 번에 더 많은 프로필을 선택할 수 있습니다. tuned 는 로드 중에 병합하려고 합니다.

충돌이 발생하면 마지막으로 지정된 프로필의 설정이 우선합니다.

> 예 2.1. 가상 게스트의 전력 소비 감소
>
> 다음 예제에서는 가상 머신에서 실행하도록 시스템을 최적화하여 최상의 성능을 발휘하고 전력 소비가 낮고 전력 소비가 낮을 때 동시에 조정할 수 있습니다.

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

주의

결과 매개변수 조합이 의미가 있는지 확인하지 않고 병합이 자동으로 수행됩니다. 결과적으로 이 기능은 일부 매개 변수를 반대로 튜닝할 수 있습니다. 예를 들어, **counterproductive**일 수 있습니다. 예를 들어 **throughput-performance** 프로필을 사용하여 높은 처리량에 대해 디스크를 설정하고, 디스크 회전을 **spindown-disk** 프로필을 통해 낮은 값으로 디스크를 설정할 수 있습니다.

추가 리소스

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

2.4. TUNED 프로필의 위치

tuned 는 프로필을 다음 디렉터리에 저장합니다.

/usr/lib/tuned/

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

/etc/tuned/

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

추가 리소스

- **tuned.conf(5) man page.**

2.5. TUNED 프로필 간 상속

tuned 프로필은 다른 프로필을 기반으로 하며 해당 상위 프로필의 특정 측면만 수정할 수 있습니다.

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

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

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

*일부 매개변수만 조정된 **/usr/lib/tuned/** 의 사전 설치된 프로필을 기반으로 **/etc/tuned/** 디렉터리에 자체 하위 프로필을 생성할 수 있습니다.*

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

*예 **2.2.** 균형을 기반으로 한 전력 절감 프로필*

*다음은 균형 있는 프로필을 확장하고 모든 장치에 대한 **Aggressive Link Power Management (ALPM)**를 최대 전력으로 설정하는 사용자 정의 프로필의 예입니다.*

```
[main]
include=balanced

[scsi_host]
alpm=min_power
```

추가 리소스

- ***tuned.conf(5) man page***

2.6. TUNED의 정적 및 동적 튜닝

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

정적 튜닝

주로 사전 정의된 **sysctl** 및 **sysfs** 설정 애플리케이션과 함께 **ethtool** 과 같은 여러 구성 툴의 일회성 활성화로 구성됩니다.

동적 튜닝

시스템의 가동 시간 동안 다양한 시스템 구성 요소가 사용되는 방식을 감시합니다. **tuned** 는 모니터링 정보를 기반으로 시스템 설정을 동적으로 조정합니다.

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

동적 튜닝은 기본적으로 비활성화되어 있습니다. 이를 활성화하려면 **/etc/tuned/tuned-main.conf** 파일을 편집하고 **dynamic_tuning** 옵션을 **1** 로 변경합니다. 그런 다음 **tuned**는 정기적으로 시스템 통계를 분석하고 이를 사용하여 시스템 튜닝 설정을 업데이트합니다. 이러한 업데이트 사이의 시간 간격을 초 단위로 구성하려면 **update_interval** 옵션을 사용합니다.

현재 동적 튜닝 알고리즘은 성능 및 절전의 균형을 유지하려고 시도하므로 성능 프로필에서 비활성화됩니다. 개별 플러그인에 대한 동적 튜닝은 **TuneD** 프로필에서 활성화하거나 비활성화할 수 있습니다.

예 2.3. 워크스테이션의 정적 및 동적 튜닝

일반적인 사무실 워크스테이션에서 이더넷 네트워크 인터페이스는 대부분의 시간 동안 비활성화됩니다. 일부 이메일만 들어오거나 나가거나 일부 웹 페이지가 로드될 수 있습니다.

이러한 종류의 로드의 경우 네트워크 인터페이스는 기본적으로와 같이 항상 전체 속도로 실행할 필요가 없습니다. **tuned** 에는 이 낮은 활동을 감지한 다음 해당 인터페이스의 속도를 자동으로 낮추어 일반적으로 전력 사용량을 줄일 수 있는 네트워크 장치에 대한 모니터링 및 튜닝 플러그인이 있습니다.

인터페이스의 활동이 더 긴 기간 동안 증가하는 경우, 예를 들어 **DVD** 이미지가 다운로드되고 있는 이메일이나 큰 첨부 파일이 있는 이메일이 열리기 때문에, **TuneD** 는 이를 감지하고 활동 수준이 높은 동안 최상의 성능을 제공하기 위해 인터페이스 속도를 최대로 설정합니다.

이 원칙은 **CPU** 및 디스크의 다른 플러그인에도 사용됩니다.

2.7. TUNED 플러그인

*플러그인은 **TuneD** 프로필에서 시스템의 다른 장치를 모니터링하거나 최적화하기 위해 사용하는 모듈입니다.*

__tuned__ 는 다음 두 가지 유형의 플러그인을 사용합니다.

모니터링 플러그인

모니터링 플러그인은 실행 중인 시스템에서 정보를 가져오는 데 사용됩니다. 동적 튜닝을 위해 플러그인을 튜닝하는 경우 모니터링 플러그인 출력을 사용할 수 있습니다.

활성화된 튜닝 플러그인에서 메트릭이 필요할 때마다 모니터링 플러그인이 자동으로 인스턴스화됩니다. 두 개의 튜닝 플러그인에 동일한 데이터가 필요한 경우 모니터링 플러그인의 인스턴스 1개만 생성되고 데이터가 공유됩니다.

플러그인 튜닝

*각 튜닝 플러그인은 개별 하위 시스템을 튜닝하고 **TuneD** 프로필에서 채워지는 여러 매개변수를 사용합니다. 각 하위 시스템에는 튜닝 플러그인의 개별 인스턴스에서 처리하는 여러 **CPU** 또는 네트워크 카드와 같은 여러 장치가 있을 수 있습니다. 개별 장치의 특정 설정도 지원됩니다.*

***TuneD** 프로필의 플러그인 구문*

플러그인 인스턴스를 설명하는 섹션은 다음과 같이 포맷됩니다.

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

NAME

로그에 사용되는 플러그인 인스턴스의 이름입니다. 임의의 문자열이 될 수 있습니다.

유형

는 튜닝 플러그인 유형입니다.

DEVICES

이 플러그인 인스턴스에서 처리하는 장치 목록입니다.

devices** 행에는 목록, 와일드카드(*) 및 부정(!)이 포함될 수 있습니다. 장치 줄이 없는 경우 **TYPE** 시스템에 연결된 모든 장치가 플러그인 인스턴스에서 처리됩니다. **devices= 옵션을 사용하는 것과 동일합니다.*

예 2.4. 플러그인을 사용하여 블록 장치 일치

다음 예제는 **sda** 또는 **sdb** 와 같이 **sd** 로 시작하는 모든 블록 장치와 일치하며, 이러한 장치에 대한 장벽을 비활성화하지 않습니다.

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

다음 예제는 **sda1** 및 **sda2** 를 제외한 모든 블록 장치와 일치합니다.

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

플러그인 인스턴스를 지정하지 않으면 플러그인이 활성화되지 않습니다.

플러그인이 더 많은 옵션을 지원하는 경우 플러그인 섹션에서 지정할 수도 있습니다. 옵션을 지정하지 않고 이전에 포함된 플러그인에 지정하지 않은 경우 기본값이 사용됩니다.

짧은 플러그인 구문

플러그인 인스턴스에 사용자 정의 이름이 필요하지 않고 구성 파일에 인스턴스에 대한 정의가 하나만 있는 경우 **TuneD** 는 다음과 같은 짧은 구문을 지원합니다.

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

이 경우 **type** 행을 생략할 수 있습니다. 그런 다음 인스턴스를 **type**과 동일한 이름으로 참조합니다. 이전 예제는 다음과 같이 다시 작성할 수 있습니다.

예 2.5. 짧은 구문을 사용하여 블록 장치 일치

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

프로필에서 플러그인 정의 충돌

include 옵션을 사용하여 동일한 섹션을 두 번 이상 지정하면 설정이 병합됩니다. 충돌로 인해 병합할 수 없는 경우 마지막으로 충돌하는 정의가 이전 설정을 덮어씁니다. 이전에 정의한 내용을 모르는 경우 *replace* 부울 옵션을 사용하여 *true* 로 설정할 수 있습니다. 이로 인해 이름이 같은 이전 정의를 모두 덮어쓰고 병합이 발생하지 않습니다.

enabled=false 옵션을 지정하여 플러그인을 비활성화할 수도 있습니다. 이 방법은 인스턴스가 정의되지 않은 경우와 동일합니다. 플러그인을 비활성화하면 *include* 옵션에서 이전 정의를 다시 정의하고 사용자 정의 프로필에서 플러그인을 활성화하지 않으려면 유용합니다.

참고

tuned 에는 튜닝 프로파일 활성화 또는 비활성화의 일부로 쉘 명령을 실행하는 기능이 포함되어 있습니다. 이를 통해 아직 *TuneD* 에 통합되지 않은 기능을 사용하여 *TuneD* 프로필을 확장할 수 있습니다.

script 플러그인을 사용하여 임의의 쉘 명령을 지정할 수 있습니다.

추가 리소스

- *tuned.conf(5) man page*

2.8. 사용 가능한 TUNED 플러그인

이 섹션에는 현재 TuneD 에서 사용 가능한 모든 모니터링 및 튜닝 플러그인이 나열됩니다.

모니터링 플러그인

현재 다음과 같은 모니터링 플러그인이 구현되어 있습니다.

disk

장치 및 측정 간격당 디스크 로드(IO 작업 수)를 가져옵니다.

net

네트워크 카드 및 측정 간격당 네트워크 로드(전송된 패킷 수)를 가져옵니다.

load

CPU 및 측정 간격당 CPU 부하를 가져옵니다.

플러그인 튜닝

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

cpu

CPU governor 옵션을 통해 지정된 값으로 CPU governor 를 설정하고 CPU 로드에 따라 PM QoS(Power Management Quality of Service) CPU Direct Memory Access(DMA) 대기 시간을 동적으로 변경합니다.

CPU 로드가 load_threshold 옵션에 지정된 값보다 작으면 대기 시간이 latency_high 옵션에 지정된 값으로 설정되고, 그렇지 않으면 latency_low 에 의해 지정된 값으로 설정됩니다.

또한 대기 시간을 특정 값으로 강제 적용하고 동적으로 변경되는 것을 방지할 수 있습니다. 이렇게 하려면 force_latency 옵션을 필요한 대기 시간으로 설정합니다.

eeepc_she

CPU 로드에 따라 프런트 사이드 버스(FSB) 속도를 동적으로 설정합니다.

이 기능은 일부 netbook에서 찾을 수 있으며 ASUS Super Hybrid Engine (SHE)이라고도합니다.

CPU 로드가 load_threshold_powersave 옵션에 지정된 값보다 작거나 같은 경우 플러그인은 FSB 속도를 she_powersave 옵션에 지정된 값으로 설정합니다. CPU 로드가 load_threshold_normal 옵션에 지정된 값과 크거나 같은 경우 FSB 속도를 she_normal 옵션에 지정된 값으로 설정합니다.

정적 튜닝이 지원되지 않으며 TuneD 에서 이 기능에 대한 하드웨어 지원을 감지하지 않으면 플러그인이 투명하게 비활성화됩니다.

net

Wake-on-LAN 기능을 wake_on_lan 옵션에 지정된 값으로 구성합니다. ethtool 유틸리티와 동일한 구문을 사용합니다. 또한 인터페이스 사용률에 따라 인터페이스 속도를 동적으로 변경합니다.

sysctl

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

*구문은 **name=value** 이며, 여기서 **name** 은 **sysctl** 유틸리티에서 제공하는 이름과 동일합니다.*

*__TuneD__ 에서 사용할 수 있는 다른 플러그인에서 다루지 않는 시스템 설정을 변경해야 하는 경우 **sysctl** 플러그인을 사용합니다. 일부 특정 플러그인에서 설정이 적용되는 경우 이러한 플러그인을 선호합니다.*

usb

*__USB__ 장치의 자동 일시 중지 시간 제한을 **autosuspend** 매개 변수에서 지정한 값으로 설정합니다.*

*값 **0** 은 자동 일시 중지가 비활성화되어 있음을 의미합니다.*

vm

__transparent_hugepages__ 옵션의 값에 따라 투명한 대규모 페이지를 활성화하거나 비활성화합니다.

__transparent_hugepages__ 옵션의 유효한 값은 다음과 같습니다.

- *"always"*
- *"never"*
- *"madvise"*

audio

*시간 제한 옵션으로 지정된 값으로 오디오 **codecs**의 자동 대기 시간 제한을 설정합니다.*

*현재 **snd_hda_intel** 및 **snd_ac97_codec codecs**가 지원됩니다. 값 **0** 은 자동 일시 중지가 비활성화되어 있음을 의미합니다. 부울 옵션 **reset_controller** 를 **true** 로 설정하여 컨트롤러 재설정을 적용할 수도 있습니다.*

disk

디스크 로켓을 엘리베이터 옵션에 지정된 값으로 설정합니다.

또한 다음을 설정합니다.

- **APM** 옵션을 통해 지정한 값으로 설정
- 스케줄러 **_quantum** 옵션에 의해 지정된 값에 대한 스케줄러 **full**
- 회전 옵션에서 지정한 값에 대한 디스크 회전 시간 초과
- **readahead** 매개변수로 지정된 값에 대한 디스크 **readahead**
- **current disk readahead to a value multiplied by the constant specified by the readahead_multiply** 옵션

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

scsi_host

SCSI 호스트의 옵션을 조정합니다.

Aggressive Link Power Management (ALPM)를 **alpm** 옵션에 지정된 값으로 설정합니다.

mounts

disable_barriers 옵션의 부울 값에 따라 마운트 장벽을 활성화하거나 비활성화합니다.

script

프로필이 로드되거나 언로드될 때 외부 스크립트 또는 바이너리를 실행합니다. 임의의 실행 파일을 선택할 수 있습니다.

중요

스크립트 플러그인은 주로 이전 릴리스와의 호환성을 위해 제공됩니다. 필요한 기능을 포함하는 경우 다른 **TuneD** 플러그인을 선호하십시오.

tuned 는 다음 인수 중 하나를 사용하여 실행 파일을 호출합니다.

- 프로필을 로드할 때 시작
- 프로필을 언로드할 때 중지

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

Bash 스크립트는 **/usr/lib/tuned/functions Bash** 라이브러리를 가져오고 여기에 정의된 함수를 사용할 수 있습니다. 이러한 기능은 **TuneD** 에서 기본적으로 제공하지 않는 기능에만 사용하십시오. 함수 이름이 **_wifi_set_power_level** 과 같은 밑줄로 시작하는 경우 함수를 비공개로 간주하고 나중에 변경될 수 있으므로 스크립트에서 이 함수를 사용하지 마십시오.

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

예 2.6. 프로필에서 Bash 스크립트 실행

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

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

sysfs

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

구문은 **name=value** 이며, 여기서 **name** 은 사용할 **sysfs** 경로입니다.

다른 플러그인에서 다루지 않는 일부 설정을 변경해야 하는 경우 이 플러그인을 사용하십시오. 필요한 설정을 포함하는 경우 특정 플러그인을 선호합니다.

video

비디오 카드에 다양한 전원 저장 수준을 설정합니다. 현재 **Radeon** 카드만 지원됩니다.

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

- **default**
- **auto**
- 낮음 **(LOW)**
- **mid**
- 높음
- **dynpm**
- **dpm-battery**
- **dpm-balanced**
- **dpm-perfomance**

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

bootloader

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

__GRUB 2__ 설정 파일의 사용자 지정 비표준 위치는 __grub2_cfg_file__ 옵션으로 지정할 수 있습니다.

커널 옵션이 현재 __GRUB__ 설정 및 해당 템플릿에 추가됩니다. 커널 옵션을 적용하려면 시스템을 재부팅해야 합니다.

다른 프로필로 전환하거나 __TuneD__ 서비스를 수동으로 중지하면 추가 옵션이 제거됩니다. 시스템을 종료하거나 재부팅하는 경우 커널 옵션은 __grub.cfg__ 파일에 유지됩니다.

커널 옵션은 다음 구문으로 지정할 수 있습니다.

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

__예 2.7.__ 커널 명령줄 수정

예를 들어, __quiet__ 커널 옵션을 __TuneD__ 프로필에 추가하려면 __tuned.conf__ 파일에 다음 행을 포함합니다.

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

다음은 __isolcpus=2__ 옵션을 커널 명령줄에 추가하는 사용자 정의 프로필의 예입니다.

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

2.9. TUNED 프로필의 변수

__TuneD__ 프로필이 활성화되면 런타임에 변수가 확장됩니다.

__TuneD__ 변수를 사용하면 __TuneD__ 프로필에서 필요한 입력 수를 줄일 수 있습니다.

__TuneD__ 프로파일에는 사전 정의된 변수가 없습니다. 프로필에서 __[Variables]__ 섹션을 생성하고 다음 구

문을 사용하여 고유한 변수 를 정의할 수 있습니다.

```
[variables]

variable_name=value
```

프로필의 변수 값을 확장하려면 다음 구문을 사용합니다.

```
${variable_name}
```

예 2.8. 변수를 사용하여 CPU 코어 분리

다음 예에서 **${isolated_cores}** 변수는 **12**로 확장됩니다. 따라서 커널은 **isolcpus=1,2** 옵션으로 부팅됩니다.

```
[variables]
isolated_cores=1,2

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

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

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

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

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

추가 리소스

- **tuned.conf(5) man page**

2.10. TUNED 프로파일의 기본 제공 함수

기본 제공 함수는 **TuneD** 프로필이 활성화되면 런타임에 확장됩니다.

다음을 수행할 수 있습니다.

- **TuneD** 변수와 함께 다양한 기본 제공 함수 사용
- **Python**에서 사용자 정의 함수를 생성하고 플러그인 형태로 **TuneD** 에 추가합니다.

함수를 호출하려면 다음 구문을 사용합니다.

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

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

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

예 2.9. 변수 및 기본 제공 함수를 사용하여 **CPU** 코어 분리

다음 예에서 **${non_isolated_cores}** 변수는 **03-5** 로 확장되고 **cpulist_invert** 기본-**invert** 함수는 **0,3-5** 인수를 사용하여 호출됩니다.

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

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

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

추가 리소스

- **tuned.conf(5) man page**

2.11. TUNED 프로필에서 사용할 수 있는 기본 제공 함수

다음 기본 제공 함수는 모든 TuneD 프로필에서 사용할 수 있습니다.

PROFILE_DIR

프로필과 tuned.conf 파일이 있는 디렉터리 경로를 반환합니다.

exec

프로세스를 실행하고 출력을 반환합니다.

assertion

두 개의 인수를 비교합니다. 이 값이 일치하지 않으면 함수에서 첫 번째 인수의 텍스트를 기록하고 프로필 로드를 중단합니다.

assertion_non_equal

두 개의 인수를 비교합니다. 이 값이 일치하면 함수는 첫 번째 인수의 텍스트를 기록하고 프로필 로드를 중단합니다.

kb2s

킬로바이트를 디스크 섹터로 변환합니다.

s2kb

디스크 섹터를 킬로바이트로 변환합니다.

strip

전달된 모든 인수에서 문자열을 만들고 선행 및 후행 공백을 모두 삭제합니다.

virt_check

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

- VM 내에서 함수는 첫 번째 인수를 반환합니다.
- 베어 메탈에서 함수는 오류가 발생하는 경우에도 두 번째 인수를 반환합니다.

cpulist_invert

CPU 목록을 반전하여 보완합니다. 예를 들어 CPU가 4개인 시스템에서는 0에서 3으로 번호가 매겨지면 목록 0,2,3 의 버전이 1 입니다.

cpulist2hex

CPU 목록을 16진수 CPU 마스크로 변환합니다.

cpulist2hex_invert

CPU 목록을 16진수 CPU 마스크로 변환하고 반전합니다.

hex2cpulist

16진수 CPU 마스크를 CPU 목록으로 변환합니다.

cpulist_online

목록의 CPU가 온라인 상태인지 확인합니다. 온라인 CPU만 포함하는 목록을 반환합니다.

cpulist_present

목록의 CPU가 있는지 확인합니다. 현재 CPU만 포함하는 목록을 반환합니다.

cpulist_unpack

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

cpulist_pack

1,2,3,5 에서 1-3,5 형식의 CPU 목록을 포함합니다.

2.12. 새 TUNED 프로필 생성

이 절차에서는 사용자 정의 성능 규칙을 사용하여 새 TuneD 프로필을 생성합니다.

사전 요구 사항

- *TuneD 서비스가 실행 중입니다. 자세한 내용은 TuneD 설치 및 활성화를 참조하십시오.*

절차

1. */etc/tuned/ 디렉터리에서 생성하려는 프로필과 동일한 라는 새 디렉터리를 생성합니다.*

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

2. *새 디렉터리에 **tuned.conf** 라는 파일을 생성합니다. 요구 사항에 따라 **[main]** 섹션 및 플러그인 정의를 추가합니다.*

   *예를 들어 **balanced** 프로파일 구성을 참조하십시오.*

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

[cpu]
governor=conservative
energy_perf_bias=normal

[audio]
timeout=10

[video]
radeon_powersave=dpm-balanced, auto

[scsi_host]
alpm=medium_power
```

3. *프로필을 활성화하려면 다음을 사용합니다.*

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

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

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

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

추가 리소스

- ***tuned.conf(5) man page***

*2.13. 기존 **TUNED** 프로필 수정*

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

사전 요구 사항

- *TuneD 서비스가 실행 중입니다. 자세한 내용은 TuneD 설치 및 활성화를 참조하십시오.*

절차

1. */etc/tuned/ 디렉터리에서 생성하려는 프로필과 동일한 라는 새 디렉터리를 생성합니다.*

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

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

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

parent-profile 을 수정 중인 프로필 이름으로 교체합니다.

3. *프로파일 수정 사항을 포함합니다.*

예 2.10. throughput-performance 프로필에서 swappiness 감소

throughput-performance 프로필의 설정을 사용하고 기본값 10이 아닌 vm.swappiness 값을 5로 변경하려면 다음을 사용합니다.

```
[main]
include=throughput-performance

[sysctl]
vm.swappiness=5
```

4. *프로필을 활성화하려면 다음을 사용합니다.*

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

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

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

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

추가 리소스

- **tuned.conf(5) man page**

2.14. TUNED를 사용하여 디스크 스케줄러 설정

이 절차에서는 선택한 블록 장치에 지정된 디스크 스케줄러를 설정하는 **TuneD** 프로필을 생성하고 활성화합니다. 시스템 재부팅 시 설정이 유지됩니다.

다음 명령 및 구성에서 다음을 교체합니다.

- 블록 장치 의 이름이 있는 장치(예: **sdf**)
- 장치에 설정하려는 디스크 스케줄러가 있는 선택된- **scheduler**(예: **bfq**)

사전 요구 사항

- **TuneD** 서비스가 설치되고 활성화됩니다. 자세한 내용은 *TuneD 설치 및 활성화를* 참조하십시오.

절차

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

   현재 활성화되어 있는 프로필을 확인하려면 다음을 사용하십시오.

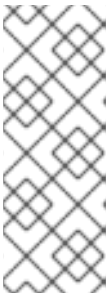
```
$ tuned-adm active
```

2. **TuneD** 프로필을 저장할 새 디렉터리를 생성합니다.

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

3. 선택한 블록 장치의 시스템 고유 식별자를 찾습니다.

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

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



참고

이 예제의 명령은 지정된 블록 장치와 연결된 **WWN(WWN)** 또는 일련 번호로 식별되는 모든 값을 반환합니다. **WWN**을 사용하는 것이 바람직하지만, 지정된 장치에 **WWN**을 항상 사용할 수는 없으며 **example** 명령에서 반환된 모든 값은 장치 시스템 고유 **ID** 로 사용할 수 있습니다.

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

a. 선택 사항: 기존 프로필이 포함되어 있습니다.

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

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

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

여기:

- **IDNAME** 을 사용 중인 식별자 이름(예: **ID_WWN**)으로 바꿉니다.

- *장치 시스템 고유 **ID** 를 선택한 식별자의 값으로 바꿉니다(예: **0x5002538d00000000**).*

  devices_udev_regex 옵션의 여러 장치와 일치하려면 식별자를 괄호로 묶고 세로 막대로 분리합니다.

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

5. *프로필을 활성화합니다.*

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

검증 단계

1. ***TuneD** 프로필이 활성화되어 적용되는지 확인합니다.*

   ```
   $ tuned-adm active

   Current active profile: my-profile
   ```

   ```
   $ tuned-adm verify

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

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

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

   [mq-deadline] kyber bfq none
   ```

   *파일 이름에서 장치를 블록 장치 이름으로 바꿉니다(예: **sdc**).*

   *활성 스케줄러는 대괄호(**[]**)로 나열됩니다.*

추가 리소스

- *TuneD 프로필 사용자 정의 .*

3장. TUNA 인터페이스를 사용하여 시스템 검토

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

tuna 툴은 다음 작업을 수행합니다.

- 시스템의 CPU 나열
- 시스템에서 현재 실행 중인 인터럽트 요청(IRQ)을 나열합니다.
- 스레드에 대한 정책 및 우선 순위 정보 변경
- 시스템의 현재 정책 및 우선순위 표시

3.1. TUNA 툴 설치

tuna 툴은 실행 중인 시스템에서 사용하도록 설계되었습니다. 이를 통해 애플리케이션별 측정 툴을 사용하여 변경 후 즉시 시스템 성능을 확인하고 분석할 수 있습니다.

이 절차에서는 tuna 툴을 설치하는 방법을 설명합니다.

절차

- tuna 툴을 설치합니다.

```
# dnf install tuna
```

검증 단계

- 사용 가능한 tuna CLI 옵션을 확인합니다.

```
# tuna -h
```

추가 리소스

- **tuna(8)** 매뉴얼 페이지

3.2. TUNA 툴을 사용하여 시스템 상태 보기

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

사전 요구 사항

- **tuna** 툴이 설치되어 있습니다. 자세한 내용은 **Installing tuna** 툴 을 참조하십시오.

절차

- 현재 정책 및 우선순위를 보려면 다음을 수행합니다.

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

- **PID**에 해당하는 특정 스레드 또는 명령 이름을 일치시키려면 다음을 수행합니다.

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

pid_or_cmd_list 인수는 쉼표로 구분된 **PID** 또는 **command-name** 패턴 목록입니다.

- **tuna CLI**를 사용하여 **CPU**를 튜닝하려면 **tuna** 툴을 사용하여 **CPU** 튜닝을 참조하십시오.

- **tuna** 툴을 사용하여 **IRQ**를 튜닝하려면 **tuna** 툴을 사용하여 **IRQ** 튜닝을 참조하십시오.

- 변경된 구성을 저장하려면 다음을 수행합니다.

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

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

추가 리소스

- tuna(8) 매뉴얼 페이지

3.3. TUNA 툴을 사용하여 CPU 튜닝

tuna 툴 명령은 개별 CPU를 대상으로 지정할 수 있습니다.

tuna 도구를 사용하면 다음을 수행할 수 있습니다.

CPU 격리

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

CPU 포함

지정된 CPU에서 작업을 실행 가능

CPU 복원

지정된 CPU를 이전 구성으로 복원합니다.

다음 프로세스에서는 tuna CLI를 사용하여 CPU를 조정하는 방법을 설명합니다.

사전 요구 사항

- tuna 툴이 설치되어 있습니다. 자세한 내용은 Installing tuna 툴 을 참조하십시오.

절차

- 명령의 영향을 받을 CPU 목록을 지정하려면 다음을 수행합니다.

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

cpu_list 인수는 쉼표로 구분된 CPU 번호 목록입니다. 예를 들면 --cpus=0,2 와 같습니다. CPU 목록은 범위에 지정할 수도 있습니다(예: --cpus="1-3", CPU 1, 2, 3을 선택하는).

현재 cpu_list 에 특정 CPU를 추가하려면 --cpus=+0 을 사용합니다.

[명령]을 (예: --isolate)로 바꿉니다.

- *CPU를 격리하려면 다음을 수행합니다.*

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

- *CPU를 포함하려면 다음을 수행합니다.*

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

- *4개 이상의 프로세서가 있는 시스템을 사용하려면 모든 ssh 스레드가 CPU 0 및 1 에서 실행되고 CPU 2 및 3 의 모든 http 스레드를 실행하는 방법을 표시합니다.*

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

이 명령은 다음 작업을 순서대로 수행합니다.

1. *CPU 0 및 1 을 선택합니다.*
2. *ssh 로 시작하는 모든 스레드를 선택합니다.*
3. *선택한 스레드를 선택한 CPU로 이동합니다. tuna는 ssh 로 시작하는 스레드의 선호도 마스크를 적절한 CPU에 설정합니다. CPU는 0 및 1 로, 16진수 마스크에서 0x3으로, 또는 바이너리에서 11으로 표시할 수 있습니다.*

4. *CPU 목록을 2 및 3 으로 재설정합니다.*

5. *http 로 시작하는 모든 스레드를 선택합니다.*

6. *선택한 스레드를 지정된 CPU로 이동합니다. tuna는 http 로 시작하는 스레드의 선호도 마스크를 지정된 CPU로 설정합니다. CPU는 2 와 3 으로, 16진수 마스크에서 0xC로 표현하거나, 바이너리에서 1100으로 표시할 수 있습니다.*

검증 단계

- *현재 구성을 표시하고 변경 사항이 예상대로 수행되었는지 확인합니다.*

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

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

이 명령은 다음 작업을 순서대로 수행합니다.

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

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

3. *CPU 0 을 선택합니다.*

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

5. *이동의 결과를 표시합니다.*

6. *CPU 목록을 CPU 1 로 재설정합니다.*

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

8. *이동 결과를 표시합니다.*

9. *CPU 0 을 CPU 목록에 추가합니다.*

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

11. *이동 결과를 표시합니다.*

추가 리소스

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

3.4. TUNA 툴을 사용하여 IRQ 튜닝

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

이 절차에서는 tuna 툴을 사용하여 IRQ를 조정하는 방법을 설명합니다.

사전 요구 사항

- **tuna** 툴이 설치되어 있습니다. 자세한 내용은 **Installing tuna** 툴 을 참조하십시오.

절차

- 현재 **IRQ** 및 선호도를 보려면 다음을 수행합니다.

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

- 명령에 영향을 줄 **IRQ** 목록을 지정하려면 다음을 수행합니다.

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

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

[명령]을 **(예: --spread)**로 바꿉니다.

- 인터럽트를 지정된 **CPU**로 이동하려면 다음을 수행합니다.

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

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

128 을 **irq_list** 인수 및 **3** 으로 **cpu_list** 인수로 바꿉니다.

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

검증 단계

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

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

추가 리소스

- ***/Procs/interrupts** 파일*
- ***tuna(8)** 매뉴얼 페이지*

4장. RHEL 시스템 역할을 사용하여 성능 모니터링

시스템 관리자는 Metrics RHEL 시스템 역할을 사용하여 시스템의 성능을 모니터링할 수 있습니다.

4.1. RHEL 시스템 역할 소개

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

Red Hat Enterprise Linux 9에서 인터페이스는 현재 다음 역할로 구성됩니다.

- 인증서 발행 및 업데이트
- 커널 설정
- 지표
- 네트워크 Bound Disk Encryption 클라이언트 및 네트워크 Bound Disk Encryption 서버
- 네트워킹
- Postfix
- SSH 클라이언트
- SSH 서버
- 시스템 전체 암호화 정책

- 터미널 세션 레코드

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

추가 리소스

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

4.2. RHEL 시스템 역할 용어

이 문서에서 다음 용어를 찾을 수 있습니다.

Ansible 플레이북

Ansible 플레이북은 **Ansible**의 구성, 배포 및 오케스트레이션 언어입니다. 원격 시스템이 적용하려는 정책을 설명하거나 일반 **IT** 프로세스에서 일련의 단계를 설명할 수 있습니다.

제어 노드

Ansible이 설치된 모든 머신. 모든 제어 노드에서 **/usr/bin/ansible** 또는 **/usr/bin/ansible-playbook**을 호출하는 명령과 플레이북을 실행할 수 있습니다. **Python**이 설치되어 있는 모든 컴퓨터를 제어 노드로 사용할 수 있습니다. **-** 랩톱, 공유 데스크탑 및 서버는 모두 **Ansible**을 실행할 수 있습니다. 그러나 **Windows** 머신을 제어 노드로 사용할 수 없습니다. 여러 개의 제어 노드를 가질 수 있습니다.

인벤토리

관리형 노드 목록입니다. 인벤토리 파일을 **"hostfile"**라고도 합니다. 인벤토리는 각 관리 노드의 **IP** 주소와 같은 정보를 지정할 수 있습니다. 인벤토리를 사용하면 관리형 노드를 구성하고, 그룹을 만들고 중첩하여 쉽게 확장할 수 있습니다. 인벤토리에 대한 자세한 내용은 **Working with Inventory** 섹션을 참조하십시오.

관리형 노드

Ansible로 관리하는 네트워크 장치, 서버 또는 둘 다입니다. 관리 노드는 **"hosts"**라고도 합니다. **Ansible**은 관리형 노드에 설치되지 않습니다.

4.3. 시스템에서 RHEL 시스템 역할 설치

RHEL 시스템 역할을 사용하려면 시스템에 필수 패키지를 설치하십시오.

사전 요구 사항

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

절차

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

```
# dnf install rhel-system-roles
```

2. **Ansible Core** 패키지를 설치합니다.

```
# dnf install ansible-core
```

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

결과적으로 **Ansible** 플레이북을 생성할 수 있습니다.

추가 리소스

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

4.4. 역할 적용

다음 절차에서는 특정 역할을 적용하는 방법을 설명합니다.

사전 요구 사항

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

```
# dnf install rhel-system-roles
```

1. **Ansible Core** 패키지를 설치합니다.

```
# dnf install ansible-core
```

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

- **Ansible** 인벤토리를 생성할 수 있는지 확인합니다.

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

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

- **Ansible** 플레이북을 생성할 수 있는지 확인합니다.

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

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

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

절차

1. 관리하려는 호스트 및 그룹이 포함된 필수 **Ansible** 인벤토리를 생성합니다. 다음은 **webservers** 라는 호스트 그룹의 **inventory.ini** 라는 파일을 사용하는 예입니다.

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

2.

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

다음 예제에서는 지정된 플레이 의 **roles:** 옵션을 통해 역할을 사용하는 방법을 보여줍니다.

```
---
- hosts: webservers
  roles:

    - rhel-system-roles.network
    - rhel-system-roles.postfix
```

참고

모든 역할에는 **README** 파일이 포함되어 있으며, 이 파일은 역할 및 지원되는 매개 변수 값을 사용하는 방법을 문서화합니다. 역할의 설명서 디렉터리에서 특정 역할에 대한 예제 플레이북을 찾을 수도 있습니다. 이러한 문서 디렉터리는 기본적으로 **rhel-system-roles** 패키지로 제공되며 다음 위치에서 찾을 수 있습니다.

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

SUBSYSTEM 을 **postfix,metrics,network,tlog** 또는 **ssh** 와 같은 필수 역할의 이름으로 바꿉니다.

3.

특정 호스트에서 플레이북을 실행하려면 다음 중 하나를 수행해야 합니다.

- **host1[,host2,...]** 또는 **hosts(**모든 **)**를 사용하도록 플레이북을 편집하고 명령을 실행합니다.

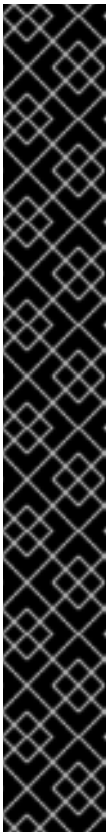
```
# ansible-playbook name.of.the.playbook
```

- 인벤토리를 편집하여 사용하려는 호스트가 그룹에 정의되어 있는지 확인하고 명령을 실행합니다.

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

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

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



중요

-i 플래그는 사용 가능한 모든 호스트의 인벤토리를 지정합니다. 대상 호스트가 여러 개 있지만 플레이북을 실행하려는 호스트를 선택하려면 플레이북에 변수를 추가하여 호스트를 선택할 수 있습니다. 예를 들어 다음과 같습니다.

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


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

플레이북 실행 명령:

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

추가 리소스

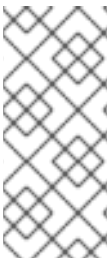
- **Ansible** 플레이북
- **Ansible** 플레이북에서 역할 사용
- **Ansible** 플레이북의 예
- 인벤토리를 작성하고 사용하는 방법은 무엇입니까?
- **ansible-playbook** 툴

4.5. 지표 시스템 역할 소개

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

표 4.1. 지표 시스템 역할 변수

역할 변수	설명	사용 예
metrics_monitored_hosts	대상 호스트에서 분석할 원격 호스트 목록입니다. 이러한 호스트에는 대상 호스트에 기록된 지표가 있으므로 각 호스트의 **/var/log** 아래에 충분한 디스크 공간이 있는지 확인합니다.	**metrics_monitored_hosts: ["*webserver.example.com*", "*database.example.com*"]**
metrics_retention_days	삭제하기 전에 성능 데이터 보존 기간의 일 수를 구성합니다.	**metrics_retention_days: 14**
metrics_graph_service	**pcp** 및 **grafana** 를 통해 성능 데이터 시각화를 위한 서비스로 호스트를 설정할 수 있는 부울 플래그입니다. 기본적으로 false로 설정합니다.	**metrics_graph_service: no**
metrics_query_service	Redis를 통해 기록된 **pcp** 지표를 쿼리하기 위한 시계열 쿼리 서비스로 호스트를 설정할 수 있는 부울 플래그 **입니다**. 기본적으로 false로 설정합니다.	**metrics_query_service: no**
metrics_provider	메트릭을 제공하는 데 사용할 지표 수집기를 지정합니다. 현재 **pcp** 는 지원되는 유일한 메트릭 공급자입니다.	**metrics_provider: "pcp"**



참고

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

4.6. 시각화로 로컬 시스템을 모니터링하기 위해 메트릭 시스템 역할을 사용

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

사전 요구 사항

- ***Ansible Core** 패키지는 제어 시스템에 설치됩니다.*
- *모니터링할 시스템에 **rhel-system-roles** 패키지가 설치되어 있습니다.*

절차

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

```
localhost ansible_connection=local
```

2. *다음 콘텐츠를 사용하여 **Ansible** 플레이북을 생성합니다.*

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

3. ***Ansible Playbook**을 실행합니다.*

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

참고

***metrics_graph_service** 부울이 **value="yes"**로 설정되므로 **Grafana** 는 데이터 소스로 **pcp** 를 추가하여 자동으로 설치 및 프로비저닝됩니다.*

4. *시스템에서 수집 중인 지표의 시각화를 보려면 **Grafana 웹 UI 액세스**에 설명된 대로 **grafana 웹 인터페이스에 액세스** 합니다.*

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

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

사전 요구 사항

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

절차

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

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

2. *다음 콘텐츠를 사용하여 **Ansible** 플레이북을 생성합니다.*

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

3. ***Ansible Playbook**을 실행합니다.*

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

*원격 시스템에 연결하기 위해 **-k** 프롬프트를 요청하는 경우 **-k**.*

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

*이 절차에서는 **Metrics** 시스템 역할을 사용하여 시스템을 중앙에서 모니터링하도록 로컬 머신을 설정*

*하는 방법을 설명하고, **grafana** 를 통해 데이터의 시각화를 프로비저닝하고 **redis** 를 통해 데이터를 쿼리합니다.*

사전 요구 사항

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

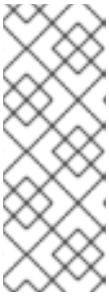
절차

1. *다음 콘텐츠를 사용하여 **Ansible** 플레이북을 생성합니다.*

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

2. ***Ansible Playbook**을 실행합니다.*

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



참고

metrics_graph_service 및 metrics_query_service 부울은 value="yes"로 설정되므로 grafana 는 pcp 데이터가 redis 로 인덱싱된 데이터 소스로서 자동으로 설치 및 프로비저닝되므로 pcp 쿼리 언어를 사용하여 데이터의 복잡한 쿼리에 pcp 쿼리 언어를 사용할 수 있습니다.

3. *시스템에서 중앙에서 수집되고 데이터를 쿼리하고 **Grafana** 웹 **UI**에 액세스하는 방법에 설명된 대로 **grafana** 웹 인터페이스에 대한 그래픽 표시를 보려면 Grafana 웹 UI에 액세스 합니다.*

4.9. METRICS 시스템 역할을 사용하여 시스템을 모니터링하는 동안 인증 설정

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

사전 요구 사항

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

절차

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

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

2. **Ansible Playbook을 실행합니다.**

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

검증 단계

- **sasl 구성을 확인합니다.**

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

ip_adress 는 호스트의 IP 주소로 교체해야 합니다.

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

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

사전 요구 사항

- *Ansible Core 패키지는 제어 시스템에 설치됩니다.*
- *모니터링할 시스템에 rhel-system-roles 패키지가 설치되어 있습니다.*
- *Microsoft SQL Server for Red Hat Enterprise Linux를 설치하고 SQL 서버에 대한 '신뢰' 연결을 설정했습니다. SQL Server 설치를 참조하고 Red Hat에 데이터베이스를 만듭니다.*
- *SQL Server for Red Hat Enterprise Linux용 Microsoft QoS 드라이버를 설치했습니다. Red Hat Enterprise Server 및 Oracle Linux 를 참조하십시오.*

절차

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

```
localhost ansible_connection=local
```

2. *다음 콘텐츠가 포함된 Ansible 플레이북을 생성합니다.*

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

3. *Ansible Playbook을 실행합니다.*

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

검증 단계

- *pcp 명령을 사용하여 SQL Server PMDA 에이전트(mssql)가 로드되고 실행되고 있는지 확인합니다.*

```
# pcp
platform: Linux rhel82-2.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC
```

```
2019 x86_64
 hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
 timezone: PDT+7
 services: pmcd pmproxy
    pmcd: Version 5.0.2-1, 12 agents, 4 clients
    pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
          jbd2 dm
 pmlogger: primary logger: /var/log/pcp/pmlogger/rhel82-2.local/20200326.16.31
    pmie: primary engine: /var/log/pcp/pmie/rhel82-2.local/pmie.log
```

추가 리소스

- ***Microsoft SQL Server*에 대한 *Performance Co-inspector* 사용에 대한 자세한 내용은 이 *Red Hat Developers* 블로그 게시물을 참조하십시오.**

---

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

5장. PCP 설정

PCP(Performance Co-915)는 시스템 수준 성능 측정을 모니터링, 시각화, 저장 및 분석하기 위한 도구, 서비스 및 라이브러리 제품군입니다.

이 섹션에서는 시스템에 PCP를 설치하고 활성화하는 방법을 설명합니다.

5.1. PCP 개요

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

라이브 결과를 아카이브 데이터와 비교하여 데이터 패턴을 분석할 수 있습니다.

PCP의 기능:

- 경량의 분산 아키텍처는 복잡한 시스템을 분석하는 동안 유용합니다.
- 실시간 데이터를 모니터링하고 관리할 수 있습니다.
- 기록 데이터를 로깅 및 검색할 수 있습니다.

PCP에는 다음과 같은 구성 요소가 있습니다.

- Performance Metric Collector Daemon(pmcd)은 설치된 Performance Metric Domain Agents(pmda)에서 성능 데이터를 수집합니다. PMDA 는 시스템에서 개별적으로 로드되거나 언로드될 수 있으며, 동일한 호스트의 PMCD 에 의해 제어됩니다.
- pminfo 또는 pmstat 와 같은 다양한 클라이언트 도구는 동일한 호스트 또는 네트워크를 통해 이 데이터를 검색, 표시, 아카이브 및 처리할 수 있습니다.

- *pcp 패키지는 명령줄 도구 및 기본 기능을 제공합니다.*

- *pcp-gui 패키지는 그래픽 애플리케이션을 제공합니다. dnf install pcp-gui 명령을 실행하여 pcp-gui 패키지를 설치합니다. 자세한 내용은 PCP 차트 애플리케이션을 사용하여 PCP 로그 아카이브 시각화를 참조하십시오.*

추가 리소스

- *pcp(1) man page*

- */usr/share/doc/pcp-doc/ directory*

- *PCP와 함께 배포된 툴*

- *Red Hat 고객 포털의 Performance Co-inspector(PCP) 문서, 솔루션, 튜토리얼, 백서*

- *기존 툴과 PCP 툴을 나란히 비교 Red Hat Knowledgebase 문서*

- *PCP 업스트림 문서*

5.2. PCP 설치 및 활성화

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

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

절차

1. *pcp 패키지를 설치합니다.*

```
# dnf install pcp
```

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

```
# systemctl enable pmcd

# systemctl start pmcd
```

검증 단계

- **pmcd** 프로세스가 호스트에서 실행 중인지 확인합니다.

```
# pcp

Performance Co-Pilot configuration on workstation:

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

추가 리소스

- **pmcd(1)** 도움말 페이지
- PCP와 함께 배포된 툴

5.3. 최소 PCP 설정 배포

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

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

사전 요구 사항

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

절차

1. ***pmlogger** 구성을 업데이트합니다.*

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

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

```
# systemctl start pmcd.service

# systemctl start pmlogger.service
```

3. *성능 데이터를 기록하는 데 필요한 작업을 실행합니다.*

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

```
# systemctl stop pmcd.service

# systemctl stop pmlogger.service
```

5. *출력을 저장하고 호스트 이름과 현재 날짜와 시간을 기반으로 이름이 지정된 **tar.gz** 파일에 저장합니다.*

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

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

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

추가 리소스

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

5.4. PCP로 배포된 시스템 서비스

다음 표에서는 PCP와 함께 배포되는 다양한 시스템 서비스의 역할을 설명합니다.

표 5.1. PCP와 함께 배포되는 시스템 서비스의 역할

이름	설명
pmcd	PMCD(Performance Metric Collector Daemon)입니다.
pmie	성능 지표 유추 엔진.
pmlogger	성능 지표 로거입니다.
pmproxy	실시간 및 기록 성능 지표 프록시, 시계열 쿼리 및 REST API 서비스.

5.5. PCP와 함께 배포된 툴

다음 표에서는 PCP와 함께 배포되는 다양한 도구 사용에 대해 설명합니다.

표 5.2. PCP와 함께 배포된 툴 사용

이름	설명
pcp	Performance Co-inspector 설치의 현재 상태를 표시합니다.
pcp-atop	CPU, 메모리, 디스크, 네트워크 등 성능 관점에서 가장 중요한 하드웨어 리소스의 시스템 수준 occupation을 보여줍니다.
pcp-atopsar	다양한 시스템 리소스 사용률에 대해 시스템 수준 활동 보고서를 생성합니다. 이 보고서는 pmlogger 또는 pcp-atop의 -w 옵션을 사용하여 이전에 기록된 원시 로그 파일에서 생성됩니다.
pcp-dmcache	장치 IOP, 캐시 및 메타데이터 장치 사용률과 각 캐시 장치에 대한 읽기 및 쓰기 비율 및 비율과 같은 구성된 장치 매퍼 캐시 대상에 대한 정보를 표시합니다.
pcp-dstat	한 번에 하나의 시스템의 지표를 표시합니다. 여러 시스템의 지표를 표시하려면 **--host** 옵션을 사용합니다.

pcp-free	시스템에서 사용 가능한 메모리 및 사용된 메모리에 대한 보고서
pcp-htop	에서 실행되는 모든 프로세스를 **최상위** 명령과 유사한 방식으로 명령줄 인수와 함께 표시하지만 마우스를 사용하여 세로로 스크롤하고 수평으로 이동할 수 있습니다. 또한 트리 형식으로 프로세스를 보고 여러 프로세스를 한 번에 선택 및 실행할 수도 있습니다.
pcp-ipcs	호출 프로세스가 읽을 수 있는 프로세스 간 통신(IPC) 기능에 대한 정보를 표시합니다.
pcp-numastat	커널 메모리 할당기의 NUMA 할당 통계를 표시합니다.
pcp-pidstat	CPU 백분율, 메모리, 스택 사용량, 스케줄링 및 우선 순위와 같은 시스템에서 실행 중인 개별 작업 또는 프로세스에 대한 정보를 표시합니다. 기본적으로 로컬 호스트의 실시간 데이터를 보고합니다.
pcp-ss	pmdasockets Performance Metrics Domain Agent(PMDA)에서 수집한 소켓 통계를 표시합니다.
pcp-uptime	는 시스템이 실행 중인 기간, 현재 로그인한 사용자 수 및 시스템 부하 평균 지난 1, 5, 15분을 표시합니다.
pcp-vmstat	5초마다 고급 시스템 성능 개요를 제공합니다. 프로세스, 메모리, 페이징, 블록 IO, 트랩 및 CPU 활동에 대한 정보를 표시합니다.
pmchart	Performance Co-inspector 기능을 통해 사용 가능한 성능 지표 값을 작성하고 있습니다.
pmclient	PMAPI(Performance Metrics Application Programming Interface)를 사용하여 고급 시스템 성능 지표를 표시합니다.
pmconfig	구성 매개변수 값을 표시합니다.
pmdbg	사용 가능한 Performance Co-inspector debug 제어 플래그 및 해당 값을 표시합니다.
pmdiff	지정된 시간 창에서 하나 또는 두 아카이브의 모든 메트릭의 평균 값을 비교하여 성능 회귀를 검색할 때 관심이 있을 수 있는 변경 사항을 비교합니다.
pmdumplog	Performance Co-inspector 아카이브 파일의 제어, 메타데이터, 인덱스 및 상태 정보를 표시합니다.
pmdumptext	실시간 또는 Performance Co-inspector 아카이브에서 수집한 성능 지표 값을 출력합니다.

pmerr	사용 가능한 Performance Co-inspector 오류 코드 및 해당 오류 메시지를 표시합니다.
pmfind	네트워크에서 PCP 서비스를 찾습니다.
pmie	정기적으로 산술, 논리 및 규칙 식 집합을 평가하는 유추 엔진입니다. 지표는 라이브 시스템 또는 Performance Co-inspector 아카이브 파일에서 수집됩니다.
pmieconf	구성 가능한 pmie 변수를 표시하거나 설정합니다.
pmiectl	pmie의 인스턴스를 관리합니다.
pminfo	성능 지표에 대한 정보를 표시합니다. 지표는 라이브 시스템 또는 Performance Co-inspector 아카이브 파일에서 수집됩니다.
pmiostat	SCSI 장치(기본적으로) 또는 device-mapper 장치에 대한 I/O 통계를 보고합니다(-x dm 옵션 사용).
pmlc	활성 pmlogger 인스턴스를 대화식으로 구성합니다.
pmlogcheck	Performance Co-inspector 아카이브 파일에서 잘못된 데이터를 식별합니다.
pmlogconf	pmlogger 구성 파일을 만들고 수정합니다.
pmlogctl	pmlogger의 인스턴스를 관리합니다.
pmloglabel	Performance Co-inspector 아카이브 파일의 레이블을 검증, 수정 또는 복구합니다.
pmlogsummary	Performance Co-inspector 아카이브 파일에 저장된 성능 지표에 대한 통계 정보를 계산합니다.
pmprobe	성능 지표의 가용성을 결정합니다.
pmrep	선택한, 쉽게 사용자 지정 가능, 성능 지표 값에 대한 보고서.
pmsocks	방화벽을 통해 Performance Co-inspector 호스트에 액세스할 수 있습니다.
pmstat	정기적으로 시스템 성능에 대한 간략한 요약을 표시합니다.
pmstore	성능 지표 값을 수정합니다.

pmtrace	추적 PMDA에 명령줄 인터페이스를 제공합니다.
pmval	현재 성능 지표 값을 표시합니다.

5.6. PCP 배포 아키텍처

***PCP(Performance Co-databind)**는 고급 설정을 수행하기 위한 다양한 옵션을 제공합니다. 가능한 다양한 아키텍처에서 이 섹션에서는 **Red Hat**이 설정한 권장 배포, 크기 조정 요인 및 구성 옵션을 기반으로 **PCP** 배포를 확장하는 방법을 설명합니다.*

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

사용 가능한 스케일링 배포 설정 변형:

localhost

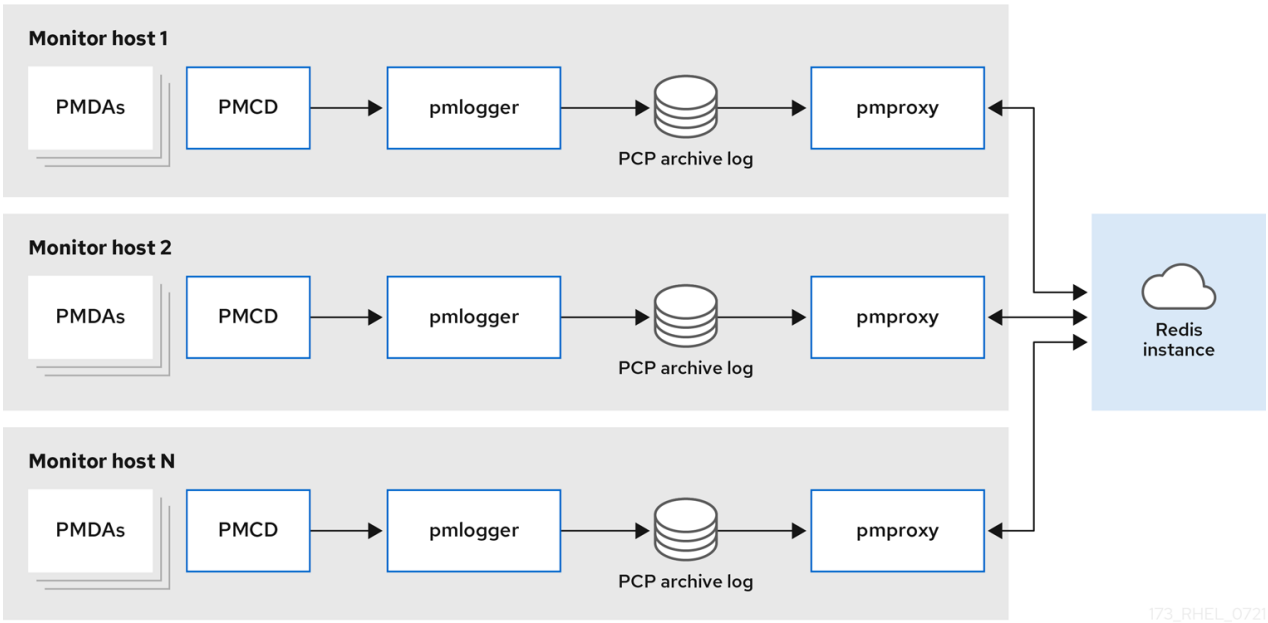
각 서비스는 모니터링된 시스템에서 로컬로 실행됩니다. 구성을 변경하지 않고 서비스를 시작하면 기본 배포입니다. 이 경우 개별 노드 이상으로 스케일링할 수 없습니다.

*기본적으로 **Redis**에 대한 배포 설정은 독립 실행형 **localhost**입니다. 그러나 **Redis**는 필요에 따라 고가용성 및 확장성이 뛰어난 클러스터형 방식으로 작업을 수행할 수 있습니다. 여기서 데이터는 여러 호스트에서 공유됩니다. 또 다른 실행 가능한 옵션은 클라우드에 **Redis** 클러스터를 배포하거나 클라우드 벤더의 관리형 **Redis** 클러스터를 사용하는 것입니다.*

Decentralized

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

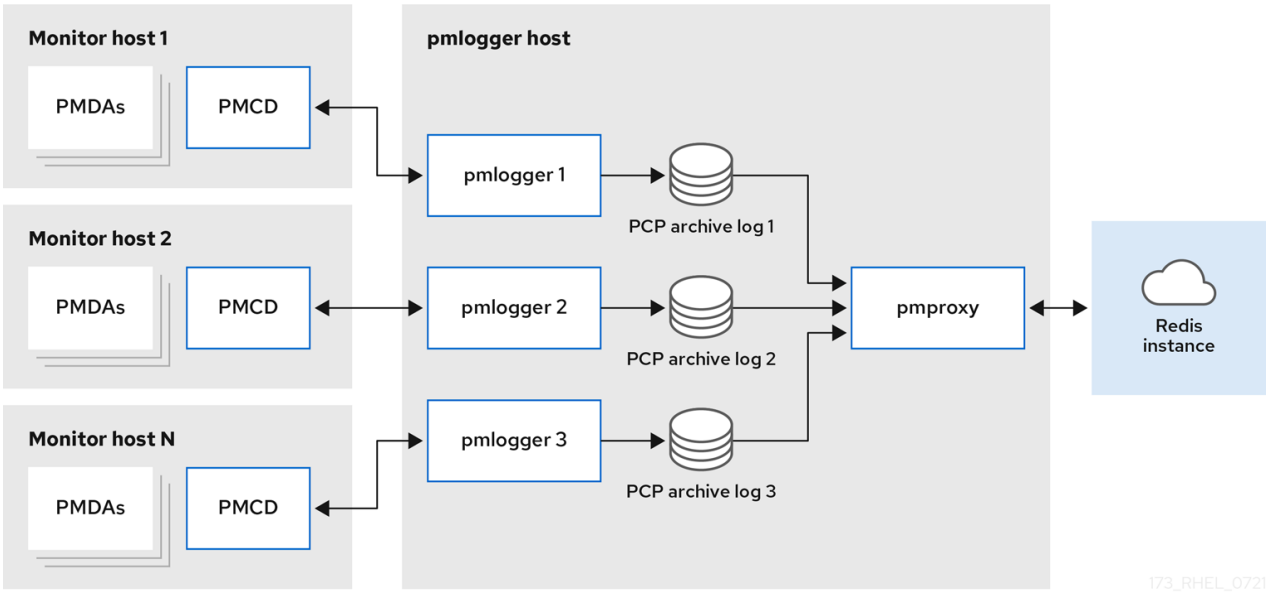
그림 5.1. 분산된 로깅



중앙 집중식 로깅 - pmloggerarm

*모니터링된 호스트의 리소스 사용량이 제한되면 다른 배포 옵션은 중앙 집중식 로깅이라고도 하는 **pmlogger arm**입니다. 이 설정에서 단일 로거 호스트는 여러 **pmlogger** 프로세스를 실행하고 각각 다른 원격 **pmcd** 호스트에서 성능 지표를 검색하도록 구성됩니다. 중앙 집중식 로거 호스트는 결과 **PCP** 아카이브 로그를 검색하고 지표 데이터를 **Redis** 인스턴스로 로드하는 **pmproxy** 서비스를 실행하도록 구성됩니다.*

그림 5.2. 중앙 집중식 로깅 - pmloggerarm

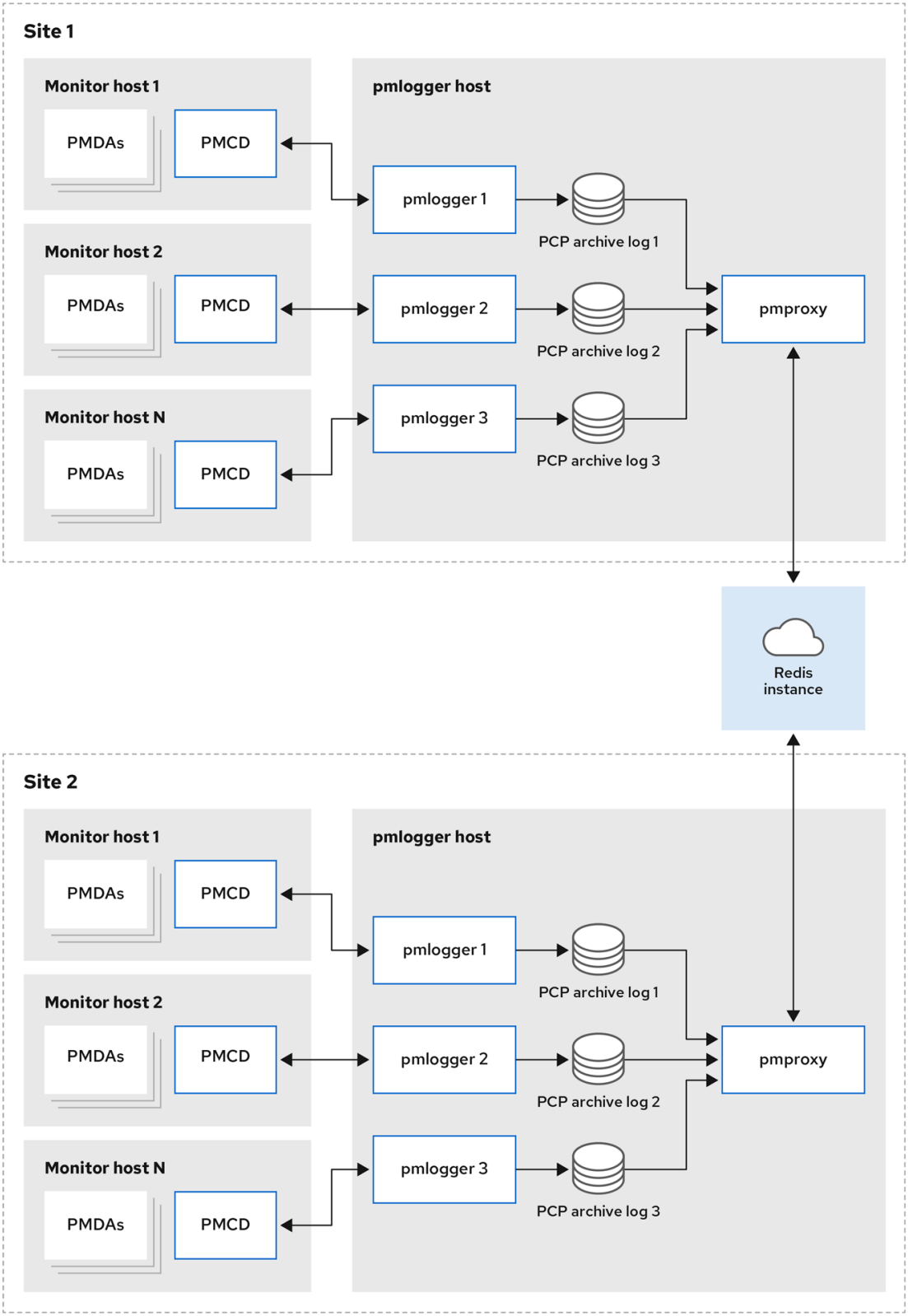


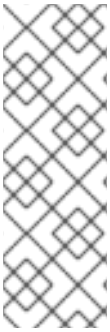
페더레이션 - 여러 pmloggerarms

*대규모 배포의 경우 **Red Hat**은 여러 **pmlogger arms**를 연합 방식으로 배포하는 것이 좋습니다. 예를 들어 랙 또는 데이터 센터당 하나의 **pmlogger arm**이 있습니다. 각 **pmlogger arm**은 중앙 **Redis***

인스턴스로 메트릭을 로드합니다.

그림 5.3. 페더레이션 - 여러 pmloggerarms





참고

*기본적으로 **Redis**에 대한 배포 설정은 독립 실행형 **localhost**입니다. 그러나 **Redis**는 필요에 따라 고가용성 및 확장성이 뛰어난 클러스터형 방식으로 작업을 수행할 수 있습니다. 여기서 데이터는 여러 호스트에서 공유됩니다. 또 다른 실행 가능한 옵션은 클라우드에 **Redis** 클러스터를 배포하거나 클라우드 벤더의 관리형 **Redis** 클러스터를 사용하는 것입니다.*

추가 리소스

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

5.7. 권장되는 배포 아키텍처

다음 표에서는 모니터링된 호스트 수에 따라 권장되는 배포 아키텍처를 설명합니다.

*표 **5.3.** 권장되는 배포 아키텍처*

호스트 수 (N)	1-10	10-100	100-1000
pmcd 서버	N	N	N
pmlogger 서버	1에서 N으로	N/10에서 N으로	N/100에서 N/100으로
pmproxy 서버	1에서 N으로	1에서 N으로	N/100에서 N/100으로
Redis 서버	1에서 N으로	1에서 N/10으로	N/100에서 N/10으로
redis cluster	없음	수 있습니다.	있음
권장되는 배포 설정	localhost, 분산화된 또는 중앙 집중식 로깅	분산, 중앙 집중식 로깅 또는 Federated	분산되거나 페어링

5.8. 크기 조정 요인

다음은 스케일링에 필요한 크기 조정 요인입니다.

원격 시스템 크기

CPU, 디스크, 네트워크 인터페이스 및 기타 하드웨어 리소스의 수는 중앙 집중식 로깅 호스트의 각 pmlogger 에서 수집한 데이터 양에 영향을 미칩니다.

기록된 메트릭

기록된 지표의 수와 유형이 중요한 역할을 합니다. 특히 프로세스별 proc.* 메트릭에는 표준 pcp-zeroconf 설정, 10s 로깅 간격 10MB, proc 메트릭이 없는 11MB - proc 메트릭이 포함된 약 10배 더 많은 디스크 공간이 필요합니다. 또한 각 메트릭의 인스턴스 수(예: CPU 수, 블록 장치 및 네트워크 인터페이스 수)도 필요한 스토리지 용량에 영향을 미칩니다.

로깅 간격

지표가 기록되는 간격은 스토리지 요구 사항에 영향을 미칩니다. 예상되는 일일 PCP 아카이브 파일 크기는 각 pmlogger 인스턴스의 pmlogger.log 파일에 작성됩니다. 이러한 값은 압축되지 않은 추정치입니다. PCP 아카이브가 매우 잘 압축되므로 약 10:1이 특정 사이트에 대해 실제 장기 디스크 공간 요구 사항을 확인할 수 있습니다.

pmlogrewrite

모든 PCP 업그레이드 후 pmlogrewrite 도구가 실행되고 이전 버전의 지표 메타데이터와 새 버전의 PCP가 변경된 경우 이전 아카이브를 다시 작성합니다. 이 프로세스 기간은 저장된 아카이브 수로 선형으로 확장됩니다.

추가 리소스

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

5.9. PCP 스케일링을 위한 구성 옵션

다음은 확장에 필요한 구성 옵션입니다.

sysctl 및 rlimit 설정

아카이브 검색이 활성화되면 pmproxy 에는 모니터링 또는 로그 세부 정보인 모든 pm proxy에 대해 서비스 로그 및 pmproxy 클라이언트 소켓의 추가 파일 설명자와 함께 4개의 설명자가 필요합니다. 각 pmlogger 프로세스는 원격 pmcd 소켓, 아카이브 파일, 서비스 로그 등에 약 20개의 파일 설명자를 사용합니다. 약 200 pmlogger 프로세스를 실행하는 시스템의 기본 1024 소프트 제한을 초과할 수 있습니다. pcp-5.3.0 이상의 pmproxy 서비스가 하드 제한으로 자동으로 증가합니다. 이전 버전의 PCP에서는 다수의 pmlogger 프로세스를 배포할 경우 튜닝이 필요하며, pmlogger 의 소프트 또는 하드 제한을 늘려 이를 수행할 수 있습니다. 자세한 내용은 systemd에서 실행하는 서비스의 제한(ulimit)을 설정하는 방법을 참조하십시오.

로컬 아카이브

pmlogger 서비스는 /var/log/pcp/pmlogger/ 디렉터리에 로컬 및 원격 pmcds 의 지표를 저장합니다. 로컬 시스템의 로깅 간격을 제어하려면 /etc/pcp/pmlogger/control.d/configfile 파일을 업데이트하고 인수에 -t X 를 추가합니다. 여기서 X 는 로깅 간격(초)입니다. 로깅해야 하는 메트릭을 구성하

려면 **pmlogconf /var/lib/pcp/config/pmlogger/config.clienthostname** 을 실행합니다. 이 명령은 기본 지표 세트를 사용하여 구성 파일을 배포합니다. 이러한 지표는 선택적으로 추가로 사용자 지정할 수 있습니다. 보존 설정을 지정하려면 이전 **PCP** 아카이브를 제거하고, **/etc/sysconfig/pmlogger_timers** 파일을 업데이트하고 **PMLOGGER_DAILY_PARAMS="-E -k X"**, 여기서 **X** 는 **PCP** 아카이브를 유지하기 위한 일 수입니다.

Redis

pmproxy 서비스는 로깅된 지표를 **pmlogger** 에서 **Redis** 인스턴스로 보냅니다. 다음은 **/etc/pcp/pmproxy/pmproxy.conf** 설정 파일에 보존 설정을 지정하는 두 가지 옵션입니다.

- **stream.expire** 는 오래된 지표를 제거해야 하는 기간, 지정된 시간 내에 업데이트되지 않은 지표를 지정합니다.
- **stream.maxlen** 은 호스트당 하나의 메트릭 값의 최대 메트릭 값을 지정합니다. 이 설정은 보존 간격(예: **20160**)에서 **14**일 동안의 보존 및 **60s** 로깅 간격(**60*60*24*14/60**)으로 나눈 보존 시간이어야 합니다.

추가 리소스

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

5.10. 예: 중앙 집중식 로깅 배포 분석

다음 결과는 기본 **pcp-zeroconf 5.3.0** 설치로 알려진 중앙 집중식 로깅 설정에서 수집되었으며, 각 원격 호스트는 **64 CPU** 코어, **376GB RAM** 및 **1**개의 디스크가 있는 서버에서 **pmcd** 를 실행하는 동일한 컨테이너 인스턴스입니다.

로깅 간격은 **10s**이며, 원격 노드의 **proc** 메트릭은 포함되지 않으며 메모리 값은 **RSS(Resident Set Size)** 값을 나타냅니다.

표 **5.4. 10s** 로깅 간격에 대한 자세한 사용률 통계

호스트 수	10	50
PCP Archives 스토리지 일당	91MB	522 MB
pmlogger Memory	160MB	580MB
pmlogger 네트워크 1일당 (In)	2MB	9MB

호스트 수	10	50
pmproxy 메모리	1.4GB	6.3GB
Redis Memory per Day	2.6 GB	12GB

*표 **5.5. 60s** 로깅 간격에 대해 모니터링된 호스트에 따라 사용되는 리소스*

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

참고

***pmproxy** 대기열 **Redis**는 **Redis pipelining** 요청을 요청하고 **Redis** 파이프**lining**을 사용하여 **Redis** 쿼리 속도를 높입니다. 이로 인해 메모리 사용량이 증가할 수 있습니다. 이 문제를 해결하려면 높은 메모리 사용량 문제 해결을 참조하십시오.*

5.11. 예: 페더레이션 설정 배포 분석

*다음 결과는 세 가지 중앙 집중식 로깅 (**pmlogger farm**) 설정으로 구성된 여러**pmlogger arms**로 알려진 페더레이션 설정에서 관찰되었으며, 각 **pmlogger farm**는 총 **300** 개의 원격 호스트를 모니터링했습니다.*

*이 **pmlogger arms**의 설정은 **Redis** 서버가 클러스터 모드에서 작동하는 경우를 제외하고 **60s** 로깅 간격에 대한 중앙 집중식 로깅 배포 와 동일합니다.*

*표 **5.6. 60s** 로깅 간격을 위한 페더레이션 호스트에 따라 사용되는 리소스*

PCP Archives 스토리지 일당	**pmlogger** Memory	네트워크 1일당 (In/Out)	**pmproxy** 메모리	Redis Memory per Day
277 MB	1058MB	15.6 MB / 12.3 MB	6-8GB	5.5GB

여기서는 모든 값은 호스트당입니다. **Redis** 클러스터의 노드 간 통신으로 인해 네트워크 대역폭이 더 높습니다.

5.12. 높은 메모리 사용량 문제 해결

다음 시나리오에서는 메모리 사용량이 증가할 수 있습니다.

- **pmproxy** 프로세스는 새 **PCP** 아카이브를 처리하는 데 사용 중이며 **Redis** 요청 및 응답을 처리하는 데 필요한 **CPU** 사이클은 없습니다.
- **Redis** 노드 또는 클러스터가 과부하되어 제 시간에 들어오는 요청을 처리할 수 없습니다.

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

이 섹션에서는 높은 메모리 사용량 문제를 해결하는 방법을 설명합니다.

사전 요구 사항

1. **pcp-pmda-redis** 패키지를 설치합니다.

```
# dnf install pcp-pmda-redis
```

2. **redis PMDA**를 설치합니다.

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

절차

- 메모리 사용량의 문제를 해결하려면 다음 명령을 실행하고 **inflight** 열을 관찰합니다.

```
$ pmrep :pmproxy
       backlog  inflight  reqs/s  resp/s   wait req err  resp err  changed  throttled
        byte     count   count/s  count/s  s/s  count/s   count/s  count/s   count/s
14:59:08  0        0       N/A      N/A   N/A    N/A      N/A      N/A       N/A
```

```
14:59:09  0      0   2268.9   2268.9   28    0      0     2.0      4.0
14:59:10  0      0     0.0      0.0    0     0      0     0.0      0.0
14:59:11  0      0     0.0      0.0    0     0      0     0.0      0.0
```

*이 열에는 진행 중인 **Redis** 요청 수가 표시됩니다. 즉, 대기하거나 전송되며 지금까지 응답이 수신되지 않았습니다.*

숫자가 높으면 다음 조건 중 하나를 나타냅니다.

- *__pmproxy__ 프로세스는 새 **PCP** 아카이브를 처리하는 데 사용 중이며 **Redis** 요청 및 응답을 처리하는 데 필요한 **CPU** 사이클은 없습니다.*
- *__Redis__ 노드 또는 클러스터가 과부하되어 제 시간에 들어오는 요청을 처리할 수 없습니다.*

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

*__Redis__ 노드가 장기간에 **100% CPU**를 사용하는 경우 더 나은 성능을 갖춘 호스트로 이동하거나 클러스터형 **Redis** 설정을 대신 사용하십시오.*

- *__pmproxy.redis.*__ 메트릭을 보려면 다음 명령을 사용합니다.*

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

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

*진행 중인 **Redis** 요청 수를 보려면 **pmproxy.redis.requests.inflight.total** 지표 및 **pmproxy.redis.requests.inflight.bytes** 지표를 참조하여 현재의 모든 **inflight Redis** 요청에 의해 사용되는 바이트 수를 확인합니다.*

*일반적으로 **redis** 요청 대기열은 **0**이지만 대규모 **pmloggerarms**를 사용하여 빌드할 수 있으므로 확장성은 제한되고 **pmproxy** 클라이언트에 대한 높은 대기 시간이 발생할 수 있습니다.*

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

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

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

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

추가 리소스

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

6장. PMLOGGER로 성능 데이터 로깅

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

pmlogger 도구를 사용하면 다음을 수행할 수 있습니다.

- 시스템에서 선택한 메트릭의 보관된 로그 생성
- 시스템에 기록된 메트릭과 얼마나 자주 기록되는지 지정합니다.

6.1. PMLOGCONF로 PMLOGGER 구성 파일 수정

pmlogger 서비스가 실행 중이면 PCP는 호스트에 기본 지표 세트를 기록합니다.

pmlogconf 유틸리티를 사용하여 기본 구성을 확인합니다. pmlogger 구성 파일이 없으면 pmlogconf 에서 기본 지표 값으로 생성합니다.

사전 요구 사항

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

절차

1. pmlogger 구성 파일을 생성하거나 수정합니다.

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

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

추가 리소스

- pmlogconf(1) 및 pmlogger(1) 도움말 페이지

- *PCP와 함께 배포된 툴*
- *PCP로 배포된 시스템 서비스*

6.2. PMLOGGER 구성 파일을 수동으로 편집

특정 지표 및 지정된 간격을 사용하여 맞춤형 로깅 구성을 생성하려면 pmlogger 구성 파일을 수동으로 편집합니다. 기본 pmlogger 설정 파일은 /var/lib/pcp/config/pmlogger/config.default 입니다. 구성 파일은 기본 로깅 인스턴스에서 로깅되는 메트릭을 지정합니다.

수동 구성에서 다음을 수행할 수 있습니다.

- *자동 구성에 나열되지 않은 지표를 기록합니다.*
- *사용자 정의 로깅 빈도를 선택합니다.*
- *애플리케이션 메트릭으로 PMDA 를 추가합니다.*

사전 요구 사항

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

절차

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

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

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

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

}

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

추가 리소스

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

6.3. PMLOGGER 서비스 활성화

pmlogger 서비스를 시작하고 로컬 시스템에서 지표 값을 로깅하도록 활성화해야 합니다.

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

사전 요구 사항

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

절차

- **pmlogger** 서비스를 시작하고 활성화합니다.

```
# systemctl start pmlogger

# systemctl enable pmlogger
```

검증 단계

- **pmlogger** 서비스가 활성화되어 있는지 확인합니다.

```
# pcp

Performance Co-Pilot configuration on workstation:

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

추가 리소스

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

6.4. 메트릭 컬렉션에 대한 클라이언트 시스템 설정

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

사전 요구 사항

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

절차

1. **pcp-system-tools** 패키지를 설치합니다.

```
# dnf install pcp-system-tools
```

2. **pmcd 의 IP 주소를 구성합니다.**

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

192.168.4.62 를 IP 주소로 교체하면 클라이언트가 수신 대기해야 합니다.

기본적으로 **pmcd** 는 **localhost**에서 수신 대기 중입니다.

3. 공용 영역을 영구적으로 추가하도록 방화벽을 구성합니다.

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

# firewall-cmd --reload
success
```

4. **SELinux 부울을 설정합니다.**

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

5. **pmcd 및 pmlogger 서비스를 활성화합니다.**

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

검증 단계

- **pmcd 가 구성된 IP 주소에서 올바르게 수신 대기 중인지 확인합니다.**

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

추가 리소스

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

6.5. 데이터 수집을 위해 중앙 서버 설정

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

사전 요구 사항

- *PCP가 설치되어 있습니다. 자세한 내용은 PCP 설치 및 활성화를 참조하십시오.*
- *지표 컬렉션에 대해 클라이언트가 구성됩니다. 자세한 내용은 메트릭 컬렉션의 클라이언트 시스템 설정을 참조하십시오.*

절차

1. *pcp-system-tools 패키지를 설치합니다.*

```
# dnf install pcp-system-tools
```

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

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

192.168.4.13 n n PCP_ARCHIVE_DIR/rhel7u4a -r -T24h10m -c config.rhel7u4a
192.168.4.14 n n PCP_ARCHIVE_DIR/rhel6u10a -r -T24h10m -c config.rhel6u10a
192.168.4.62 n n PCP_ARCHIVE_DIR/rhel8u1a -r -T24h10m -c config.rhel8u1a
192.168.4.69 n n PCP_ARCHIVE_DIR/rhel9u3a -r -T24h10m -c config.rhel9u3a
```

192.168.4.13,192.168.4.14,192.168.4.62 및 **192.168.4.69** 를 클라이언트 **IP** 주소로 교체합니다.

3. **pmcd** 및 **pmlogger** 서비스를 활성화합니다.

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

검증 단계

- 각 디렉터리에서 최신 아카이브 파일에 액세스할 수 있는지 확인합니다.

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

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

추가 리소스

- **pmlogger(1)** 도움말 페이지
- **PCP**와 함께 배포된 툴
- **PCP**로 배포된 시스템 서비스
- **/var/lib/pcp/config/pmlogger/config.default file**

6.6. PMREP로 PCP 로그 아카이브 재생

지표 데이터를 기록한 후 **PCP** 로그 아카이브를 재생할 수 있습니다. 로그를 텍스트 파일로 내보내고 서드 시트로 가져오려면 **pcp2csv,pcp2xml,pmrep** 또는 **pmlogsummary** 와 같은 **PCP** 유틸리티를 사용하십시오.

pmrep 툴을 사용하면 다음을 수행할 수 있습니다.

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

사전 요구 사항

- **PCP**가 설치되어 있습니다. 자세한 내용은 *PCP 설치 및 활성화를* 참조하십시오.
- **pmlogger** 서비스가 활성화되어 있습니다. 자세한 내용은 *pmlogger 서비스 활성화를* 참조하십시오.
- **pcp-system-tools** 패키지를 설치합니다.

```
# dnf install pcp-gui
```

절차

- 메트릭에 데이터를 표시합니다.

```
$ pmrep --start @3:00am --archive 20211128 --interval 5seconds --samples 10 --output csv
disk.dev.write
Time,"disk.dev.write-sda","disk.dev.write-sdb"
2021-11-28 03:00:00,,
2021-11-28 03:00:05,4.000,5.200
2021-11-28 03:00:10,1.600,7.600
2021-11-28 03:00:15,0.800,7.100
```

```
2021-11-28 03:00:20,16.600,8.400
2021-11-28 03:00:25,21.400,7.200
2021-11-28 03:00:30,21.200,6.800
2021-11-28 03:00:35,21.000,27.600
2021-11-28 03:00:40,12.400,33.800
2021-11-28 03:00:45,9.800,20.600
```

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

참고

*이 예제의 **20211128** 을 데이터 표시하려는 **pmlogger** 아카이브가 포함된 파일 이름으로 교체합니다.*

추가 리소스

- ***pmlogger(1), pmrep(1)** 및 **pmlogsummary(1)** 매뉴얼 페이지*
- *PCP와 함께 배포된 툴*
- *PCP로 배포된 시스템 서비스*

7장. PERFORMANCE CO-INSPECTOR를 사용하여 성능 모니터링

PCP(Performance Co-915)는 시스템 수준 성능 측정을 모니터링, 시각화, 저장 및 분석하기 위한 도구, 서비스 및 라이브러리 제품군입니다.

시스템 관리자는 **Red Hat Enterprise Linux 9**의 **PCP** 애플리케이션을 사용하여 시스템의 성능을 모니터링할 수 있습니다.

7.1. PMDA-JOURNALD로 POSTFIX 모니터링

이 절차에서는 **pmda- journald**로 **postfix** 메일 서버의 성능 지표를 모니터링하는 방법에 대해 설명합니다. 초당 받은 이메일 수를 확인하는 데 도움이 됩니다.

사전 요구 사항

- **PCP**가 설치되어 있습니다. 자세한 내용은 **PCP** 설치 및 활성화를 참조하십시오.
- **pmlogger** 서비스가 활성화되어 있습니다. 자세한 내용은 **pmlogger** 서비스 활성화를 참조하십시오.

절차

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

```
# dnf install pcp-system-tools
```

   b. **pmda-cnv** 패키지를 설치하여 **postfix** 를 모니터링합니다.

```
# dnf install pcp-pmda-postfix postfix
```

   c. 로깅 데몬을 설치합니다.

```
# dnf install rsyslog
```

d. *테스트를 위해 메일 클라이언트를 설치합니다.*

```
# dnf install mutt
```

2. ***postfix** 및 **rsyslog** 서비스를 활성화합니다.*

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

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

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

4. ***PMDA** 를 설치합니다.*

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

# ./Install

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

검증 단계

- ***pmda-octets** 작업을 확인합니다.*

```
echo testmail | mutt root
```

- *사용 가능한 지표를 확인합니다.*

```
# pminfo postfix

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

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

추가 리소스

- **rsyslogd(8), postfix(1) 및 setsebool(8) 도움말 페이지**
- **PCP와 함께 배포된 툴**
- **PCP로 배포된 시스템 서비스**
- **/var/lib/pcp/config/pmlogger/config.default file**

7.2. PCP 차트 애플리케이션을 사용하여 PCP 로그 아카이브를 시각적으로 추적

지표 데이터를 기록한 후 **PCP** 로그 아카이브를 그래프로 재생할 수 있습니다. 메트릭은 **PCP** 로그 아카이브의 지표 데이터를 기록 데이터의 소스로 사용하기 위한 대체 옵션으로 하나 이상의 라이브 호스트에서 가져옵니다. 성능 지표의 데이터를 표시하도록 **PCP** 차트 애플리케이션 인터페이스를 사용자 지정하려면 행 플롯, 막대 그래프 또는 사용률 그래프를 사용할 수 있습니다.

PCP 차트 애플리케이션을 사용하면 다음을 수행할 수 있습니다.

- **PCP** 차트 애플리케이션 애플리케이션에서 데이터를 리플레이하고 그래프를 사용하여 시스템의 실시간 데이터와 함께 소급 데이터를 시각화합니다.
- 성능 지표 값을 그래프로 플롯합니다.
- 여러 개의 차트를 동시에 표시합니다.

사전 요구 사항

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

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

- *pcp-gui 패키지를 설치합니다.*

```
# dnf install pcp-gui
```

절차

1. *명령 줄에서 PCP 차트 애플리케이션을 시작합니다.*

```
# pmchart
```

그림 7.1. PCP 차트 애플리케이션

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

- *PCP가 메트릭 데이터를 폴링하는 간격*
- *기록 데이터 지표의 날짜 및 시간*

2. *File (파일)을 클릭한 다음 New Chart (새 차트)를 클릭하여 호스트 이름 또는 주소를 지정하여 로컬 시스템과 원격 머신 모두에서 메트릭을 선택합니다. 고급 구성 옵션에는 차트의 축 값을*

수동으로 설정하고 플롯의 색상을 수동으로 선택할 수 있는 기능이 포함되어 있습니다.

3. **PCP** 차트 애플리케이션에 생성된 보기를 기록합니다.

   다음은 이미지를 사용하거나 **PCP** 차트 애플리케이션에 생성된 보기를 기록하는 옵션입니다.

   - 파일을 클릭한 다음 내보내기 를 클릭하여 현재 뷰의 이미지를 저장합니다.
   - 레코드 를 클릭한 다음 녹화를 시작하여 레코딩을 시작합니다. 레코드 를 클릭한 다음 중지 하여 레코딩을 중지합니다. 레코딩을 중지하면 기록된 지표가 나중에 볼 수 있도록 보관됩니다.

4. 선택 사항: **PCP** 차트 애플리케이션에서 보기 라고 하는 기본 구성 파일을 사용하면 하나 이상의 차트와 연결된 메타데이터를 저장할 수 있습니다. 이 메타데이터는 사용된 메트릭과 차트 열을 포함하여 모든 차트 측면을 설명합니다. **File (**파일)을 클릭한 다음 **Save View (** 보기 저장)를 클릭하여 사용자 지정 보기 구성을 저장하고 나중에 보기 구성을 로드합니다.

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

```
#kmchart
version 1

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

추가 리소스

- **pmchart(1)** 및 **pmtime(1)** 매뉴얼 페이지
- **PCP**와 함께 배포된 툴

7.3. PCP를 사용하여 SQL 서버에서 데이터 수집

SQL Server 에이전트는 데이터베이스 성능 문제를 모니터링하고 분석하는 데 도움이 되는 **PCP(Performance Co-inspector)**에서 사용할 수 있습니다.

*이 절차에서는 시스템의 **pcp** 를 통해 **Microsoft SQL Server**에 대한 데이터를 수집하는 방법을 설명합니다.*

사전 요구 사항

- ***Microsoft SQL Server for Red Hat Enterprise Linux**를 설치하고 **SQL** 서버에 대한 '신뢰' 연결을 설정했습니다.*
- ***SQL Server for Red Hat Enterprise Linux**용 **Microsoft QoS** 드라이버를 설치했습니다.*

절차

1. ***PCP**를 설치합니다.*

```
# dnf install pcp-zeroconf
```

2. ***pyodbc** 드라이버에 필요한 패키지를 설치합니다.*

```
# dnf install python3-pyodbc
```

3. ***mssql** 에이전트를 설치합니다.*

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

```
# dnf install pcp-pmda-mssql
```

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

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

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

4. *에이전트를 설치합니다.*

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

검증 단계

- ***pcp** 명령을 사용하여 **SQL Server PMDA(mssql)**가 로드되고 실행되고 있는지 확인합니다.*

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

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

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

```
# pminfo mssql
```

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

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

- ***pmchart** 명령을 사용하여 시스템에서 이러한 지표의 그래픽 차트를 확인합니다. 자세한 내용은 **PCP** 차트 애플리케이션을 사용하여 **PCP** 로그 아카이브 시각화를 참조하십시오.*

추가 리소스

- *PCP(1), pminfo(1), pmval(1), pmchart(1), pmdamssql(1) 도움말 페이지*

- *RHEL 8.2 Red Hat Developers 블로그의 Microsoft SQL Server용 Performance Co-databind for Microsoft SQL Server*

7.4. WEC 아카이브에서 PCP 아카이브 생성

NetNamespace 패키지에서 제공하는 슬픈f 도구를 사용하여 네이티브 슬라이 크 아카이브에서 PCP 아카이브를 생성할 수 있습니다.

사전 요구 사항

- *슬픈 아카이브가 생성되었습니다.*

```
# /usr/lib64/sa/sadc 1 5 -
```

이 예에서 슬픈 은 5초 간격으로 시스템 데이터를 샘플링합니다. outfile은 - 로 지정되며, 이로 인해 슬픈 이 매일 데이터 파일에 데이터를 표준 시스템 활동에 씁니다. 이 파일의 이름은 saDD이며 기본적으로 /var/log/sa 디렉터리에 있습니다.

절차

- *슬퍼크 아카이브에서 PCP 아카이브를 생성합니다.*

```
# sadf -l -O pcparchive=/tmp/recording -2
```

이 예에서 -2 옵션을 사용하면 슬픈f 가 2일 전에 기록된 슬픈 아카이브에서 PCP 아카이브를 생성합니다.

검증 단계

PCP 명령을 사용하면 네이티브 PCP 아카이브와 마찬가지로 슬픈 아카이브에서 생성된 PCP 아카이브를 검사하고 분석할 수 있습니다. 예를 들어 다음과 같습니다.

- *슬픈 아카이브 아카이브에서 생성된 PCP 아카이브에 메트릭 목록을 표시하려면 다음을 실행합니다.*

```
$ pminfo --archive /tmp/recording
Disk.dev.avactive
Disk.dev.read
Disk.dev.write
Disk.dev.blkread
[...]
```

- ***PCP** 아카이브의 아카이브 및 호스트 이름의 시간 공간을 표시하려면 다음을 실행합니다.*

```
$ pmdumplog --label /tmp/recording
Log Label (Log Format Version 2)
Performance metrics from host shard
        commencing Tue Jul 20 00:10:30.642477 2021
        ending     Wed Jul 21 00:10:30.222176 2021
```

- *성능 지표 값을 그래프로 분석하려면 다음을 실행합니다.*

```
$ pmchart --archive /tmp/recording
```

8장. PCP를 사용한 XFS의 성능 분석

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

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

8.1. XFS PMDA를 수동으로 설치

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

이 절차에서는 PMDA 에이전트를 수동으로 설치하는 방법을 설명합니다.

사전 요구 사항

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

절차

1. xfs 디렉터리로 이동합니다.

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

검증 단계

- pmcd 프로세스가 호스트에서 실행 중이고 XFS PMDA가 구성에 활성화된 것으로 나열되는지 확인합니다.

```
# pcp

Performance Co-Pilot configuration on workstation:

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

추가 리소스

- pmcd(1) 도움말 페이지
- PCP와 함께 배포된 툴

8.2. PMINFO를 사용하여 XFS 성능 지표 검사

PCP를 사용하면 XFS PMDA를 사용하여 마운트된 XFS 파일 시스템별로 특정 XFS 지표를 보고할 수 있습니다. 이렇게 하면 마운트된 특정 파일 시스템 문제를 더 쉽게 찾아 성능을 평가할 수 있습니다.

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

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

사전 요구 사항

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

절차

- XFS PMDA에서 제공하는 사용 가능한 모든 지표 목록을 표시합니다.

```
# pminfo xfs
```

- 개별 지표에 대한 정보를 표시합니다. 다음 예제에서는 pminfo 툴을 사용하여 특정 XFS에서 지표를 읽고 씁니다.
  - xfs.write_bytes 메트릭에 대한 간단한 설명을 표시합니다.

```
# pminfo --oneline xfs.write_bytes

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

  - xfs.read_bytes 메트릭에 대한 자세한 설명을 표시합니다.

```
# pminfo --helptext xfs.read_bytes

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

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

```
# pminfo --fetch xfs.read_bytes

xfs.read_bytes
    value 4891346238
```

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

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

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

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

추가 리소스

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

8.3. PMSTORE를 사용하여 XFS 성능 지표 재설정

PCP를 사용하면 특히 지표가 **xfs.control.reset** 지표와 같은 제어 변수 역할을 하는 경우 특정 지표의 값을 수정할 수 있습니다. 지표 값을 수정하려면 **pmstore** 툴을 사용합니다.

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

사전 요구 사항

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

절차

1. *지표 값을 표시합니다.*

```
$ pminfo -f xfs.write

xfs.write
    value 325262
```

2. *모든 **XFS** 지표를 재설정합니다.*

```
# pmstore xfs.control.reset 1

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

검증 단계

- *지표를 재설정한 후 정보를 확인합니다.*

```
$ pminfo --fetch xfs.write

xfs.write
    value 0
```

추가 리소스

- ***pmstore(1)** 및 **pminfo(1)** 매뉴얼 페이지*
- ***PCP**와 함께 배포된 툴*
- ***XFS**의 **PCP** 메트릭 그룹*

8.4. XFS의 PCP 메트릭 그룹

*다음 표에서는 **XFS**에 사용 가능한 **PCP** 지표 그룹을 설명합니다.*

*표 **8.1. XFS**의 지표 그룹*

메트릭 그룹	제공된 지표
xfs.*	읽기 및 쓰기 작업 수, 바이트 수 읽기 및 쓰기를 포함한 일반 XFS 지표. inode가 플러시되고 클러스터형 및 클러스터에 실패 횟수의 카운터와 함께 사용할 수 있습니다.
xfs.allocs.* **xfs.alloc_btree.***	파일 시스템에서 오브젝트 할당과 관련된 지표 범위에는 범위 및 블록 생성/없음이 포함됩니다. 할당 트리 조회 및 btree에서 레코드 생성 및 삭제 확장과 비교합니다.
xfs.block_map.* **xfs.bmap_btree.***	지표에는 블록 맵 읽기/쓰기 및 블록 삭제 수, 삽입 목록 작업, 삭제 및 조회가 포함됩니다. 또한 blockmap의 비교, 조회, 삽입 및 삭제 작업을 위한 작업 카운터도 있습니다.
xfs.dir_ops.*	생성, 입력 삭제, "getdent" 작업 수를 위한 XFS 파일 시스템의 디렉터리 작업에 대한 카운터입니다.
xfs.transactions.*	메타 데이터 트랜잭션 수에 대한 카운터에는 빈 트랜잭션 수와 함께 동기 및 비동기 트랜잭션 수가 포함됩니다.
xfs.inode_ops.*	운영 체제가 inode 캐시의 XFS inode를 찾는 횟수에 대한 카운터는 다양한 결과를 제공합니다. 이러한 개수 캐시 적중, 캐시 누락 등입니다.
xfs.log.* **xfs.log_tail.***	XFS 파일 암호화에 대한 로그 버퍼 수에 대한 카운터에는 디스크에 기록된 블록 수가 포함됩니다. 또한 로그 플러시 및 고정 수에 대한 메트릭입니다.
xfs.xstrat.*	XFS 플러시 중단으로 플러시한 파일 데이터의 바이트 수와 디스크의 연속 및 비동기 공간이 플러시되는 버퍼 수에 대한 카운터를 계산합니다.
xfs.attr.*	모든 XFS 파일 시스템에 대한 속성 get, set, remove 및 list 작업의 수를 계산합니다.
xfs.quota.*	XFS 파일 시스템에 대한 할당량 작업 지표에는 할당량 회수 횟수, 할당량 캐시 누락, 캐시 적중 및 할당량 데이터 회수에 대한 카운터가 포함됩니다.

xfs.buffer.*	XFS 버퍼 오브젝트와 관련된 지표 범위. 카운터에는 요청된 버퍼 호출 수, 성공한 버퍼 잠금, 대기 중인 버퍼 잠금, miss_locks, miss_retries 및 버퍼 적중 수가 포함됩니다.
xfs.btree.*	XFS btree의 작업에 관한 지표.
xfs.control.reset	XFS 통계의 지표 카운터를 재설정하는 데 사용되는 구성 지표입니다. 제어 메트릭은 pmstore 툴을 통해 전환됩니다.

8.5. XFS용 장치별 PCP 지표 그룹

*다음 표에서는 **XFS**에 대해 사용 가능한 장치별 **PCP** 지표 그룹을 설명합니다.*

표 8.2. XFS용 장치별 PCP 지표 그룹

메트릭 그룹	제공된 지표
xfs.perdev.*	읽기 및 쓰기 작업 수, 바이트 수 읽기 및 쓰기를 포함한 일반 XFS 지표. inode가 플러시되고 클러스터형 및 클러스터에 실패 횟수의 카운터와 함께 사용할 수 있습니다.
xfs.perdev.allocs.* **xfs.perdev.alloc_btree.***	파일 시스템에서 오브젝트 할당과 관련된 지표 범위에는 범위 및 블록 생성/없음이 포함됩니다. 할당 트리 조회 및 btree에서 레코드 생성 및 삭제 확장과 비교합니다.
xfs.perdev.block_map.* **xfs.perdev.bmap_btree.***	지표에는 블록 맵 읽기/쓰기 및 블록 삭제 수, 삽입 목록 작업, 삭제 및 조회가 포함됩니다. 또한 blockmap의 비교, 조회, 삽입 및 삭제 작업을 위한 작업 카운터도 있습니다.
xfs.perdev.dir_ops.*	생성, 입력 삭제, "getdent" 작업 수를 위한 XFS 파일 시스템의 디렉터리 작업에 대한 카운터입니다.
xfs.perdev.transactions.*	메타 데이터 트랜잭션 수에 대한 카운터에는 빈 트랜잭션 수와 함께 동기 및 비동기 트랜잭션 수가 포함됩니다.
xfs.perdev.inode_ops.*	운영 체제가 inode 캐시의 XFS inode를 찾는 횟수에 대한 카운터는 다양한 결과를 제공합니다. 이러한 개수 캐시 적중, 캐시 누락 등입니다.

xfs.perdev.log.* **xfs.perdev.log_tail.***	로그 버퍼 수에 대한 카운터는 XFS 파일SERVICEms를 통해 쓰는 블록 수가 디스크에 기록된 블록 수를 포함합니다. 또한 로그 플러시 및 고정 수에 대한 메트릭입니다.
xfs.perdev.xstrat.*	XFS 플러시 중단으로 플러시한 파일 데이터의 바이트 수와 디스크의 연속 및 비동기 공간이 플러시되는 버퍼 수에 대한 카운터를 계산합니다.
xfs.perdev.attr.*	모든 XFS 파일 시스템에 대한 속성 get, set, remove 및 list 작업의 수를 계산합니다.
xfs.perdev.quota.*	XFS 파일 시스템에 대한 할당량 작업 지표에는 할당량 회수 횟수, 할당량 캐시 누락, 캐시 적중 및 할당량 데이터 회수에 대한 카운터가 포함됩니다.
xfs.perdev.buffer.*	XFS 버퍼 오브젝트와 관련된 지표 범위. 카운터에는 요청된 버퍼 호출 수, 성공한 버퍼 잠금, 대기 중인 버퍼 잠금, miss_locks, miss_retries 및 버퍼 적중 수가 포함됩니다.
xfs.perdev.btree.*	XFS btree의 작업에 관한 지표.

9장. PCP 메트릭의 그래픽 표현 설정

pcp,grafana,pcp redis,pcp bpftrace 및 pcp 벡터 는 라이브 데이터 또는 PCP (Performance Co-inspector)에서 수집한 데이터 또는 데이터를 기반으로 그래프를 제공합니다.

이 섹션에서는 PCP 메트릭의 그래픽 표시를 설정하고 액세스하는 방법을 설명합니다.

9.1. PCP-ZEROCONF로 PCP 설정

이 절차에서는 pcp-zeroconf 패키지가 있는 시스템에서 PCP를 설정하는 방법을 설명합니다. pcp-zeroconf 패키지가 설치되면 시스템에서 기본 지표 세트를 아카이브 파일에 기록합니다.

절차

- pcp-zeroconf 패키지를 설치합니다.

```
# dnf install pcp-zeroconf
```

검증 단계

- pmlogger 서비스가 활성화되어 있는지 확인하고 메트릭 보관을 시작합니다.

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

추가 리소스

- pmlogger 도움말 페이지
- Performance Co-inspector를 사용하여 성능 모니터링

9.2. GRAFANA-SERVER 설정

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

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

사전 요구 사항

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

절차

1. 다음 패키지를 설치합니다.

```
# dnf install grafana grafana-pcp
```

2. 다음 서비스를 다시 시작하고 활성화합니다.

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

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

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

# firewall-cmd --reload
success
```

검증 단계

- **grafana-server** 가 수신 대기 중이며 요청에 응답하는지 확인합니다.

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

- **grafana-pcp** 플러그인이 설치되었는지 확인합니다.

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

performancecopilot-pcp-app @ 3.1.0
```

추가 리소스

- pmproxy(1) 및 grafana-server 도움말 페이지

9.3. GRAFANA 웹 UI 액세스

다음 절차에서는 Grafana 웹 인터페이스에 액세스하는 방법을 설명합니다.

Grafana 웹 인터페이스를 사용하면 다음을 수행할 수 있습니다.

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

사전 요구 사항

1. PCP가 구성되어 있습니다. 자세한 내용은 pcp-zeroconf를 사용하여 PCP 설정을 참조하십시오.
2. grafana-server 가 구성되어 있습니다. 자세한 내용은 grafana-server 설정에서 참조하십시오.

절차

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

   192.0.2.0 을 사용자 시스템 IP로 바꿉니다.

2.

*첫 번째 로그인의 경우 **Email** 또는 **username** 및 **Password** 필드에 **admin** 을 입력합니다.*

***Grafana**는 보안 계정을 생성하기 위해 새 암호를 설정하라는 메시지를 표시합니다. 나중에 설정하려는 경우 **Skip** 을 클릭합니다.*

3.

메뉴에서

구성 아이콘 위에 마우스를 이동한 다음 플러그인 을 클릭합니다.

4.

*플러그인 탭에서 이름 또는 유형 텍스트 상자에서 검색에 성능 **co-pilot**을 입력한 다음 **Performance Co- 915 (PCP)** 플러그인을 클릭합니다.*

5.

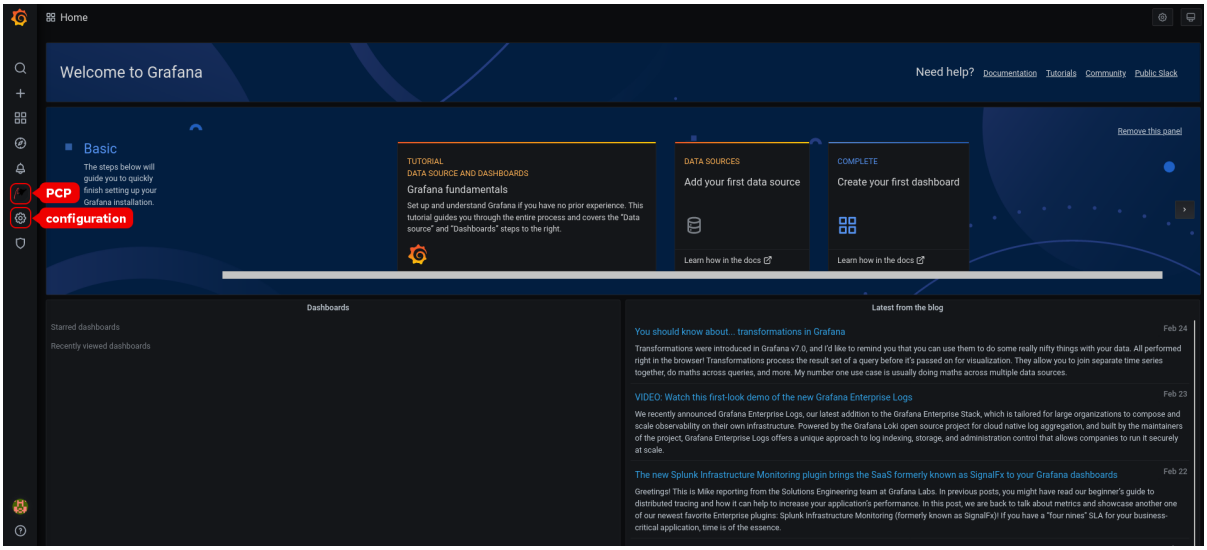
*플러그인 / **Performance Co-inspector** 창에서 사용을 클릭합니다.*

6.

Grafana

*아이콘을 클릭합니다. **Grafana** 홈 페이지가 표시됩니다.*

*그림 **9.1.** 홈 대시보드*



참고

화면의 상단 코너에는 유사한

아이콘이 있지만 일반 대시보드 설정을 제어합니다.

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

- pcp 빨간색is 데이터 소스를 추가하려면 기본 대시보드를 보고, 패널을 만들고 경고 규칙을 만들려면 PCP Redis 데이터 소스에서 패널 및 경고 생성을 참조하십시오.
- pcp bpftrace 데이터 소스를 추가하고 기본 대시보드를 보려면 PCP bpftrace System Analysis 대시보드 보기를 참조하십시오.
- pcp 벡터 데이터 소스를 추가하려면 기본 대시보드를 보고 벡터 체크리스트를 보려면 PCP 벡터 검사 목록 보기를 참조하십시오.

8. 선택 사항: 메뉴에서 admin 프로필

아이콘 위에 커서를 두고 프로필 편집,암호 변경 또는 로그아웃 을 포함한 기본 설정을 변경합니다.

추가 리소스

- Grafana-cli 및 grafana-server 도움말 페이지

9.4. PCP REDIS 구성

이 섹션에서는 PCP Redis 데이터 소스 설정에 대한 정보를 제공합니다.

PCP Redis 데이터 소스를 사용하여 다음을 수행합니다.

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

사전 요구 사항

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

절차

1. **redis** 패키지를 설치합니다.

```
# dnf install redis
```

2. 다음 서비스를 시작하고 활성화합니다.

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

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

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

allow_loading_unsigned_plugins = pcp-redis-datasource
```

5. **grafana-server** 를 다시 시작하십시오.

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

검증 단계

- ***pmproxy** 및 **redis** 가 작동하는지 확인합니다.*

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

*이 명령은 **redis** 패키지가 설치되지 않은 경우 데이터를 반환하지 않습니다.*

추가 리소스

- ***PMseries(1)** 도움말 페이지*

9.5. PCP REDIS 데이터 소스에서 패널 생성 및 경고

***PCP Redis** 데이터 소스를 추가한 후 유용한 지표에 대한 개요를 사용하여 대시보드를 보고, 로드 그래프를 시각화하는 쿼리를 추가하고, 시스템 문제가 발생한 후 시스템 문제를 확인하는 데 도움이 되는 경고를 생성할 수 있습니다.*

사전 요구 사항

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

절차

1. ***Grafana 웹 UI**에 로그인합니다.*
2. ***Grafana Home** 페이지에서 첫 번째 데이터 소스 추가를 클릭합니다.*
3. *데이터 소스 추가 창의 이름으로 필터링하거나 텍스트 유형으로 **redis**를 입력한 다음 **PCP Redis** 를 클릭합니다.*

4.

*데이터 소스 / **PCP Redis** 창에서 다음을 수행합니다.*

a.

***URL** 필드에 **http://localhost:44322** 을 추가한 다음 저장 및 테스트를 클릭합니다.*

b.

***Dashboards tab → Import → PCP Redis: Host Overview (PCP Redis:** 호스트 개요)를 클릭하여 유용한 지표의 개요가 포함된 대시보드를 확인합니다.*

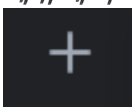
*그림 **9.2. PCP Redis:** 호스트 개요*

5.

새 패널을 추가합니다.

a.

메뉴에서



만들기 아이콘 → 새 패널 추가 아이콘을 클릭하여 패널을 추가합니다.

b.

*쿼리 탭에서 선택한 기본 옵션 대신 쿼리 목록에서 **PCP Redis** 를 선택하고 **A** 의 텍스트 필드에 **metric**를 입력합니다. 예를 들어 **kernel.all.load** 를 입력하여 커널 로드 그래프를 시각화합니다.*

c.

선택 사항: 패널 제목 과 설명 을 추가하고 설정에서 다른 옵션을 업데이트합니다.

d.

저장을 클릭하여 변경 사항을 적용하고 대시보드를 저장합니다. 대시보드 이름 추가 .

e.

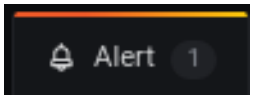
적용을 클릭하여 변경 사항을 적용하고 대시보드로 돌아갑니다.

그림 9.3. PCP Redis 쿼리 패널

6.

경고 규칙을 만듭니다.

a.

***PCP Redis** 쿼리 패널에서* 

***Alert** 를 클릭한 다음 알림 생성을 클릭합니다.*

b.

***Name** (이름),**Evaluate** 쿼리 (**Evalluate**) 쿼리 및 규칙의 경우 필드를 편집하고 경고에 대한 조건을 지정합니다.*

c.

저장을 클릭하여 변경 사항을 적용하고 대시보드를 저장합니다. 적용을 클릭하여 변경 사항을 적용하고 대시보드로 돌아갑니다.

그림 9.4. PCP Redis 패널에서 경고 생성

d.

선택 사항: 동일한 패널에서 아래로 스크롤하고 삭제 아이콘을 클릭하여 생성된 규칙을 삭제합니다.

e.

선택 사항: 메뉴에서

경고 아이콘을 클릭하여 다른 경고 상태로 생성된 경고 규칙을 보거나, 경고 규칙을 편집하거나 경고 규칙 탭에서 기존 규칙을 일시 중지합니다.

***Grafana**에서 경고 알림을 수신하는 생성된 경고 규칙에 대한 알림 채널을 추가하려면 알림에 대한 알림 채널 추가를 참조하십시오.*

9.6. 경고에 대한 알림 채널 추가

*알림 채널을 추가하면 경고 규칙 조건이 충족되고 시스템 모니터링이 필요할 때마다 **Grafana**에서 경고 알림을 수신할 수 있습니다.*

*지원되는 알림 목록에서 **DingDing,Discord,Email,Google Hangouts chat ,HipChat , LINE ,LINE LINE ,Microsoft Teams** 를 선택한 후 이러한 알림을 수신할 수 있습니다. **Opsgenie,PagerDuty,Prometheus Alertmanager,Pushover,Sensu,Slack, 전이ma 게이트웨이,VictorOps** 및 **Webhook.***

사전 요구 사항

1.

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

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

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

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

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

***abc@gmail.com** 을 유효한 이메일 주소로 바꿉니다.*

절차

1. *메뉴에서*

경고 아이콘 → 알림 채널 추가 채널 추가를클릭합니다.

2. *알림 채널 세부 정보 추가 창에서 다음을 수행합니다.*

   a. *이름 텍스트 상자에 이름을 입력합니다.*

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

   c. *선택 사항: 선택적 이메일 설정 및 알림 설정을 구성합니다.*

3. *저장을 클릭합니다.*

4. *경고 규칙에서 알림 채널을 선택합니다.*

   a. *메뉴에서*

경고 아이콘 위로 마우스를 이동한 다음 경고 규칙을 클릭합니다.

b. 경고 규칙 탭에서 생성된 경고 규칙을 클릭합니다.

c. 알림 탭의 **Send to** 옵션에서 알림 채널 이름을 선택한 다음 경고 메시지를 추가합니다.

d. 적용을 클릭합니다.

추가 리소스

- 경고 알림에 대한 업스트림 **Grafana** 문서

9.7. PCP 구성 요소 간 인증 설정

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

절차

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

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

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

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

mech_list: scram-sha-256

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

3. 새 사용자를 생성합니다.

```
# useradd -r metrics
```

지표를 사용자 이름으로 교체합니다.

4. *사용자 데이터베이스에 생성된 사용자를 추가합니다.*

```
# saslpasswd2 -a pmcd metrics

Password:
Again (for verification):
```

생성된 사용자를 추가하려면 지표 계정 암호를 입력해야 합니다.

5. *사용자 데이터베이스의 권한을 설정합니다.*

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

6. ***pmcd** 서비스를 다시 시작합니다.*

```
# systemctl restart pmcd
```

검증 단계

- ***sasl** 구성을 확인합니다.*

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

추가 리소스

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

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

9.8. PCP BPFTRACE 설치

PCP bpftrace 에이전트를 설치하여 시스템을 검사하고 커널 및 사용자 공간 추적점에서 지표를 수집합니다.

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

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

사전 요구 사항

1. **PCP**가 구성되어 있습니다. 자세한 내용은 **pcp-zeroconf**를 사용하여 **PCP** 설정을 참조하십시오.
2. **grafana-server** 가 구성되어 있습니다. 자세한 내용은 **grafana-server** 설정에서 참조하십시오.
3. **scram-sha-256** 인증 메커니즘이 구성되어 있습니다. 자세한 내용은 **PCP** 구성 요소 간 인증 설정을 참조하십시오.

절차

1. **pcp-pmda-bpftrace** 패키지를 설치합니다.

```
# dnf install pcp-pmda-bpftrace
```

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

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

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

지표를 사용자 이름으로 교체합니다.

3. **bpftrace PMDA**를 설치합니다.

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

pmda-bpftrace 가 이제 설치되었으며 사용자를 인증한 후에만 사용할 수 있습니다. 자세한 내용은 **PCP bpftrace** 시스템 분석 대시보드 보기를 참조하십시오.

추가 리소스

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

9.9. PCP BPFTRACE 시스템 분석 대시보드 보기

PCP bpftrace 데이터 소스를 사용하면 **pmlogger** 또는 아카이브에서 일반 데이터로 사용할 수 없는 소스의 실시간 데이터에 액세스할 수 있습니다.

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

사전 요구 사항

1. **PCP bpftrace**가 설치되어 있습니다. 자세한 내용은 **Installing PCP bpftrace** 를 참조하십시오.
2. **grafana-server** 에 액세스할 수 있습니다. 자세한 내용은 **Grafana 웹 UI** 액세스를 참조하십시오.

절차

1. **Grafana 웹 UI**에 로그인합니다.
2. **Grafana Home** 페이지에서 첫 번째 데이터 소스 추가를 클릭합니다.

3.

Add data source (데이터 소스 추가) 창에서 Filter by name or type text box에 bpftrace를 입력한 다음 PCP bpftrace 를 클릭합니다.

4.

*데이터 소스 / **PCP bpftrace** 창에서 다음을 수행합니다.*

a.

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

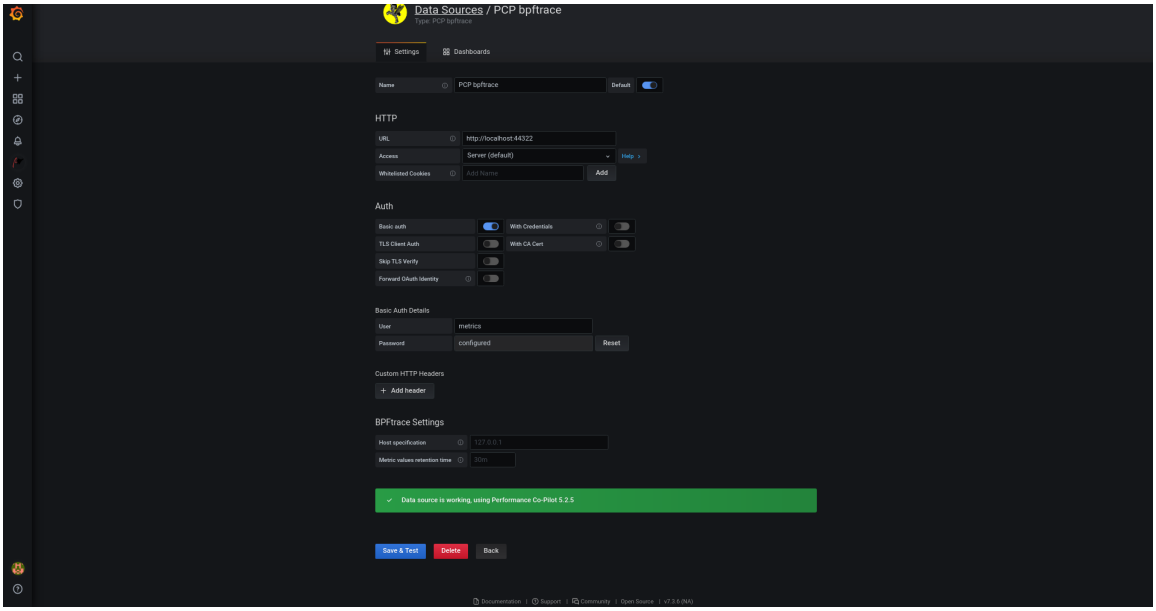
b.

***Basic Auth** 옵션을 전환하고 **User** 및 **Password** 필드에 생성된 사용자 자격 증명을 추가합니다.*

c.

***Save & Test** 를 클릭합니다.*

*그림 9.5. 데이터 소스에 **PCP bpftrace** 추가*



d.

***Dashboards** 탭 → **Import** → **PCP bpftrace: System Analysis** 를 클릭하여 유용한 지표에 대한 개요가 포함된 대시보드를 확인합니다.*

그림 9.6. PCP bpftrace: 시스템 분석

9.10. PCP 벡터 설치

*이 절차에서는 **pcp** 벡터 를 설치하는 방법을 설명합니다.*

사전 요구 사항

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

2. ***grafana-server** 가 구성되어 있습니다. 자세한 내용은 **grafana-server** 설정에서 참조하십시오.*

절차

1. ***pcp-pmda-bcc** 패키지를 설치합니다.*

```
# dnf install pcp-pmda-bcc
```

2. ***bcc PMDA**를 설치합니다.*

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

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

추가 리소스

- ***pmdabcc(1) man page***

9.11. PCP 벡터 체크리스트 보기

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

***PCP** 벡터 데이터 소스를 추가한 후 유용한 지표에 대한 개요를 사용하여 대시보드를 보고 체크리스트의 관련 문제 해결 또는 참조 링크를 볼 수 있습니다.*

사전 요구 사항

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

절차

1. ***Grafana 웹 UI**에 로그인합니다.*
2. ***Grafana Home** 페이지에서 첫 번째 데이터 소스 추가를 클릭합니다.*
3. *데이터 소스 추가 창의 이름으로 필터링하거나 텍스트 유형으로 벡터를 입력한 다음 **PCP** 벡터 를 클릭합니다.*
4. *데이터 소스 / **PCP** 벡터 창에서 다음을 수행합니다.*
   a. ***URL** 필드에 **http://localhost:44322** 을 추가한 다음 저장 및 테스트를 클릭합니다.*

URL 필드에 http://localhost:44322 을 추가한 다음 저장 및 테스트를 클릭합니다.

b.

Dashboards(대시보드) 탭 → Import → PCP Vector: Host Overview (PCP 벡터: 호스트 개요)를 클릭하여 유용한 지표에 대한 개요가 포함된 대시보드를 확인합니다.

그림 9.7. PCP 벡터: 호스트 개요

5.

메뉴에서

***Performance Co-inspector** 플러그인을 마우스로 이동한 다음 **PCP Vector Checklist** 를 클릭합니다.*

***PCP** 체크리스트에서*

***help** 또는*

경고 아이콘을 클릭하여 관련 문제 해결 또는 참조 링크를 확인합니다.

그림 9.8. Performance Co-915 / PCP Vector Checklist

9.12. GRAFANA 문제 해결

이 섹션에서는 **Grafana** 문제 (예: **Grafana**) 문제를 해결하는 방법에 대해 설명합니다.

절차

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

```
$ systemctl status pmlogger
```

- 다음 명령을 실행하여 파일이 디스크에 생성되거나 수정되었는지 확인합니다.

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

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

```
$ systemctl status pmproxy
```

- **pmproxy** 가 실행 중이고, 시계열 지원이 활성화되었으며 **/var/log/pcp/pmproxy/pmproxy.log** 파일을 보고 **Redis**에 대한 연결이 설정되어 있는지 확인하

고 다음 텍스트가 포함되어 있는지 확인합니다.

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

여기에서 **1716** 은 **pmproxy** 의 **PID**이며, **pmproxy**를 호출할 때마다 다릅니다.

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

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

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

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

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

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

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

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

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

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

10장. 웹 콘솔을 사용하여 시스템 성능 최적화

RHEL 웹 콘솔에서 성능 프로필을 설정하여 선택한 작업에 맞게 시스템의 성능을 최적화하는 방법을 알아봅니다.

10.1. 웹 콘솔의 성능 튜닝 옵션

Red Hat Enterprise Linux 9는 다음 작업을 위해 시스템을 최적화하는 여러 성능 프로필을 제공합니다.

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

TuneD 서비스는 선택한 프로필과 일치하도록 시스템 옵션을 최적화합니다.

웹 콘솔에서 시스템에서 사용하는 성능 프로필을 설정할 수 있습니다.

추가 리소스

- TuneD 시작하기

10.2. 웹 콘솔에서 성능 프로파일 설정

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

사전 요구 사항

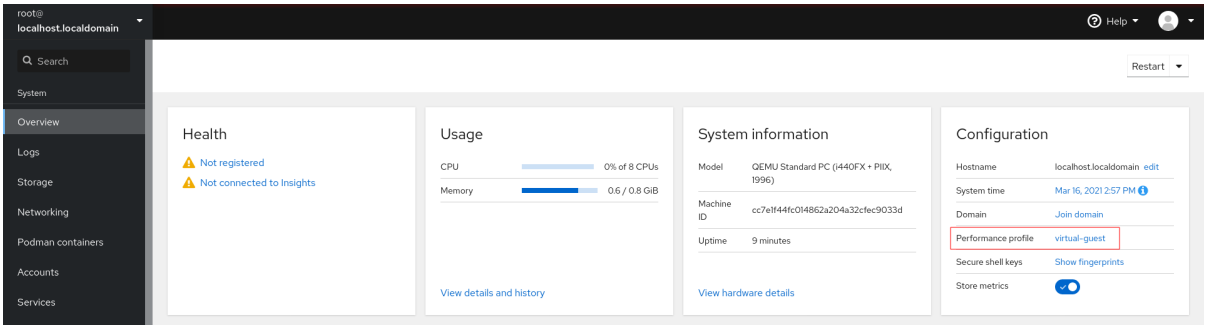
- *웹 콘솔이 설치되어 있고 액세스할 수 있는지 확인합니다. 자세한 내용은 웹 콘솔 설치를 참조하십시오.*

절차

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

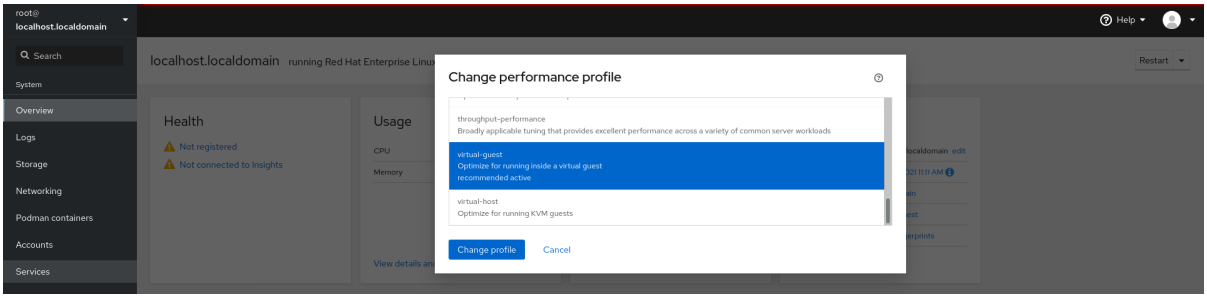
2. *개요 를 클릭합니다.*

3. *성능 프로필 필드에서 현재 성능 프로필을 클릭합니다.*



4. *성능 프로필 변경 대화 상자에서 필요한 경우 프로필을 변경합니다.*

5. *프로필 변경을 클릭합니다.*



검증 단계

- *이제 개요 탭에 선택한 성능 프로필이 표시됩니다.*

10.3. 웹 콘솔을 사용하여 성능 모니터링

***Red Hat**의 웹 콘솔은 문제 해결을 위해 **Utilization Saturation and Errors (USE)** 방식을 사용합니다. 새로운 성능 지표 페이지에는 데이터의 과거 보기가 고전적으로 맨 위에 있는 최신 데이터와 함께 체계화되어 있습니다.*

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

사전 요구 사항

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

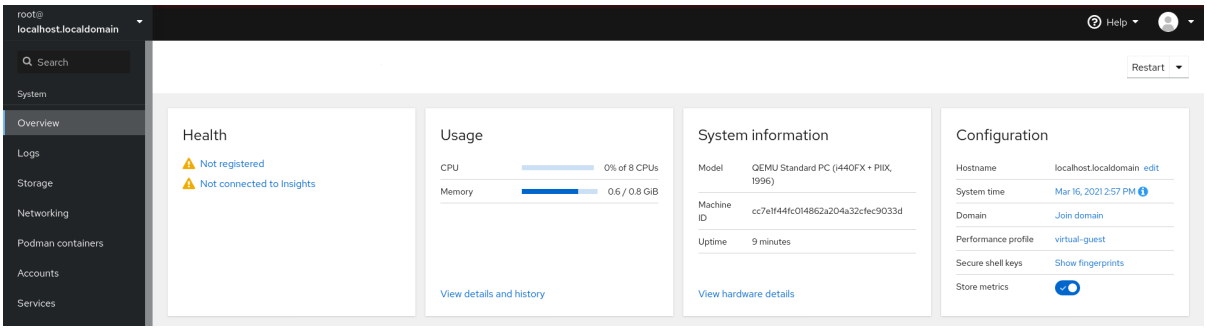
2. *성능 지표 수집을 활성화하는 **cockpit-pcp** 패키지를 설치합니다.*

```
# dnf install cockpit-pcp
```

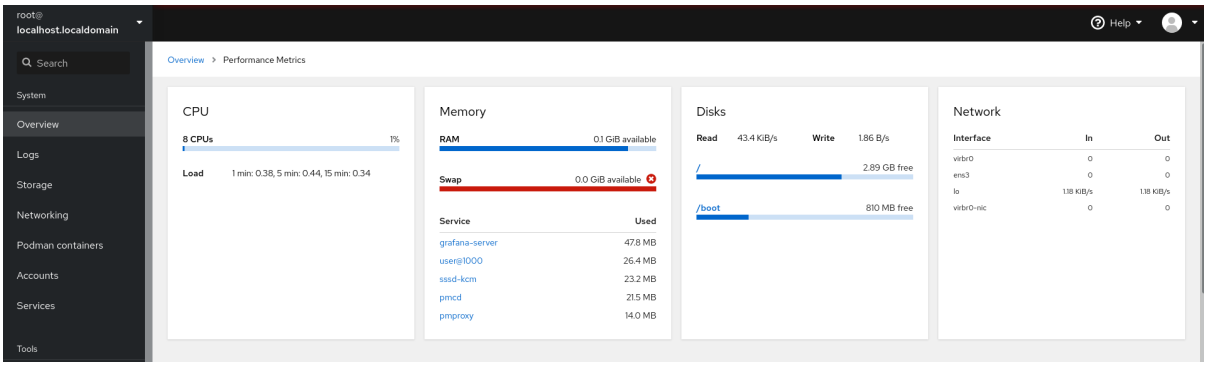
절차

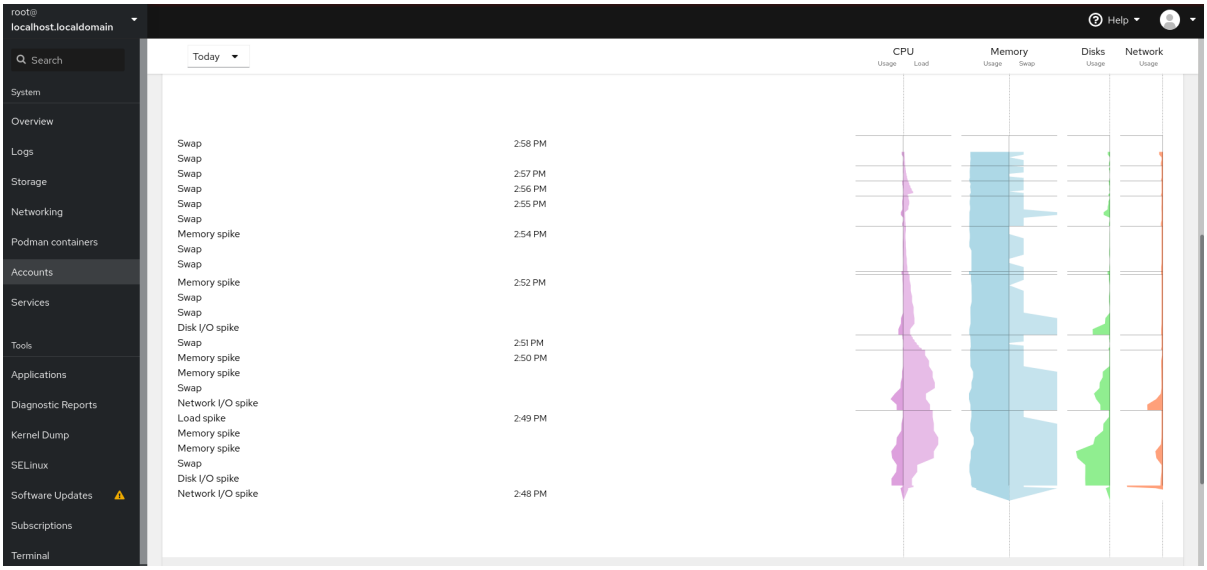
1. ***RHEL 9** 웹 콘솔에 로그인합니다. 자세한 내용은 웹 콘솔 로그인을 참조하십시오.*

2. *개요 를 클릭합니다.*



3. *세부 정보 보기 및 기록 을 클릭하여 성능 지표를 확인합니다.*



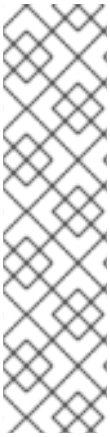


11장. 디스크 스케줄러 설정

디스크 스케줄러는 스토리지 장치에 전송된 I/O 요청을 정렬합니다.

다음과 같은 다양한 방법으로 스케줄러를 구성할 수 있습니다.

- TuneD 를 사용하여 디스크 스케줄러 설정에 설명된 대로 TuneD를 사용하여 스케줄러 설정
- udev 규칙을 사용하여 디스크 스케줄러 설정에 설명된 대로 udev를 사용하여 스케줄러 설정
- 일반적으로 특정 디스크에 대한 스케줄러 설정에 설명된 대로 실행 중인 시스템에서 스케줄러를 임시로변경합니다.



참고

Red Hat Enterprise Linux 9에서 블록 장치는 다중 대기열 스케줄링만 지원합니다. 이를 통해 솔리드 스테이트 드라이브(SSD) 및 멀티 코어 시스템으로 블록 계층 성능을 확장할 수 있습니다.

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

11.1. 사용 가능한 디스크 스케줄러

Red Hat Enterprise Linux 9에서 지원되는 다중 대기열 디스크 스케줄러는 다음과 같습니다.

none

first-in first-out(databind) 스케줄링 알고리즘을 구현합니다. 간단한 마지막 캐시를 통해 일반 블록 계층의 요청을 병합합니다.

mq-deadline

요청이 스케줄러에 도달하는 시점의 요청에 대해 보장되는 대기 시간을 제공하려고 합니다.

mq-deadline 스케줄러는 대기 중인 I/O 요청을 읽기 또는 쓰기 일괄 처리로 정렬한 다음 논리 블록 주소 지정(LBA) 순서를 늘리기 위해 해당 요청을 실행하도록 예약합니다. 기본적으로 읽기 배치는

애플리케이션이 읽기 I/O 작업에서 차단될 가능성이 높기 때문에 쓰기 일괄 처리보다 우선합니다. mq-deadline 이 배치를 처리한 후 쓰기 작업이 프로세서 시간을 능가하는 기간을 확인하고 필요에 따라 다음 읽기 또는 쓰기 배치를 예약합니다.

이 스케줄러는 대부분의 사용 사례에 적합하지만, 특히 쓰기 작업이 대부분 비동기인 경우도 있습니다.

bfq

데스크탑 시스템 및 대화형 작업을 대상으로 합니다.

bfq 스케줄러는 단일 애플리케이션이 모든 대역폭을 절대 사용하지 않도록 합니다. 실제로 스토리지 장치는 유휴 상태인 것처럼 항상 반응하는 것처럼 반응합니다. 기본 구성에서 bfq 는 최대 처리량을 달성하는 대신 가장 짧은 대기 시간을 제공하는 데 중점을 둡니다.

BFQ는 cfq 코드를 기반으로 합니다. 고정 시간 슬라이스의 각 프로세스에 디스크를 부여하지 않지만 섹터 수로 측정된 예산을 프로세스에 할당합니다.

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

kyber

스케줄러는 블록 I/O 계층에 제출된 모든 I/O 요청의 대기 시간을 계산하여 대기 시간 목표를 달성하기 위해 자체적으로 튜닝합니다. 캐시 허용 및 동기 쓰기 요청의 경우 읽기, 대상 대기 시간을 구성할 수 있습니다.

이 스케줄러는 NVMe, SSD 또는 기타 낮은 대기 시간 장치와 같은 빠른 장치에 적합합니다.

11.2. 다른 사용 사례에 대한 다양한 디스크 스케줄러

시스템에서 수행하는 작업에 따라 분석 및 튜닝 작업 전에 다음 디스크 스케줄러가 기준선으로 권장됩니다.

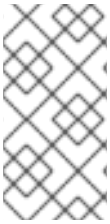
표 11.1. 다양한 사용 사례의 디스크 스케줄러

사용 사례	디스크 스케줄러
SCSI 인터페이스가 있는 기존HD	**mq-deadline** 또는 **bfq** 를 사용합니다.

사용 사례	디스크 스케줄러
고성능 SSD 또는 빠른 스토리지가 있는 CPU 바인딩 시스템	특히 엔터프라이즈 애플리케이션을 실행할 때 **아무 것도 사용하지 않습니다**. 또는 **kyber** 를 사용합니다.
데스크탑 또는 대화형 작업	**bfq** 를 사용합니다.
가상 게스트	**mq-deadline** 을 사용합니다. 멀티 대기열이 가능한 호스트 버스 어댑터(HBA) 드라이버를 사용하면 **아무 것도 사용하지 않습니다**.

11.3. 기본 디스크 스케줄러

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



참고

***NUMA (Non-volatile Memory Express)** 블록 장치의 경우 기본 스케줄러는 **none** 이며 **Red Hat**은 이를 변경하지 않는 것이 좋습니다.*

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

11.4. 활성 디스크 스케줄러 확인

이 절차에서는 지정된 블록 장치에서 현재 활성화되어 있는 디스크 스케줄러를 결정합니다.

절차

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

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

[mq-deadline] kyber bfq none
```

*파일 이름에서 장치를 블록 장치 이름으로 바꿉니다(예: **sdc**).*

활성 스케줄러는 대괄호([ ])로 나열됩니다.

11.5. TUNED를 사용하여 디스크 스케줄러 설정

이 절차에서는 선택한 블록 장치에 지정된 디스크 스케줄러를 설정하는 TuneD 프로필을 생성하고 활성화합니다. 시스템 재부팅 시 설정이 유지됩니다.

다음 명령 및 구성에서 다음을 교체합니다.

- 블록 장치 의 이름이 있는 장치(예: sdf)
- 장치에 설정하려는 디스크 스케줄러가 있는 선택된- scheduler(예: bfq)

사전 요구 사항

- TuneD 서비스가 설치되고 활성화됩니다. 자세한 내용은 TuneD 설치 및 활성화를 참조하십시오.

절차

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

   현재 활성화되어 있는 프로필을 확인하려면 다음을 사용하십시오.

   ```
   $ tuned-adm active
   ```

2. TuneD 프로필을 저장할 새 디렉터리를 생성합니다.

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

3. 선택한 블록 장치의 시스템 고유 식별자를 찾습니다.

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

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

참고

이 예제의 명령은 지정된 블록 장치와 연결된 **WWN(WWN)** 또는 일련 번호로 식별되는 모든 값을 반환합니다. **WWN**을 사용하는 것이 바람직하지만, 지정된 장치에 **WWN**을 항상 사용할 수는 없으며 **example** 명령에서 반환된 모든 값은 장치 시스템 고유 **ID** 로 사용할 수 있습니다.

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

a. 선택 사항: 기존 프로필이 포함되어 있습니다.

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

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

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

여기:

- **IDNAME** 을 사용 중인 식별자 이름(예: **ID_WWN**)으로 바꿉니다.
- 장치 시스템 고유 **ID** 를 선택한 식별자의 값으로 바꿉니다(예: **0x5002538d00000000**).

devices_udev_regex 옵션의 여러 장치와 일치하려면 식별자를 괄호로 묶고 세로 막대로 분리합니다.

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

5. 프로필을 활성화합니다.

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

검증 단계

1. **TuneD** 프로필이 활성화되어 적용되는지 확인합니다.

```
$ tuned-adm active

Current active profile: my-profile
```

```
$ tuned-adm verify

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

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

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

[mq-deadline] kyber bfq none
```

파일 이름에서 장치를 블록 장치 이름으로 바꿉니다(예: **sdc**).

활성 스케줄러는 대괄호(**[]**)로 나열됩니다.

추가 리소스

- **TuneD** 프로필 사용자 정의 .

11.6. UDEV 규칙을 사용하여 디스크 스케줄러 설정

이 절차에서는 **udev** 규칙을 사용하여 특정 블록 장치에 대해 지정된 디스크 스케줄러를 설정합니다. 시스템 재부팅 시 설정이 유지됩니다.

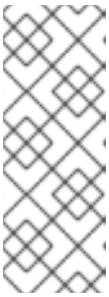
다음 명령 및 구성에서 다음을 교체합니다.

- *블록 장치 의 이름이 있는 장치(예: **sdf**)*
- *장치에 설정하려는 디스크 스케줄러가 있는 선택된**- scheduler(예: bfq)***

절차

1. *블록 장치의 시스템 고유 식별자를 찾습니다.*

```
$ udevadm info --name=/dev/device | grep -E '(WWN|SERIAL)'
E: ID_WWN=0x5002538d00000000
E: ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
E: ID_SERIAL_SHORT=20120501030900000
```



참고

*이 예제의 명령은 지정된 블록 장치와 연결된 **WWN(WWN)** 또는 일련 번호로 식별되는 모든 값을 반환합니다. **WWN**을 사용하는 것이 바람직하지만, 지정된 장치에 **WWN**을 항상 사용할 수는 없으며 **example** 명령에서 반환된 모든 값은 장치 시스템 고유 **ID** 로 사용할 수 있습니다.*

2. ***udev** 규칙을 구성합니다. 다음 콘텐츠를 사용하여 **/etc/udev/rules.d/99-scheduler.rules** 파일을 생성합니다.*

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{IDNAME}=="device system unique id", ATTR{queue/scheduler}="selected-scheduler"
```

여기:

- ***IDNAME** 을 사용 중인 식별자 이름(예: **ID_WWN**)으로 바꿉니다.*
- *장치 시스템 고유 **ID** 를 선택한 식별자의 값으로 바꿉니다(예: **0x5002538d00000000**).*

3. ***udev** 규칙을 다시 로드합니다.*

```
# udevadm control --reload-rules
```

4. *스케줄러 구성을 적용합니다.*

```
# udevadm trigger --type=devices --action=change
```

검증 단계

- *활성 스케줄러를 확인합니다.*

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

11.7. 특정 디스크에 대한 스케줄러를 임시로 설정

이 절차에서는 특정 블록 장치에 대해 지정된 디스크 스케줄러를 설정합니다. 시스템 재부팅 시 설정이 유지되지 않습니다.

절차

- *선택한 스케줄러의 이름을 **/sys/block/device/queue/scheduler** 파일에 씁니다.*

```
# echo selected-scheduler > /sys/block/device/queue/scheduler
```

*파일 이름에서 장치를 블록 장치 이름으로 바꿉니다(예: **sdc**).*

검증 단계

- *장치에서 스케줄러가 활성화되어 있는지 확인합니다.*

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

12장. SAMBA 서버의 성능 튜닝

이 장에서는 특정 상황에서 Samba의 성능을 향상시킬 수 있는 설정과 성능에 부정적인 영향을 미칠 수 있는 설정에 대해 설명합니다.

사전 요구 사항

- Samba가 파일 또는 인쇄 서버로 설정

12.1. SMB 프로토콜 버전 설정

각각의 새로운 SMB 버전은 기능을 추가하고 프로토콜의 성능을 향상시킵니다. 최근의 Windows 및 Windows Server 운영 체제는 항상 최신 프로토콜 버전을 지원합니다. 또한 Samba가 최신 프로토콜 버전을 사용하는 경우 Samba에 연결된 Windows 클라이언트는 성능 개선의 이점을 누릴 수 있습니다. Samba에서 서버 max 프로토콜의 기본값은 지원되는 최신 stable SMB 프로토콜 버전으로 설정됩니다.

참고

항상 안정적인 최신 SMB 프로토콜 버전을 사용하려면 server max protocol 매개 변수를 설정하지 마십시오. 매개 변수를 수동으로 설정하는 경우 최신 프로토콜 버전을 사용하도록 새 버전의 SMB 프로토콜을 사용하여 설정을 수정해야 합니다.

다음 절차에서는 server max protocol 매개 변수에서 기본값을 사용하는 방법을 설명합니다.

절차

1. /etc/ controlPlane/octets.conf 파일의 [global] 섹션에서 server max protocol 매개 변수를 제거합니다.

2. Samba 구성 다시 로드

```
# smbcontrol all reload-config
```

12.2. 많은 수의 파일이 포함된 디렉터리와의 공유 튜닝

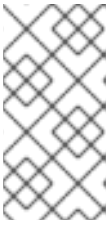
Linux는 대/소문자를 구분하지 않는 파일 이름을 지원합니다. 따라서 파일을 검색하거나 액세스할 때 Samba에서 대문자 및 소문자 파일 이름의 디렉터리를 스캔해야 합니다. 소문자 또는 대문자에서만 새 파일을 만들어 성능을 개선하도록 공유를 구성할 수 있습니다.

사전 요구 사항

- Samba가 파일 서버로 구성되어 있습니다.

절차

1. 공유의 모든 파일의 이름을 소문자로 바꿉니다.



참고

이 절차의 설정을 사용하여 소문자가 아닌 이름이 있는 파일은 더 이상 표시되지 않습니다.

2. 공유 섹션에서 다음 매개변수를 설정합니다.

```
case sensitive = true
default case = lower
preserve case = no
short preserve case = no
```

매개변수에 대한 자세한 내용은 rootfs .conf(5) 매뉴얼 페이지에서 해당 설명을 참조하십시오.

3. /etc/samba/smb.conf 파일을 확인합니다.

```
# testparm
```

4. Samba 구성을 다시 로드합니다.

```
# smbcontrol all reload-config
```

*이러한 설정을 적용한 후 이 공유에서 새로 생성된 모든 파일의 이름은 소문자를 사용합니다. 이러한 설정으로 인해 **Samba**는 더 이상 대문자 및 소문자로 디렉터리를 스캔할 필요가 없으므로 성능이 향상됩니다.*

12.3. 성능에 부정적인 영향을 줄 수 있는 설정

*기본적으로 **Red Hat Enterprise Linux**의 커널은 높은 네트워크 성능을 위해 조정됩니다. 예를 들어 커널은 버퍼 크기에 자동 튜닝 메커니즘을 사용합니다. **/etc/ controlPlane/octets.conf** 파일에서 **socket options** 매개 변수를 설정하면 이러한 커널 설정이 재정의됩니다. 결과적으로 이 매개 변수를 설정하면 대부분의 경우 **Samba** 네트워크 성능이 저하됩니다.*

*커널에서 최적화된 설정을 사용하려면 **/etc/ controlPlane/octets.conf**의 **[global]** 섹션에서 **socket options** 매개 변수를 제거합니다.*

13장. 가상 머신 성능 최적화

가상 머신(VM)은 항상 호스트와 비교하여 어느 정도 성능 저하가 발생합니다. 다음 섹션에서는 이러한 완화의 이유를 설명하고 RHEL 9에서 가상화의 성능 영향을 최소화하는 방법에 대한 지침을 제공하여 하드웨어 인프라 리소스를 최대한 효율적으로 사용할 수 있도록 합니다.

13.1. 가상 머신 성능에 미치는 영향

VM은 호스트에서 사용자 공간 프로세스로 실행됩니다. 따라서 하이퍼바이저는 VM이 사용할 수 있도록 호스트의 시스템 리소스를 변환해야 합니다. 그 결과 변환에서 리소스 일부를 사용하고 VM은 호스트와 동일한 성능 효율성을 달성할 수 없습니다.

가상화가 시스템 성능에 미치는 영향

VM 성능 저하에 대한 구체적인 이유는 다음과 같습니다.

- 가상 CPU(vCPU)는 호스트에서 스레드로 구현되며 Linux 스케줄러에서 처리합니다.
- VM은 호스트 커널에서 NUMA 또는 대규모 페이지와 같은 최적화 기능을 자동으로 상속하지 않습니다.
- 호스트의 디스크 및 네트워크 I/O 설정이 VM에 상당한 성능 영향을 미칠 수 있습니다.
- 일반적으로 네트워크 트래픽은 소프트웨어 기반 브릿지를 통해 VM으로 이동합니다.
- 호스트 장치와 해당 모델에 따라 특정 하드웨어 에뮬레이션으로 인해 상당한 오버헤드가 발생할 수 있습니다.

VM 성능에 미치는 가상화의 심각도는 다음과 같은 다양한 요인의 영향을 받습니다.

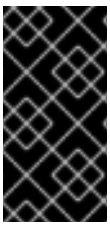
- 동시에 실행 중인 VM 수입니다.
- 각 VM에서 사용하는 가상 장치의 양입니다.

- VM에서 사용하는 장치 유형입니다.

VM 성능 손실 감소

RHEL 9는 가상화의 부정적인 성능 영향을 줄이는 데 사용할 수 있는 다양한 기능을 제공합니다. 특히:

- TuneD 서비스는 VM의 리소스 배포 및 성능을 자동으로 최적화할 수 있습니다.
- 블록 I/O 튜닝 은 디스크와 같은 VM 블록 장치의 성능을 향상시킬 수 있습니다.
- NUMA 튜닝 은 vCPU 성능을 향상시킬 수 있습니다.
- 가상 네트워킹을 다양한 방법으로 최적화할 수 있습니다.



중요

VM 성능 튜닝은 다른 가상화 기능에 부정적인 영향을 미칠 수 있습니다. 예를 들어, 수정된 VM을 마이그레이션하는 것이 더 어려워질 수 있습니다.

13.2. TUNED를 사용하여 가상 머신 성능 최적화

TuneD 유틸리티는 CPU 집약적인 작업 또는 스토리지 네트워크 처리량 응답성 요구 사항과 같은 특정 워크로드 특성에 RHEL을 조정하는 튜닝 프로필 전달 메커니즘입니다. 성능을 향상시키고 여러 특정 사용 사례에서 전력 소비를 줄이기 위해 사전 구성된 여러 튜닝 프로필을 제공합니다. 이러한 프로필을 편집하거나 새로운 프로필을 생성하여 가상화된 환경을 포함하여 사용자 환경에 맞는 성능 솔루션을 생성할 수 있습니다.

가상화를 위해 RHEL 9를 최적화하려면 다음 프로필을 사용하십시오.

- RHEL 9 가상 머신의 경우 virtual-guest 프로필을 사용합니다. 일반적으로 적용 가능한 throughput-performance 프로필을 기반으로 하지만 가상 메모리의 스왑도 감소합니다.
- RHEL 9 가상화 호스트의 경우 virtual-host 프로필을 사용합니다. 이렇게 하면 더 공격적으로 더티 메모리 페이지의 쓰기를 수행할 수 있으므로 호스트 성능이 향상됩니다.

사전 요구 사항

- *TuneD 서비스가 설치되고 활성화됩니다.*

절차

특정 TuneD 프로필을 활성화하려면 다음을 수행합니다.

1. *사용 가능한 TuneD 프로필을 나열합니다.*

```
# tuned-adm list

Available profiles:
- balanced             - General non-specialized TuneD profile
- desktop              - Optimize for the desktop use-case
[...]
- virtual-guest        - Optimize for running inside a virtual guest
- virtual-host         - Optimize for running KVM guests
Current active profile: balanced
```

2. *선택 사항: 새 TuneD 프로필을 생성하거나 기존 TuneD 프로필을 편집합니다.*

   자세한 내용은 TuneD 프로필 사용자 지정을 참조하십시오.

3. *TuneD 프로필을 활성화합니다.*

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

   - *가상화 호스트를 최적화하려면 virtual-host 프로필을 사용합니다.*

```
# tuned-adm profile virtual-host
```

   - *RHEL 게스트 운영 체제에서 virtual-guest 프로필을 사용합니다.*

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

추가 리소스

- *시스템 상태 및 성능 모니터링 및 관리*

13.3. LIBVIRT 데몬 최적화

libvirt 가상화 제품군은 RHEL 하이퍼바이저의 관리 계층으로 작동하며 libvirt 구성이 가상화 호스트에 큰 영향을 미칩니다. 특히 RHEL 9에는 두 가지 유형의 libvirt 데몬, 모놀리식 또는 모듈식이 포함되어 있으며 사용하는 데몬 유형이 개별 가상화 드라이버를 구성하는 방법에 미치는 영향에 영향을 미칩니다.

13.3.1. libvirt 데몬 유형

RHEL 9는 다음과 같은 libvirt 데몬 유형을 지원합니다.

모놀리식 libvirt

기존 libvirt 데몬 libvirtd 는 단일 구성 파일인 /etc/libvirt/libvirtd.conf 를 사용하여 다양한 가상화 드라이버를 제어합니다.

따라서 libvirtd 는 중앙 집중식 하이퍼바이저 구성을 허용하지만 시스템 리소스를 효율적으로 사용할 수 있습니다. 따라서 libvirtd 는 향후 RHEL의 주요 릴리스에서 지원되지 않습니다.

그러나 RHEL 8에서 RHEL 9로 업데이트한 경우에도 호스트는 기본적으로 libvirtd 를 사용합니다.

모듈식 libvirt

RHEL 9에 새로 도입된 모듈식 libvirt 는 각 가상화 드라이버에 대해 특정 데몬을 제공합니다. 여기에는 다음이 포함됩니다.

- *virtqemud - 하이퍼바이저 관리를 위한 기본 데몬*
- *virtinterfaced - 호스트 NIC 관리를 위한 보조 데몬*
- *virtnetworkd - 가상 네트워크 관리를 위한 보조 데몬*
- *virtnodedevd - 호스트 물리적 장치 관리를 위한 보조 데몬*

- *virtnwfilterd - 호스트 방화벽 관리를 위한 보조 데몬*
- *virtsecretd - 호스트 보안 관리를 위한 보조 데몬*
- *virtstoraged - 스토리지 관리를 위한 보조 데몬*

각 데몬에는 별도의 구성 파일(예: /etc/libvirt/virtqemud.conf)이 있습니다. 따라서 모듈식 libvirt 데몬은 libvirt 리소스 관리를 세부적으로 조정할 수 있는 더 나은 옵션을 제공합니다.

RHEL 9 새로 설치를 수행한 경우 모듈식 libvirt 가 기본적으로 구성됩니다.

다음 단계

- *RHEL 9에서 libvirtd 를 사용하는 경우 Red Hat은 모듈식 데몬으로 전환하는 것이 좋습니다. 자세한 내용은 모듈식 libvirt 데몬 활성화를 참조하십시오.*

13.3.2. 모듈식 libvirt 데몬 활성화

RHEL 9에서 libvirt 라이브러리는 호스트의 개별 가상화 드라이버 세트를 처리하는 모듈식 데몬을 사용합니다. 예를 들어 virtqemud 데몬은 QEMU 드라이버를 처리합니다.

RHEL 9 호스트 새로 설치를 수행한 경우 하이퍼바이저는 기본적으로 모듈식 libvirt 데몬을 사용합니다. 그러나 호스트를 RHEL 8에서 RHEL 9로 업그레이드하는 경우 하이퍼바이저는 RHEL 8의 기본값인 모놀리식 libvirtd 데몬을 사용합니다.

이 경우 libvirt 리소스 관리에 더 나은 옵션을 제공하기 때문에 Red Hat은 대신 모듈식 libvirt 데몬을 활성화할 것을 권장합니다. 또한 libvirtd 는 향후 RHEL의 주요 릴리스에서 지원되지 않습니다.

사전 요구 사항

- *하이퍼바이저는 모놀리식 libvirtd 서비스를 사용합니다. 이 경우인지 확인하려면 다음을 수행하십시오.*

```
# systemctl is-active libvirtd.service
active
```

- 이 명령이 활성 으로 표시되면 **libvirtd** 를 사용합니다.

- 가상 머신이 종료되었습니다.

절차

1. **libvirtd** 및 해당 소켓을 중지합니다.

```
# systemctl stop libvirtd.service
# systemctl stop libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

2. **libvirtd** 를 비활성화하여 부팅 시 시작되지 않도록 합니다.

```
$ systemctl disable libvirtd.service
$ systemctl disable libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

3. 모듈식 **libvirt** 데몬을 활성화합니다.

```
# for drv in qemu interface network nodedev nwfilter secret storage; do systemctl
unmask virt${drv}d.service; systemctl unmask virt${drv}d{,-ro,-admin}.socket;
systemctl enable virt${drv}d.service; systemctl enable virt${drv}d{,-ro,-admin}.socket;
done
```

4. 모듈식 데몬의 소켓을 시작합니다.

```
# for drv in qemu network nodedev nwfilter secret storage; do systemctl start
virt${drv}d{,-ro,-admin}.socket; done
```

5. 선택 사항: 원격 호스트에서 호스트에 연결해야 하는 경우 가상화 프록시 데몬을 활성화하고 시작합니다.

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin,-tls}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin,-tls}.socket
# systemctl start virtproxyd{,-ro,-admin,-tls}.socket
```

검증

1. 활성화된 가상화 데몬을 활성화합니다.

```
# virsh uri
qemu:///system
```

2. 호스트가 **virtqemud** 모듈식 데몬을 사용하고 있는지 확인합니다.

```
# systemctl is-active virtqemud.service
active
```

이 명령이 활성 상태로 표시되면 모듈식 **libvirt** 데몬을 성공적으로 활성화했습니다.

13.4. 가상 머신 메모리 구성

VM(가상 머신)의 성능을 개선하기 위해 VM에 추가 호스트 RAM을 할당할 수 있습니다. 마찬가지로 VM에 할당된 메모리 양을 감소시켜 호스트 메모리를 다른 VM 또는 작업에 할당할 수 있습니다.

이러한 작업을 수행하려면 웹 콘솔 또는 명령줄 인터페이스를 사용할 수 있습니다.

13.4.1. 웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거

VM(가상 머신)의 성능을 개선하거나 사용 중인 호스트 리소스를 확보하기 위해 웹 콘솔을 사용하여 VM에 할당된 메모리 양을 조정할 수 있습니다.

사전 요구 사항

- 게스트 OS는 메모리 balloon 드라이버를 실행하고 있습니다. 이 경우를 확인하려면 다음을 수행하십시오.

  1. VM 구성에 memballoon 장치가 포함되어 있는지 확인합니다.

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

  이 명령이 모든 출력을 표시하고 모델이 none 으로 설정되지 않은 경우 memballoon 장치가 있습니다.

2. 게스트 OS에서 balloon 드라이버가 실행 중인지 확인합니다.

   - Windows 게스트에서 드라이버는 virtio-win 드라이버 패키지의 일부로 설치됩니다. 자세한 내용은 Windows 가상 머신용 반가상화 KVM 드라이버 설치를 참조하십시오.

   - Linux 게스트에서는 일반적으로 드라이버가 기본적으로 포함되어 있으며 memballoon 장치가 있을 때 활성화됩니다.

- 웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.

절차

1. 선택 사항: 최대 메모리 및 VM에 현재 사용된 메모리에 대한 정보를 획득합니다. 이는 변경 사항에 대한 기준선 역할을 하고 확인을 위한 역할을 합니다.

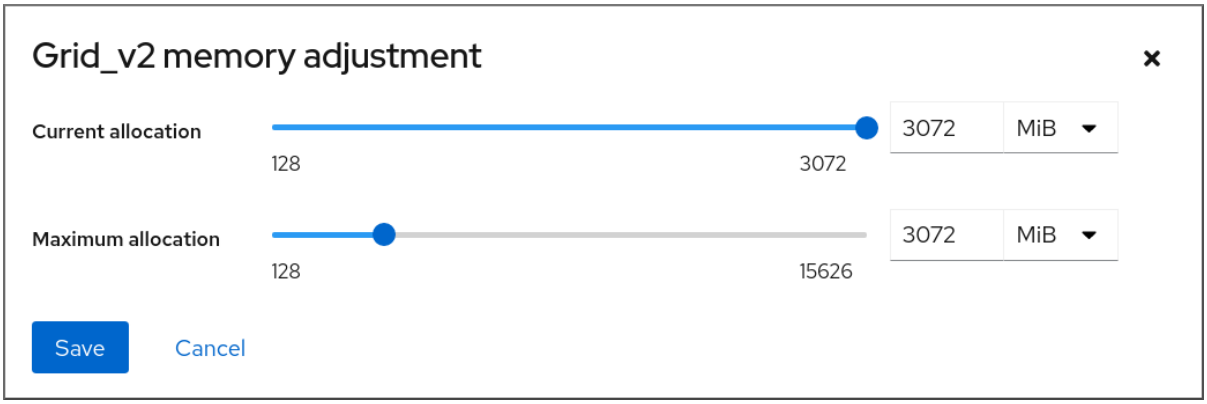
```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. 가상 머신 인터페이스에서 정보를 확인할 VM을 클릭합니다.

   선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 개요 섹션이 있는 새 페이지가 열리고 VM의 그래픽 인터페이스에 액세스합니다.

3. 개요 창에서 Memory 행 옆에 있는 편집 을 클릭합니다.

   메모리 조정 대화 상자가 표시됩니다.



4. 선택한 VM의 가상 CPU를 구성합니다.

- 최대 할당 - VM에서 프로세스에 사용할 수 있는 최대 호스트 메모리 양을 설정합니다. VM을 생성할 때 최대 메모리를 지정하거나 나중에 늘릴 수 있습니다. 메모리를 다수의 MiB 또는 GiB로 지정할 수 있습니다.

  최대 메모리 할당 조정은 종료된 VM에서만 가능합니다.

- 현재 할당 - VM에 할당된 실제 메모리 양을 설정합니다. 이 값은 최대 할당보다 작을 수 있지만 초과할 수는 없습니다. 해당 프로세스에 대해 VM에서 사용할 수 있는 메모리를 규제하도록 값을 조정할 수 있습니다. 메모리를 다수의 MiB 또는 GiB로 지정할 수 있습니다.

  이 값을 지정하지 않으면 기본 할당은 최대 할당 값입니다.

5. 저장을 클릭합니다.

  VM의 메모리 할당이 조정됩니다.

추가 리소스

- 명령줄 인터페이스를 사용하여 가상 머신 메모리 추가 및 제거
- 가상 머신 CPU 성능 최적화

13.4.2. 명령줄 인터페이스를 사용하여 가상 머신 메모리 추가 및 제거

VM(가상 머신)의 성능을 개선하거나 사용 중인 호스트 리소스를 확보하기 위해 CLI를 사용하여 VM에 할당된 메모리 양을 조정할 수 있습니다.

사전 요구 사항

- *게스트 OS는 메모리 balloon 드라이버를 실행하고 있습니다. 이 경우를 확인하려면 다음을 수행하십시오.*

1. *VM 구성에 memballoon 장치가 포함되어 있는지 확인합니다.*

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

이 명령이 모든 출력을 표시하고 모델이 none 으로 설정되지 않은 경우 memballoon 장치가 있습니다.

2. *게스트 OS에서 ballon 드라이버가 실행 중인지 확인합니다.*

   - *Windows 게스트에서 드라이버는 virtio-win 드라이버 패키지의 일부로 설치됩니다. 자세한 내용은 Windows 가상 머신용 반가상화 KVM 드라이버 설치를 참조하십시오.*

   - *Linux 게스트에서는 일반적으로 드라이버가 기본적으로 포함되어 있으며 memballoon 장치가 있을 때 활성화됩니다.*

절차

1. *선택 사항: 최대 메모리 및 VM에 현재 사용된 메모리에 대한 정보를 획득합니다. 이는 변경 사항에 대한 기준선 역할을 하고 확인을 위한 역할을 합니다.*

```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. *VM에 할당된 최대 메모리를 조정합니다. 이 값을 늘리면 VM의 성능 가능성이 향상되고, VM이 호스트에 있는 성능 풋프린트를 줄일 수 있습니다. 이 변경 사항은 종료된 VM에서만 수행할 수 있으므로 실행 중인 VM을 조정하는 작업을 적용하려면 재부팅이 필요합니다.*

예를 들어 testguest VM에서 사용할 수 있는 최대 메모리를 4096MiB로 변경하려면 다음을 수행합니다.

```
# virt-xml testguest --edit --memory memory=4096,currentMemory=4096
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

실행 중인 VM의 최대 메모리를 늘리려면 메모리 장치를 VM에 연결할 수 있습니다. 이를 메모리 핫 플러그 라고도 합니다. 자세한 내용은 가상 머신에 장치 연결을 참조하십시오.

주의

실행 중인 VM(메모리 핫 플러그라고도 함)에서 메모리 장치를 제거하는 것은 지원되지 않으며 Red Hat의 디스크가 높습니다.

3. 선택 사항: VM에서 현재 사용하는 메모리를 최대 할당까지 조정할 수도 있습니다. 이렇게 하면 최대 VM 할당을 변경하지 않고 다음 재부팅까지 VM에 있는 메모리 로드가 조정됩니다.

```
# virsh setmem testguest --current 2048
```

검증

1. VM에서 사용하는 메모리가 업데이트되었는지 확인합니다.

```
# virsh dominfo testguest
Max memory:     4194304 KiB
Used memory:    2097152 KiB
```

2. 선택 사항: 현재 VM 메모리를 조정한 경우 VM의 메모리 balloon 통계를 가져와 메모리 사용을 얼마나 효과적으로 조정할 수 있습니다.

```
 # virsh domstats --balloon testguest
Domain: 'testguest'
  balloon.current=365624
  balloon.maximum=4194304
  balloon.swap_in=0
  balloon.swap_out=0
  balloon.major_fault=306
  balloon.minor_fault=156117
  balloon.unused=3834448
```

```
balloon.available=4035008
balloon.usable=3746340
balloon.last-update=1587971682
balloon.disk_caches=75444
balloon.hugetlb_pgalloc=0
balloon.hugetlb_pgfail=0
balloon.rss=1005456
```

추가 리소스

- 웹 콘솔을 사용하여 가상 머신 메모리 추가 및 제거
- 가상 머신 CPU 성능 최적화

13.4.3. 추가 리소스

- 장치를 가상 머신에 연결하는 가상 머신에 장치 연결.

13.5. 가상 머신 I/O 성능 최적화

VM(가상 머신)의 입력 및 출력(I/O) 기능은 VM의 전반적인 효율성을 크게 제한할 수 있습니다. 이를 해결하기 위해 블록 I/O 매개변수를 구성하여 VM의 I/O를 최적화할 수 있습니다.

13.5.1. 가상 머신에서 블록 I/O 튜닝

하나 이상의 VM에서 여러 블록 장치를 사용하는 경우 I/O 가중치 를 수정하여 특정 가상 장치의 I/O 우선 순위를 조정하는 것이 중요할 수 있습니다.

장치의 I/O 가중치를 늘리면 I/O 대역폭에 대한 우선 순위가 증가하므로 더 많은 호스트 리소스가 제공됩니다. 마찬가지로 장치의 가중치를 줄이면 호스트 리소스를 적게 사용합니다.

참고

각 장치의 가중치 값은 100 ~1000 범위 내에 있어야 합니다. 또는 값은 0 일 수 있으며 장치별 목록에서 해당 장치를 제거할 수 있습니다.

절차

VM의 블록 I/O 매개변수를 표시하고 설정하려면 다음을 수행합니다.

1. VM의 현재 <blkio> 매개변수를 표시합니다.

```
# virsh dumpxml VM-name
```

```
<domain>
 [...]
 <blkiotune>
  <weight>800</weight>
  <device>
   <path>/dev/sda</path>
   <weight>1000</weight>
  </device>
  <device>
   <path>/dev/sdb</path>
   <weight>500</weight>
  </device>
 </blkiotune>
 [...]
</domain>
```

2. 지정된 장치의 I/O 가중치를 편집합니다.

```
# virsh blkiotune VM-name --device-weights device, I/O-weight
```

예를 들어, 다음에서는 리프트 VM의 /dev/sda 장치의 가중치를 500으로 변경합니다.

```
# virsh blkiotune liftbrul --device-weights /dev/sda, 500
```

13.5.2. 가상 머신에서 디스크 I/O 제한

여러 VM이 동시에 실행되는 경우 과도한 디스크 I/O를 사용하여 시스템 성능을 방해할 수 있습니다. KVM 가상화의 디스크 I/O 제한을 사용하면 VM에서 호스트 시스템으로 전송된 디스크 I/O 요청에 제한을 설정할 수 있습니다. 이를 통해 VM이 공유 리소스를 과도하게 활용하는 것을 방지하고 다른 VM의 성능에 영향을 미칠 수 있습니다.

디스크 I/O 제한을 활성화하려면 VM에 연결된 각 블록 장치에서 호스트 시스템에 전송된 디스크 I/O 요청 제한을 설정합니다.

절차

1. *virsh domblklist 명령을 사용하여 지정된 VM에 있는 모든 디스크 장치의 이름을 나열합니다.*

```
# virsh domblklist rollin-coal
Target     Source
------------------------------------------------
vda        /var/lib/libvirt/images/rollin-coal.qcow2
sda        -
sdb        /home/horridly-demanding-processes.iso
```

2. *추적할 가상 디스크가 마운트된 호스트 블록 장치를 찾습니다.*

예를 들어 이전 단계에서 sdb 가상 디스크를 제한하려는 경우 다음 출력에서는 디스크가 /dev/nvme0n1p3 파티션에 마운트되었음을 보여줍니다.

```
$ lsblk
NAME                                          MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
zram0                                         252:0    0     4G  0 disk  [SWAP]
nvme0n1                                       259:0    0 238.5G  0 disk
├─nvme0n1p1                                   259:1    0   600M  0 part  /boot/efi
├─nvme0n1p2                                   259:2    0     1G  0 part  /boot
└─nvme0n1p3                                   259:3    0 236.9G  0 part
  └─luks-a1123911-6f37-463c-b4eb-fxzy1ac12fea 253:0    0 236.9G  0 crypt /home
```

3. *virsh blkiotune 명령을 사용하여 블록 장치에 대한 I/O 제한을 설정합니다.*

```
# virsh blkiotune VM-name --parameter device,limit
```

다음 예제에서는 Rollin-coal VM의 sdb 디스크를 초당 1000개의 읽기 및 쓰기 I/O 작업과 초당 50MB 읽기 및 쓰기 처리량을 제한합니다.

```
# virsh blkiotune rollin-coal --device-read-iops-sec /dev/nvme0n1p3,1000 --device-write-iops-sec /dev/nvme0n1p3,1000 --device-write-bytes-sec /dev/nvme0n1p3,52428800 --device-read-bytes-sec /dev/nvme0n1p3,52428800
```

추가 정보

- *디스크 I/O 제한은 여러 고객에 속한 VM이 동일한 호스트에서 실행 중이거나 다른 VM에 대해 서비스 품질 보장이 제공되는 경우와 같이 다양한 상황에서 유용할 수 있습니다. 디스크 I/O 제한을 사용하여 느린 디스크를 시뮬레이션할 수도 있습니다.*
- *I/O 제한은 VM에 연결된 각 블록 장치에 독립적으로 적용할 수 있으며 처리량 및 I/O 작업에*

대한 제한을 지원합니다.

- Red Hat은 virsh blkdeviotune 명령을 사용하여 VM에서 I/O 제한을 구성할 수 없습니다. RHEL 9를 VM 호스트로 사용할 때 지원되지 않는 기능에 대한 자세한 내용은 RHEL 9 가상화의 지원되지 않는 기능을 참조하십시오.

13.5.3. 다중 대기열 virtio-scsi 활성화

VM(가상 머신)에서 virtio-scsi 스토리지 장치를 사용하는 경우 다중 대기열 virtio-scsi 기능은 향상된 스토리지 성능과 확장성을 제공합니다. 이를 통해 각 가상 CPU(vCPU)는 다른 vCPU에 영향을 주지 않고 별도의 대기열과 인터럽트를 사용할 수 있습니다.

절차

- 특정 VM에 대해 다중 대기열 virtio-scsi 지원을 활성화하려면 VM의 XML 구성에 다음을 추가합니다. 여기서 N 은 총 vCPU 대기열 수입니다.

```
<controller type='scsi' index='0' model='virtio-scsi'>
  <driver queues='N' />
</controller>
```

13.6. 가상 머신 CPU 성능 최적화

호스트 시스템의 물리적 CPU와 마찬가지로 vCPU는 VM(가상 머신) 성능에 매우 중요합니다. 따라서 vCPU 최적화는 VM의 리소스 효율성에 상당한 영향을 미칠 수 있습니다. vCPU를 최적화하려면 다음을 수행합니다.

1. VM에 할당되는 호스트 CPU 수를 조정합니다. CLI 또는 웹 콘솔 을 사용하여 이 작업을 수행할 수 있습니다.

2. vCPU 모델이 호스트의 CPU 모델에 맞게 조정되었는지 확인합니다. 예를 들어, 호스트의 CPU 모델을 사용하도록 testguest1 VM을 설정하려면 다음을 수행합니다.

```
# virt-xml testguest1 --edit --cpu host-model
```

3. 커널 동일한 페이지 병합(KSM)을 관리합니다.

4. 호스트 시스템에서 NUMA(Non-Uniform Memory Access)를 사용하는 경우 해당 VM에 대

해 NUMA를 구성할 수도 있습니다. 이렇게 하면 호스트의 CPU 및 메모리 프로세스가 VM의 CPU 및 메모리 프로세스에 최대한 밀접하게 매핑됩니다. 실제로 NUMA 튜닝은 vCPU에 할당된 시스템 메모리에 보다 간소화된 액세스를 제공하여 vCPU 처리 효율성을 향상시킬 수 있습니다.

자세한 내용은 가상 머신의 NUMA 구성 및 Sample vCPU 성능 튜닝 시나리오를 참조하십시오.

13.6.1. 명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거

VM(가상 머신)의 CPU 성능을 늘리거나 최적화하기 위해 VM에 할당된 가상 CPU(vCPU)를 추가하거나 제거할 수 있습니다.

실행 중인 VM에서 수행할 때 이 작업을 vCPU 핫 플러그링 및 핫 플러그 연결이라고도 합니다. 그러나 vCPU 핫 플러그는 RHEL 9에서 지원되지 않으며 Red Hat은 사용이 좋지 않습니다.

사전 요구 사항

- 선택 사항: 대상 VM에서 vCPU의 현재 상태를 확인합니다. 예를 들어 testguest VM에 vCPU 수를 표시하려면 다음을 수행합니다.

```
# virsh vcpucount testguest
maximum      config         4
maximum      live           2
current      config         2
current      live           1
```

이 출력은 현재 testguest 가 1 vCPU를 사용하고 있으며 VM 성능을 높이기 위해 1개의 vCPu를 핫 플러그로 연결할 수 있음을 나타냅니다. 그러나 재부팅 후 testguest 의 수는 2로 변경되며 두 개의 vCPU를 핫 플러그로 변경할 수 있습니다.

절차

1. VM에 연결할 수 있는 최대 vCPU 수를 조정하여 VM의 다음 부팅에 적용됩니다.

   예를 들어 testguest VM의 최대 vCPU 수를 8로 늘리려면 다음을 수행합니다.

```
# virsh setvcpus testguest 8 --maximum --config
```

CPU 토폴로지, 호스트 하드웨어, 하이퍼바이저 및 기타 요인에 의해 최대값이 제한될 수 있습니다.

2. VM에 연결된 현재 vCPU 수를 이전 단계에서 구성한 최대값까지 조정합니다. 예를 들어 다음과 같습니다.

- 실행 중인 testguest VM에 연결된 vCPU 수를 4로 늘리려면 다음을 수행합니다.

```
# virsh setvcpus testguest 4 --live
```

이렇게 하면 VM의 다음 부팅이 될 때까지 VM의 성능 및 호스트 로드 공간이 testguest의 성능 및 호스트 로드가 증가합니다.

- testguest VM에 연결된 vCPU 수를 1로 영구적으로 줄입니다.

```
# virsh setvcpus testguest 1 --config
```

이렇게 하면 VM의 다음 부팅 후 VM의 성능 및 호스트 로드 풋프린트가 감소됩니다. 하지만 필요한 경우 추가 vCPU를 VM에 핫 플러그하여 일시적으로 성능을 향상시킬 수 있습니다.

검증

- VM의 현재 vCPU 상태가 변경 사항을 반영하는지 확인합니다.

```
# virsh vcpucount testguest
maximum      config         8
maximum      live           4
current      config         1
current      live           4
```

추가 리소스

- 웹 콘솔을 사용하여 가상 CPU 관리

13.6.2. 웹 콘솔을 사용하여 가상 CPU 관리

RHEL 9 웹 콘솔을 사용하면 웹 콘솔이 연결된 VM(가상 머신)에서 사용하는 가상 CPU를 검토하고 구성할 수 있습니다.

사전 요구 사항

- *웹 콘솔 VM 플러그인이 시스템에 설치되어 있습니다.*

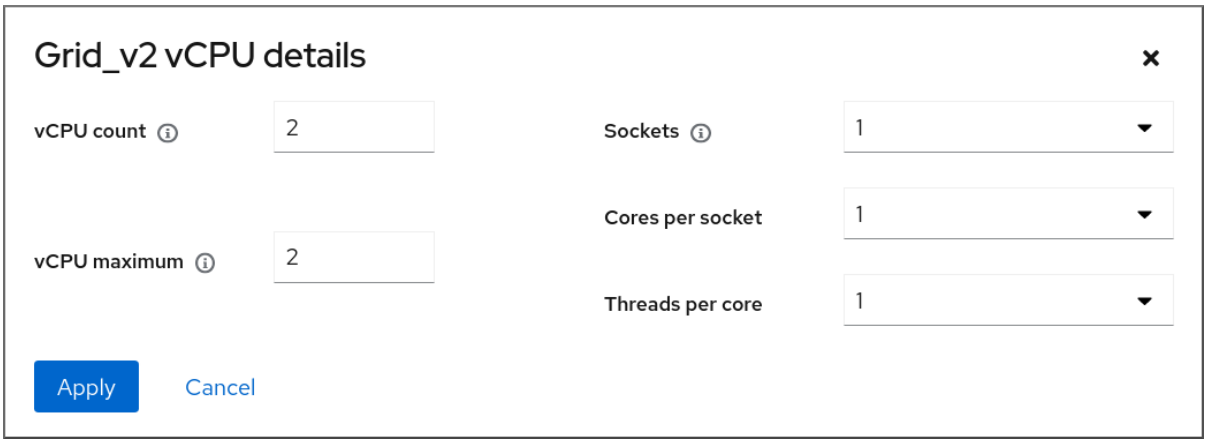
절차

1. *가상 머신 인터페이스에서 정보를 확인할 VM을 클릭합니다.*

   선택한 VM 및 콘솔 섹션에 대한 기본 정보가 포함된 개요 섹션이 있는 새 페이지가 열리고 VM의 그래픽 인터페이스에 액세스합니다.

2. *개요 창에서 vCPU 수 옆에 있는 편집 을 클릭합니다.*

   vCPU 세부 정보 대화 상자가 나타납니다.



1. *선택한 VM의 가상 CPU를 구성합니다.*

   - *vCPU 수 - 현재 사용 중인 vCPU 수입니다.*

     참고

     vCPU 수는 vCPU 최대값보다 클 수 없습니다.

   - *vCPU 최대 - VM에 대해 구성할 수 있는 가상 CPU의 최대 수입니다. 이 값이 vCPU 개수보다 높으면 추가 vCPU 를 VM에 연결할 수 있습니다.*

- *sockets - VM에 노출하는 소켓 수입니다.*

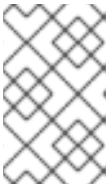
- *코어당 코어 수 - VM에 노출하는 각 소켓의 코어 수입니다.*

- *코어당 스레드 수 - VM에 노출될 각 코어의 스레드 수입니다.*

소켓,소켓당 코어 수 및 코어 옵션당 스레드 는 VM의 CPU 토폴로지를 조정합니다. 이는 vCPU 성능에 유용할 수 있으며 게스트 OS에서 특정 소프트웨어의 기능에 영향을 미칠 수 있습니다. 배포에 다른 설정이 필요하지 않은 경우 기본값을 유지합니다.

2. *적용을 클릭합니다.*

VM의 가상 CPU가 구성됩니다.



참고

가상 CPU 설정 변경 사항은 VM을 재시작한 후에만 적용됩니다.

추가 리소스

- *명령줄 인터페이스를 사용하여 가상 CPU 추가 및 제거*

13.6.3. 가상 머신에서 NUMA 구성

다음 방법을 사용하여 RHEL 9 호스트에서 가상 머신(VM)의 NUMA(Non-Uniform Memory Access) 설정을 구성할 수 있습니다.

사전 요구 사항

- *호스트는 NUMA 호환 시스템입니다. 대소문자를 감지하려면 virsh nodeinfo 명령을 사용하여 NUMA 셀 을 확인합니다.*

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              48
CPU frequency:       1200 MHz
CPU socket(s):       1
```

```
Core(s) per socket:  12
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         67012964 KiB
```

행 값이 2 이상인 경우 호스트는 NUMA와 호환됩니다.

절차

쉽게 사용할 수 있도록 자동화된 유틸리티 및 서비스를 사용하여 VM의 NUMA 구성을 설정할 수 있습니다. 그러나 수동 NUMA 설정은 성능이 크게 향상될 가능성이 높습니다.

자동 방법

- VM의 NUMA 정책을 Preferred 로 설정합니다. 예를 들어 testguest5 VM에 대해 이렇게 하려면 다음을 수행합니다.

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- 호스트에서 자동 NUMA 분산을 활성화합니다.

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- numad 명령을 사용하여 VM CPU를 메모리 리소스와 자동으로 조정합니다.

```
# numad
```

수동 방법

1. 특정 vCPU 스레드를 특정 호스트 CPU 또는 CPU 범위에 고정합니다. 이는 NUMA 이외의 호스트 및 VM에서도 가능하며 안전한 vCPU 성능 향상 방법으로 권장됩니다.

   예를 들어 다음 명령은 vCPU 스레드 0~5를 testguest6 VM의 호스트 CPU 1, 3, 5, 7, 9, 11에 고정합니다.

```
# virsh vcpupin testguest6 0 1
# virsh vcpupin testguest6 1 3
```

```
# virsh vcpupin testguest6 2 5
# virsh vcpupin testguest6 3 7
# virsh vcpupin testguest6 4 9
# virsh vcpupin testguest6 5 11
```

그런 다음 이것이 성공했는지 확인할 수 있습니다.

```
# virsh vcpupin testguest6
VCPU   CPU Affinity
----------------------
0      1
1      3
2      5
3      7
4      9
5      11
```

2. vCPU 스레드를 고정한 후 지정된 VM과 연결된 QEMU 프로세스 스레드를 특정 호스트 CPU 또는 CPU 범위에 고정할 수도 있습니다. 예를 들어 다음 명령은 testguest6 의 QEMU 프로세스 스레드를 CPU 13 및 15에 고정하고 성공적으로 완료되었는지 확인합니다.

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
----------------------------------
       *: 13,15
```

3. 마지막으로 특정 VM에 구체적으로 할당할 호스트 NUMA 노드를 지정할 수도 있습니다. 이를 통해 VM의 vCPU에 의해 호스트 메모리 사용량을 개선할 수 있습니다. 예를 들어 다음 명령은 호스트 NUMA 노드 3을 5로 사용하도록 testguest6 를 설정하고 성공적인지 확인합니다.

```
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```

참고

최상의 성능 결과를 얻으려면 위에 나열된 수동 튜닝 방법을 모두 사용하는 것이 좋습니다.

확인된 문제

- 현재 IBM Z 호스트에서 NUMA 튜닝을 수행할 수 없습니다.

추가 리소스

- 샘플 vCPU 성능 튜닝 시나리오
- numastat 유틸리티를 사용하여 시스템의 현재 NUMA 구성 보기

13.6.4. 샘플 vCPU 성능 튜닝 시나리오

최상의 vCPU 성능을 얻으려면 다음 시나리오와 같이 수동 vcpupin,emulatorpin 및 numatune 설정을 함께 사용하는 것이 좋습니다.

시작 시나리오

- 호스트에는 다음과 같은 하드웨어 관련 사항이 있습니다.
  - NUMA 노드 2개
  - 각 노드의 CPU 코어 3개
  - 각 코어의 2개 스레드

  이러한 머신의 virsh nodeinfo 출력은 다음과 유사합니다.

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              12
CPU frequency:       3661 MHz
CPU socket(s):       2
Core(s) per socket:  3
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         31248692 KiB
```

- 기존 VM을 8개의 vCPU를 사용하도록 수정하려고 합니다. 즉, 단일 NUMA 노드에 맞지 않습니다.

  따라서 각 NUMA 노드에 4개의 vCPU를 배포하고 vCPU 토폴로지를 호스트 토폴로지와 최

대한 가깝게 만들어야 합니다. 즉, 지정된 물리적 CPU의 형제 스레드로 실행되는 vCPU를 동일한 코어의 호스트 스레드에 고정해야 합니다. 자세한 내용은 아래 솔루션을 참조하십시오.

해결책

1. *호스트 토폴로지에 대한 정보를 가져옵니다.*

```
# virsh capabilities
```

출력에는 다음과 유사한 섹션이 포함되어야 합니다.

```
<topology>
  <cells num="2">
    <cell id="0">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="10" />
        <sibling id="1" value="21" />
      </distances>
      <cpus num="6">
        <cpu id="0" socket_id="0" core_id="0" siblings="0,3" />
        <cpu id="1" socket_id="0" core_id="1" siblings="1,4" />
        <cpu id="2" socket_id="0" core_id="2" siblings="2,5" />
        <cpu id="3" socket_id="0" core_id="0" siblings="0,3" />
        <cpu id="4" socket_id="0" core_id="1" siblings="1,4" />
        <cpu id="5" socket_id="0" core_id="2" siblings="2,5" />
      </cpus>
    </cell>
    <cell id="1">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="21" />
        <sibling id="1" value="10" />
      </distances>
      <cpus num="6">
        <cpu id="6" socket_id="1" core_id="3" siblings="6,9" />
        <cpu id="7" socket_id="1" core_id="4" siblings="7,10" />
        <cpu id="8" socket_id="1" core_id="5" siblings="8,11" />
        <cpu id="9" socket_id="1" core_id="3" siblings="6,9" />
        <cpu id="10" socket_id="1" core_id="4" siblings="7,10" />
        <cpu id="11" socket_id="1" core_id="5" siblings="8,11" />
      </cpus>
```

```
    </cell>
  </cells>
</topology>
```

2. *선택 사항: 해당 툴 및 유틸리티를 사용하여 VM의 성능을 테스트합니다.*

3. *호스트에서 1GiB 대규모 페이지를 설정하고 마운트합니다.*

   a. *호스트의 커널 명령줄에 다음 행을 추가합니다.*

   ```
   default_hugepagesz=1G hugepagesz=1G
   ```

   b. *다음 콘텐츠를 사용하여 /etc/systemd/system/hugetlb-gigantic-pages.service 파일을 만듭니다.*

   ```
   [Unit]
   Description=HugeTLB Gigantic Pages Reservation
   DefaultDependencies=no
   Before=dev-hugepages.mount
   ConditionPathExists=/sys/devices/system/node
   ConditionKernelCommandLine=hugepagesz=1G

   [Service]
   Type=oneshot
   RemainAfterExit=yes
   ExecStart=/etc/systemd/hugetlb-reserve-pages.sh

   [Install]
   WantedBy=sysinit.target
   ```

   c. *다음 콘텐츠를 사용하여 /etc/systemd/hugetlb-reserve-pages.sh 파일을 만듭니다.*

   ```
   #!/bin/sh

   nodes_path=/sys/devices/system/node/
   if [ ! -d $nodes_path ]; then
    echo "ERROR: $nodes_path does not exist"
    exit 1
   fi

   reserve_pages()
   {
    echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
   }
   ```

```
reserve_pages 4 node1
reserve_pages 4 node2
```

이렇게 하면 **node1** 에서 **4GiB** 대규모 페이지를 예약하고 **node2** 에서 **4GiB** 대규모 페이지를 예약합니다.

d. 이전 단계에서 생성한 스크립트를 실행 파일로 만듭니다.

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e. 부팅 시 대규모 페이지 예약을 활성화합니다.

```
# systemctl enable hugetlb-gigantic-pages
```

4. 이 예제에서는 **super-VM** 예제에서 최적화하려는 **VM**의 **XML** 구성을 편집하려면 **virsh edit** 명령을 사용합니다.

```
# virsh edit super-vm
```

5. **VM**의 **XML** 구성을 다음과 같이 조정합니다.

a. **8**개의 정적 **vCPU**를 사용하도록 **VM**을 설정합니다. **< vcpu/>** 요소를 사용하여 이 작업을 수행합니다.

b. 각 **vCPU** 스레드를 토폴로지에서 미러링하는 해당 호스트 **CPU** 스레드에 고정합니다. 이렇게 하려면 **< cpu tune>** 섹션의 **<vcpu pin/ >** 요소를 사용합니다.

위의 **virsh capabilities** 유틸리티에서와 같이 호스트 **CPU** 스레드는 해당 코어에서 순차적으로 정렬되지 않습니다. 또한 **vCPU** 스레드는 동일한 **NUMA** 노드에서 사용 가능한 최고 호스트 코어 세트에 고정되어야 합니다. 테이블 그림은 아래의 샘플 토폴로지 섹션을 참조하십시오.

1단계와 **b** 단계를 위한 **XML** 구성은 다음과 같습니다.

```
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='4'/>
```

```
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
 </cputune>
```

c. *1GiB 대규모 페이지를 사용하도록 VM을 설정합니다.*

```
<memoryBacking>
 <hugepages>
  <page size='1' unit='GiB'/>
 </hugepages>
</memoryBacking>
```

d. *호스트의 해당 NUMA 노드의 메모리를 사용하도록 VM의 NUMA 노드를 구성합니다. 이렇게 하려면 < numatune /> 섹션의 <memnode / > 요소를 사용합니다.*

```
<numatune>
 <memory mode="preferred" nodeset="1"/>
 <memnode cellid="0" mode="strict" nodeset="0"/>
 <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

e. *CPU 모드가 host-passthrough 로 설정되어 있고 CPU가 passthrough 모드에서 캐시를 사용하는지 확인합니다.*

```
<cpu mode="host-passthrough">
 <topology sockets="2" cores="2" threads="2"/>
 <cache mode="passthrough"/>
```

검증

1. *VM의 결과 XML 구성에 다음과 유사한 섹션이 포함되어 있는지 확인합니다.*

```
[...]
 <memoryBacking>
  <hugepages>
   <page size='1' unit='GiB'/>
  </hugepages>
 </memoryBacking>
 <vcpu placement='static'>8</vcpu>
 <cputune>
  <vcpupin vcpu='0' cpuset='1'/>
```

```
    <vcpupin vcpu='1' cpuset='4'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='5'/>
    <vcpupin vcpu='4' cpuset='7'/>
    <vcpupin vcpu='5' cpuset='10'/>
    <vcpupin vcpu='6' cpuset='8'/>
    <vcpupin vcpu='7' cpuset='11'/>
    <emulatorpin cpuset='6,9'/>
  </cputune>
  <numatune>
    <memory mode="preferred" nodeset="1"/>
    <memnode cellid="0" mode="strict" nodeset="0"/>
    <memnode cellid="1" mode="strict" nodeset="1"/>
  </numatune>
  <cpu mode="host-passthrough">
    <topology sockets="2" cores="2" threads="2"/>
    <cache mode="passthrough"/>
    <numa>
      <cell id="0" cpus="0-3" memory="2" unit="GiB">
        <distances>
          <sibling id="0" value="10"/>
          <sibling id="1" value="21"/>
        </distances>
      </cell>
      <cell id="1" cpus="4-7" memory="2" unit="GiB">
        <distances>
          <sibling id="0" value="21"/>
          <sibling id="1" value="10"/>
        </distances>
      </cell>
    </numa>
  </cpu>
</domain>
```

2. *선택 사항: 해당 툴 및 유틸리티를 사용하여 **VM**의 성능을 테스트하여 **VM** 최적화의 영향을 평가합니다.*

토폴로지 샘플

- *다음 표에서는 **vCPU**와 해당 **CPU**가 고정된 호스트 **CPU** 간 연결을 보여줍니다.*

*표 **13.1.** 호스트 토폴로지*

CPU 스레드	0	3	1	4	2	5	6	9	7	10	8	11
코어	0		1		2		3		4		5	
소켓	0						1					
NUMA 노드	0						1					

표 13.2. VM 토폴로지

vCPU 스레드	0	1	2	3	4	5	6	7
코어	0		1		2		3	
소켓	0				1			
NUMA 노드	0				1			

표 13.3. 결합된 호스트 및 VM 토폴로지

vCPU 스레드			0	1	2	3			4	5	6	7
호스트 CPU 스레드	0	3	1	4	2	5	6	9	7	10	8	11
코어	0		1		2		3		4		5	
소켓	0						1					
NUMA 노드	0						1					

이 시나리오에는 NUMA 노드 2개와 8개의 vCPU가 있습니다. 따라서 4 vCPU 스레드를 각 노드에 고정해야 합니다.

또한 Red Hat은 호스트 시스템 작업을 위해 각 노드에서 적어도 하나의 CPU 스레드를 사용할 수 있도록 하는 것이 좋습니다.

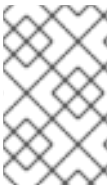
이 예제에서 각 NUMA 노드에는 각각 2개의 호스트 CPU 스레드가 있는 코어 3개가 있으므로 노드 0에 대한 세트가 다음과 같이 변환됩니다.

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

13.6.5. 커널 동일한 페이지 병합 관리

KSM(커널 동일 페이지 병합)은 VM(가상 머신) 간에 동일한 메모리 페이지를 공유하여 메모리 밀도를 향상시킵니다. 그러나 KSM을 활성화하면 CPU 사용률이 증가하며 워크로드에 따라 전체 성능에 부정적인 영향을 미칠 수 있습니다.

요구 사항에 따라 단일 세션에 대해 KSM을 활성화하거나 비활성화할 수 있습니다.



참고

RHEL 9 이상에서는 KSM이 기본적으로 비활성화되어 있습니다.

사전 요구 사항

- 호스트 시스템에 대한 루트 액세스.

절차

- KSM을 비활성화합니다.
  - 단일 세션에 대해 KSM을 비활성화하려면 systemctl 유틸리티를 사용하여 ksm 및 ksmtuned 서비스를 중지합니다.

```
# systemctl stop ksm

# systemctl stop ksmtuned
```

  - KSM을 영구적으로 비활성화하려면 systemctl 유틸리티를 사용하여 ksm 및 ksmtuned 서비스를 비활성화합니다.

```
# systemctl disable ksm
Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
# systemctl disable ksmtuned
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```

참고

KSM을 비활성화하기 전에 VM 간에 공유되는 메모리 페이지는 공유로 유지됩니다. 공유를 중지하려면 다음 명령을 사용하여 시스템의 모든 PageKSM 페이지를 삭제합니다.

```
# echo 2 > /sys/kernel/mm/ksm/run
```

익명 페이지가 KSM 페이지를 교체한 후 khugepaged 커널 서비스는 VM의 실제 메모리에서 투명한 hugepages를 다시 빌드합니다.

- KSM을 활성화합니다.

주의

KSM을 활성화하면 CPU 사용률이 증가하고 전체 CPU 성능에 영향을 미칩니다.

1. ksmtuned 서비스를 설치합니다.

```
# yum install ksmtuned
```

2. 서비스를 시작합니다.

- 단일 세션에 KSM을 활성화하려면 systemctl 유틸리티를 사용하여 ksm 및 ksmtuned 서비스를 시작합니다.

```
# systemctl start ksm
# systemctl start ksmtuned
```

- KSM을 영구적으로 활성화하려면 systemctl 유틸리티를 사용하여 ksm 및 ksmtuned 서비스를 활성화합니다.

```
# systemctl enable ksm
Created symlink /etc/systemd/system/multi-user.target.wants/ksm.service →
/usr/lib/systemd/system/ksm.service

# systemctl enable ksmtuned
Created symlink /etc/systemd/system/multi-user.target.wants/ksmtuned.service →
/usr/lib/systemd/system/ksmtuned.service
```

13.7. 가상 머신 네트워크 성능 최적화

VM의 NIC(네트워크 인터페이스 카드)의 가상 특성으로 인해 VM은 할당된 호스트 네트워크 대역폭의 일부를 손실하므로 VM의 전체 워크로드 효율성을 줄일 수 있습니다. 다음 팁은 가상 NIC(vNIC) 처리량에 가상화의 부정적인 영향을 최소화할 수 있습니다.

절차

다음 방법 중 하나를 사용하고 VM 네트워크 성능에 유용한 영향을 미치는지 확인합니다.

***vhost_net** 모듈 활성화*

*호스트에서 **vhost_net** 커널 기능이 활성화되었는지 확인합니다.*

```
# lsmod | grep vhost
vhost_net          32768  1
vhost              53248  1 vhost_net
tap                24576  1 vhost_net
tun                57344  6 vhost_net
```

*이 명령의 출력이 비어 있으면 **vhost_net** 커널 모듈을 활성화합니다.*

```
# modprobe vhost_net
```

*다중 큐 **virtio-net** 설정*

*VM의 다중 대기열 **virtio-net** 기능을 설정하려면 **virsh edit** 명령을 사용하여 VM의 **XML** 구성에 대해 편집합니다. **XML**에서 **< devices>** 섹션에 다음을 추가하고 **N** 을 VM의 **vCPU** 수로 최대 **16**개씩 바꿉니다.*

```
<interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
    <driver name='vhost' queues='N'/>
</interface>
```

VM이 실행 중인 경우 변경 사항이 적용되도록 다시 시작합니다.

네트워크 패킷 배치

*긴 전송 경로가 있는 **Linux VM** 구성에서 패킷을 커널에 제출하기 전에 배치하면 캐시 사용률이 향상될 수 있습니다. 패킷 배치를 설정하려면 호스트에서 다음 명령을 사용하고 **tap0** 을 VM에서 사용하는 네트워크 인터페이스 이름으로 바꿉니다.*

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

*호스트 **NIC**에서 **SR-IOV**를 지원하는 경우 **vNICs**에 **SR-IOV** 장치 할당을 사용합니다. 자세한 내용*

은 SR-IOV 장치 관리를 참조하십시오.

추가 리소스

- 가상 네트워킹 이해

13.8. 가상 머신 성능 모니터링 툴

가장 많은 VM 리소스를 소비하는 항목과 최적화가 필요한 VM 성능 측면을 식별하려면 일반 및 VM별 성능 진단 툴을 모두 사용할 수 있습니다.

기본 OS 성능 모니터링 툴

표준 성능 평가를 위해 호스트 및 게스트 운영 체제에서 기본적으로 제공되는 유틸리티를 사용할 수 있습니다.

- RHEL 9 호스트에서 root로 최상위 유틸리티 또는 시스템을 모니터링하고 출력에서 qemu 및 virt 를 찾습니다. VM에서 사용하는 호스트 시스템 리소스의 양을 보여줍니다.
  - 모니터링 도구가 qemu 또는 virt 프로세스가 호스트 CPU 또는 메모리 용량의 많은 부분을 소비하는 것을 표시하는 경우, perf 유틸리티를 사용하여 조사합니다. 자세한 내용은 아래를 참조하십시오.
  - 또한 vhost_net 스레드 프로세스(예: vhost_net -1234)가 과도하게 많은 호스트 CPU 용량을 소비하는 경우 다중 대기열 virtio-net 과 같은 가상 네트워크 최적화 기능을 사용하는 것이 좋습니다.
- 게스트 운영 체제에서는 시스템에서 사용 가능한 성능 유틸리티 및 애플리케이션을 사용하여 가장 많은 시스템 리소스를 사용하는 프로세스를 평가합니다.
  - Linux 시스템에서는 top 유틸리티를 사용할 수 있습니다.
  - Windows 시스템에서는 작업 관리자 애플리케이션을 사용할 수 있습니다.

perf kvm

perf 유틸리티를 사용하여 RHEL 9 호스트의 성능에 대한 가상화별 통계를 수집하고 분석할 수 있습니

다. 이렇게 하려면 다음을 수행합니다.

1. 호스트에서 **perf** 패키지를 설치합니다.

```
# dnf install perf
```

2. 가상화 호스트의 각 통계를 표시하려면 **perf kvm stat** 명령 중 하나를 사용합니다.

- 하이퍼바이저에 대한 실시간 모니터링을 위해서는 **perf kvm stat** 라이브 명령을 사용하십시오.

- 일정 기간 동안 하이퍼바이저의 **perf** 데이터를 기록하려면 **perf kvm stat record** 명령을 사용하여 로깅을 활성화합니다. 명령이 취소되거나 중단되면 데이터는 **perf.data.guest** 파일에 저장되며, **perf kvm** 통계 보고서 명령을 사용하여 분석할 수 있습니다.

3. **VM-EXIT** 이벤트 유형 및 해당 배포에 대한 **perf** 출력을 분석합니다. 예를 들어 **PAUSE_INSTRUCTION** 이벤트는 자주 발생하지 않아야 하지만 다음 출력에서는 호스트 **CPU**가 실행 중인 **vCPU**를 제대로 처리하지 않는다고 제안합니다. 이러한 시나리오에서는 활성 **VM** 중 일부를 종료하거나, 이러한 **VM**에서 **vCPU**를 제거하거나, **vCPU의 성능을 조정하는** 것이 좋습니다.

```
# perf kvm stat report

Analyze events for all VMs, all VCPUs:


             VM-EXIT    Samples  Samples%     Time%    Min Time    Max Time         Avg time

  EXTERNAL_INTERRUPT     365634    31.59%    18.04%      0.42us  58780.59us    204.08us ( +-   0.99% )
         MSR_WRITE     293428    25.35%     0.13%      0.59us  17873.02us      1.80us ( +-   4.63% )
    PREEMPTION_TIMER     276162    23.86%     0.23%      0.51us  21396.03us      3.38us ( +-   5.19% )
  PAUSE_INSTRUCTION     189375    16.36%    11.75%      0.72us  29655.25us    256.77us ( +-   0.70% )
                HLT      20440     1.77%    69.83%      0.62us  79319.41us  14134.56us ( +-   0.79% )
             VMCALL      12426     1.07%     0.03%      1.02us   5416.25us      8.77us ( +-   7.36% )
      EXCEPTION_NMI         27     0.00%     0.00%      0.69us      1.34us      0.98us ( +-   3.50% )
      EPT_MISCONFIG          5     0.00%     0.00%      5.15us     10.85us      7.88us ( +-
```

```
11.67% )

Total Samples:1157497, Total events handled time:413728274.66us.
```

perf kvm 통계 출력의 신호 문제를 신호할 수 있는 기타 이벤트 유형은 다음과 같습니다.

- INSN_EMULATION - 하위 VM I/O 구성을 제안합니다.

perf 를 사용하여 가상화 성능을 모니터링하는 방법에 대한 자세한 내용은 perf-kvm 매뉴얼 페이지를 참조하십시오.

numastat

시스템의 현재 NUMA 구성을 보려면 numactl 패키지를 설치하여 제공하는 numastat 유틸리티를 사용하면 됩니다.

다음은 각각 여러 NUMA 노드에서 메모리를 가져오는 4개의 VM이 있는 호스트를 보여줍니다. 이는 vCPU 성능에 적합하지 않으며 보증 조정:

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)     68     16    357   6936      2      3    147    598  8128
51747 (qemu-kvm)    245     11      5     18   5172   2532      1     92  8076
53736 (qemu-kvm)     62    432   1661    506   4851    136     22    445  8116
53773 (qemu-kvm)   1393      3      1      2     12      0      0   6702  8114
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total              1769    463   2024   7462  10037   2672    169   7837 32434
```

반면 다음은 단일 노드에서 각 VM에 제공되는 메모리를 훨씬 더 효율적으로 보여줍니다.

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51747 (qemu-kvm)      0      0      7      0   8072      0      1      0  8080
53736 (qemu-kvm)      0      0      7      0      0      0   8113      0  8120
53773 (qemu-kvm)      0      0      7      0      0      0      1   8110  8118
59065 (qemu-kvm)      0      0   8050      0      0      0      0      0  8051
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                 0      0   8072      0   8072      0   8114   8110 32368
```

13.9. 추가 리소스

- *Windows 가상 머신 최적화*

14장. POWERTOP를 사용하여 전력 소비 관리

시스템 관리자는 PowerTOP 도구를 사용하여 전력 소비를 분석하고 관리할 수 있습니다.

14.1. POWERTOP의 목적

PowerTOP 는 전력 소비와 관련된 문제를 진단하고 PV 수명을 연장하는 방법에 대한 제안을 제공하는 프로그램입니다.

PowerTOP 툴은 시스템의 총 전원 사용량과 각 프로세스, 장치, 커널 작업자, 타이머 및 인터럽트 처리기에 대한 개별 전원 사용량을 추정할 수 있습니다. 이 툴은 CPU를 자주 사용하는 커널 및 사용자 공간 애플리케이션의 특정 구성 요소를 식별할 수도 있습니다.

Red Hat Enterprise Linux 9는 PowerTOP 버전 2.x를 사용합니다.

14.2. POWERTOP 사용

사전 요구 사항

- PowerTOP 를 사용할 수 있으려면 powertop 패키지가 시스템에 설치되어 있는지 확인합니다.

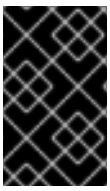
```
# dnf install powertop
```

14.2.1. PowerTOP 시작

절차

- PowerTOP 를 실행하려면 다음 명령을 사용하십시오.

```
# powertop
```



중요

powertop 명령을 실행할 때 PC 전원에서 노트북을 실행해야 합니다.

14.2.2. PowerTOP 교정

절차

1. *노트북에서 다음 명령을 실행하여 전력 추정 엔진을 교정할 수 있습니다.*

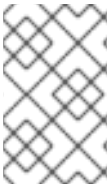
```
# powertop --calibrate
```

2. *프로세스 중에 시스템과 상호 작용하지 않고 교정을 완료하도록 합니다.*

   프로세스에서 다양한 테스트, 밝기 레벨 및 스위치 장치를 통해 사이클을 수행하고 켜거나 끄면 시간이 걸립니다.

3. ***Calibration** 프로세스가 완료되면 **PowerTOP** 가 정상적으로 시작됩니다. 데이터를 수집하기 위해 약 **1**시간 동안 실행되도록 합니다.*

   충분한 데이터가 수집되면 출력 테이블의 첫 번째 열에 전력 추정치가 표시됩니다.



참고

***powertop --calibrate** 는 노트북에서만 사용할 수 있습니다.*

14.2.3. 측정 간격 설정

*기본적으로 **PowerTOP** 는 **20**초 간격으로 측정을 수행합니다.*

이 측정 빈도를 변경하려면 다음 절차를 따르십시오.

절차

- *시간 옵션을 사용하여 **powertop** 명령을 실행합니다.*

```
# powertop --time=time in seconds
```

14.2.4. 추가 리소스

***PowerTOP** 사용 방법에 대한 자세한 내용은 **powertop man** 페이지를 참조하십시오.*

14.3. POWERTOP 통계

실행되는 동안 **PowerTOP** 는 시스템에서 통계를 수집합니다.

PowerTOP의 출력은 여러 탭을 제공합니다.

- 개요
- 유휴 상태 통계
- 빈도 통계
- 장치 통계
- 튜닝 가능 항목
- **WakeUp**

Tab 및 **Shift+Tab** 키를 사용하여 이러한 탭을 통해 반복할 수 있습니다.

14.3.1. 개요 탭

개요 탭에서 가장 자주 **CPU**로 **wakeups**를 보내거나 가장 많은 전원을 사용하는 구성 요소 목록을 볼 수 있습니다. 프로세스, 인터럽트, 장치 및 기타 리소스를 포함한 개요 탭의 항목은 사용률에 따라 정렬됩니다.

개요 탭 내의 인접 열은 다음과 같은 정보를 제공합니다.

사용법

리소스 사용 방법에 대한 전원 추정치입니다.

events/s

초당 Wakeup입니다. 초당 wakeups 수는 서비스 또는 장치 및 커널 드라이버가 얼마나 효율적으로 수행되는지를 나타냅니다. 압축은 더 적은 전력이 감소한다는 것을 의미합니다. 구성 요소는 전력 사용량을 얼마나 더 최적화할 수 있는지에 따라 정렬됩니다.

카테고리

프로세스, 장치 또는 타이머와 같은 구성 요소를 분류합니다.

설명

구성 요소에 대한 설명입니다.

적절하게 교정을하면 첫 번째 열의 나열된 모든 항목에 대한 전력 소비 추정치도 표시됩니다.

이 외에도 개요 탭에는 다음과 같은 요약 통계가 포함된 행이 포함되어 있습니다.

- 총 전력 소비
- 남은 수명은 (해당되는 경우에만)
- 초당 총 절전 모드, 초당 GPU 작업 및 초당 가상 파일 시스템 작업 요약

14.3.2. Idle 통계 탭

Idle stats 탭에서는 모든 프로세서와 코어에 대한 C-states의 사용법을 보여주지만, Frequency stats 탭에는 모든 프로세서 및 코어에 대해 ScanSetting 모드를 포함한 P-states의 사용법을 보여줍니다. C- 또는 P-상태의 기간은 CPU 사용량이 얼마나 최적화되었는지를 나타냅니다. CPU가 더 높은 C- 또는 P-상태(예: C4가 C3)보다 더 긴 경우 CPU 사용량 최적화가 더 좋습니다. 이상적으로는 시스템이 유휴 상태일 때 가장 높은 C- 또는 P-상태에서 90% 이상이 됩니다.

14.3.3. 장치 통계 탭

장치 통계 탭은 개요 탭과 유사한 정보를 제공하지만 장치에 대해서만 정보를 제공합니다.

14.3.4. Tunables 탭

Tunables 탭에는 전력 소비를줄이기 위해 시스템을 최적화하기 위한 제안사항이 포함되어 있습니다.

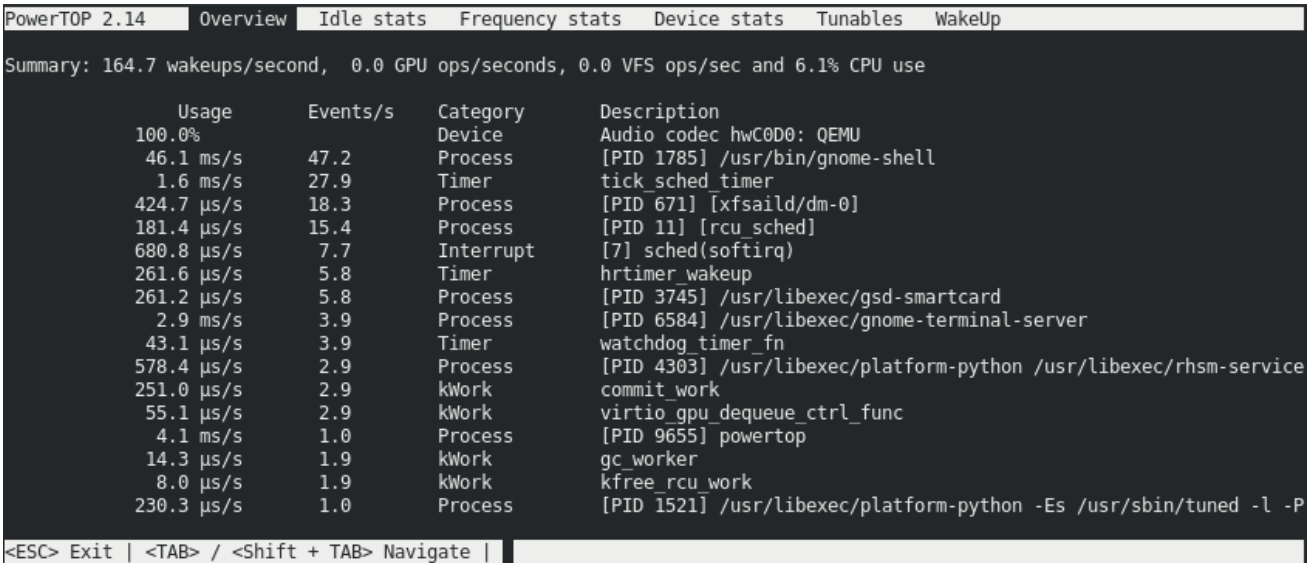
위쪽 및 아래 키를 사용하여 제안을 통해 이동하고, **Enter** 키를 사용하여 제안을 전환하거나 해제할 수 있습니다.

14.3.5. WakeUp 탭

WakeUp 탭에는 필요에 따라 사용자가 변경할 수 있는 장치 활성화 설정이 표시됩니다.

up 및 **down** 키를 사용하여 사용 가능한 설정으로 이동하고 **enter** 키를 사용하여 설정을 활성화하거나 비활성화합니다.

그림 14.1. PowerTOP 출력



추가 리소스

PowerTOP 에 대한 자세한 내용은 *PowerTOP의 홈 페이지를* 참조하십시오.

14.4. POWERTOP에서 일부 인스턴스에서 FREQUENCY STATS 값을 표시하지 않는 이유

Intel P-State 드라이버를 사용하는 동안 **PowerTOP**는 드라이버가 **Passive** 모드인 경우 **Frequency statistics** 탭에 값만 표시합니다. 그러나 이 경우에도 값이 불완전할 수 있습니다.

전체적으로 **Intel P-State** 드라이버의 세 가지 가능한 모드가 있습니다.

- *HWP(HWP) 하드웨어 P-States를 사용하는 활성 모드*
- *HWP가 없는 활성 모드*
- *Passive 모드*

ACPI CPUfreq 드라이버로 전환하면 PowerTOP에서 완전한 정보가 표시됩니다. 그러나 시스템을 기본 설정으로 유지하는 것이 좋습니다.

로드된 드라이버를 확인하고 어떤 모드에서 다음을 실행합니다.

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_driver
```

- *Intel P-State 드라이버가 로드되고 활성 모드에서 intel_pstate 가 반환됩니다.*
- *Intel P-State 드라이버가 로드되고 passive 모드인 경우 intel_cpufreq 가 반환됩니다.*
- *ACPI CPU freq 드라이버가 로드되면 ACPI-cpu freq가 반환됩니다.*

Intel P-State 드라이버를 사용하는 동안 커널 부팅 명령줄에 다음 인수를 추가하여 드라이버를 패시브 모드로 강제 실행합니다.

```
intel_pstate=passive
```

Intel P-State 드라이버를 비활성화하고 를 사용하려면 ACPI CPUfreq 드라이버를 통해 커널 부팅 명령줄에 다음 인수를 추가합니다.

```
intel_pstate=disable
```

14.5. HTML 출력 생성

터미널의 powertop 출력 외에도 HTML 보고서를 생성할 수도 있습니다.

절차

- --html 옵션을 사용하여 powertop 명령을 실행합니다.

```
# powertop --html=htmlfile.html
```

  htmlfile.html 매개 변수를 출력 파일에 필요한 이름으로 바꿉니다.

14.6. 전력 소비 최적화

전력 소비를 최적화하기 위해 powertop 서비스 또는 powertop 2tuned 유틸리티를 사용할 수 있습니다.

14.6.1. powertop 서비스를 사용하여 전원 소비 최적화

powertop 서비스를 사용하여 부팅 시 Tunables 탭에서 모든 PowerTOP의 제안을 자동으로 활성화할 수 있습니다.

절차

- powertop 서비스를 활성화합니다.

```
# systemctl enable powertop
```

14.6.2. powertop2tuned 유틸리티

powertop2tuned 유틸리티를 사용하면 PowerTOP 제안에서 사용자 정의 TuneD 프로필을 생성할 수 있습니다.

기본적으로 powertop2tuned 는 /etc/tuned/ 디렉터리에 프로필을 생성하고 현재 선택한 TuneD 프로필의 사용자 정의 프로필을 기반으로 합니다. 보안상의 이유로 새 프로필에서는 모든 PowerTOP 튜닝이 처음에 비활성화되어 있습니다.

튜닝을 활성화하려면 다음을 수행할 수 있습니다.

- /etc/tuned/profile_name/tuned.conf 파일에서 해당 파일의 주석을 제거합니다.

- *enable 또는 -e 옵션을 사용하여 PowerTOP 에서 제공하는 대부분의 튜닝을 활성화하는 새 프로필을 생성합니다.*

  USB 자동 중지와 같은 잠재적으로 문제가 있는 특정 튜닝은 기본적으로 비활성화되어 있으며 수동으로 주석 처리를 해제해야 합니다.

14.6.3. powertop2tuned 유틸리티를 사용하여 전원 소비 최적화

사전 요구 사항

- *powertop2tuned 유틸리티는 시스템에 설치되어 있습니다.*

```
# dnf install tuned-utils
```

절차

1. *사용자 정의 프로필을 생성합니다.*

```
# powertop2tuned new_profile_name
```

2. *새 프로필을 활성화합니다.*

```
# tuned-adm profile new_profile_name
```

추가 정보

- *powertop2tuned 에서 지원하는 전체 옵션 목록은 다음을 사용합니다.*

```
$ powertop2tuned --help
```

14.6.4. powertop.service 및 powertop2tuned 비교

다음과 같은 이유로 powertop2tuned 로 전력 소비를 최적화하는 것이 powertop.service 보다 우선합니다.

- *powertop2tuned 유틸리티는 PowerTOP 를 TuneD 에 통합하므로 두 툴의 이점을 활용할 수 있습니다.*

- *powertop2tuned 유틸리티를 사용하면 활성화된 튜닝을 세부적으로 제어할 수 있습니다.*
- *powertop2tuned 를 사용하면 잠재적으로 위험한 튜닝이 자동으로 활성화되지 않습니다.*
- *powertop2tuned 를 사용하면 재부팅하지 않고 롤백이 가능합니다.*

15장. PERF 시작하기

시스템 관리자는 perf 툴을 사용하여 시스템의 성능 데이터를 수집하고 분석할 수 있습니다.

15.1. PERF 소개

커널 기반 하위 시스템 성능 카운터(PCL)와 perf 사용자 공간 툴을 사용합니다. perf 는 PMU(Performance Monitoring Unit)를 사용하여 다양한 하드웨어 및 소프트웨어 이벤트를 측정, 기록 및 모니터링하는 강력한 도구입니다. perf 는 또한 tracepoints, kprobes, uprobes를 지원합니다.

15.2. PERF 설치

이 절차에서는 perf 사용자 공간 툴을 설치합니다.

절차

- perf 툴을 설치합니다.

```
# dnf install perf
```

15.3. 일반적인 PERF 명령

이 섹션에서는 일반적으로 사용되는 perf 명령의 개요를 제공합니다.

일반적으로 사용되는 perf 명령

perf 통계

이 명령은 실행된 명령 실행 및 클럭 사이클을 포함하여 일반적인 성능 이벤트에 대한 전반적인 통계를 제공합니다. 옵션을 사용하면 기본 측정 이벤트 이외의 이벤트를 선택할 수 있습니다.

perf record

이 명령은 성능 데이터를 파일 perf.data 에 기록합니다. 이 파일은 나중에 perf report 명령을 사용하여 분석할 수 있습니다.

perf 보고서

이 명령은 perf 레코드에서 생성한 perf.data 파일의 성능 데이터를 읽고 표시합니다.

perf 목록

이 명령은 특정 시스템에서 사용 가능한 이벤트를 나열합니다. 이러한 이벤트는 시스템의 성능 모니터링 하드웨어 및 소프트웨어 구성에 따라 달라집니다.

perf top

이 명령은 **top** 유틸리티와 유사한 기능을 수행합니다. 실시간으로 성능 카운터 프로필을 생성하고 표시합니다.

perf 추적

이 명령은 **strace** 툴과 유사한 기능을 수행합니다. 지정된 스레드 또는 프로세스에서 사용하는 시스템 호출과 해당 애플리케이션에서 수신한 모든 신호를 모니터링합니다.

perf 도움말

이 명령을 수행하면 전체 **perf** 명령 목록이 표시됩니다.

추가 리소스

- 하위 명령에 **--help** 옵션을 추가하여 도움말 페이지를 엽니다.

16장. PERF를 사용하여 CPU 사용량을 실시간으로 프로파일링

perf top 명령을 사용하여 다른 함수의 **CPU** 사용량을 실시간으로 측정할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

16.1. PERF의 목적

perf top 명령은 **top** 유틸리티와 유사한 실시간 시스템 프로파일링 및 기능에 사용됩니다. 그러나 **top** 유틸리티에서는 일반적으로 지정된 프로세스 또는 스레드가 사용 중인 **CPU** 시간을 표시합니다. 여기서 **perf** 는 각 특정 함수가 사용하는 **CPU** 시간을 보여줍니다. 기본 상태에서 **perf** 는 사용자 공간 및 커널 공간 모두에서 모든 **CPU**에서 사용되는 기능에 대해 알려줍니다. **perf** 를 사용하려면 루트 액세스가 필요합니다.

16.2. PERF를 사용하여 CPU 사용량 프로파일링

이 절차에서는 **perf top** 및 프로필 **CPU** 사용량을 실시간으로 활성화합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- **root** 액세스 권한이 있어야 합니다.

절차

- **perf**의 주요 모니터링 인터페이스를 시작합니다.

```
# perf top
```

모니터링 인터페이스는 다음과 유사합니다.

```
Samples: 8K of event 'cycles', 2000 Hz, Event count (approx.): 4579432780 lost: 0/0 drop:
0/0
Overhead  Shared Object       Symbol
```

```
  2.20%  [kernel]          [k] do_syscall_64
  2.17%  [kernel]          [k] module_get_kallsym
  1.49%  [kernel]          [k] copy_user_enhanced_fast_string
  1.37%  libpthread-2.29.so  [.] pthread_mutex_lock 1.31% [unknown] [.] 0000000000000000
1.07% [kernel] [k] psi_task_change 1.04% [kernel] [k] switch_mm_irqs_off 0.94% [kernel] [k]
fget
  0.74%  [kernel]          [k] entry_SYSCALL_64
  0.69%  [kernel]          [k] syscall_return_via_sysret
  0.69%  libxul.so         [.] 0x000000000113f9b0
  0.67%  [kernel]          [k] kallsyms_expand_symbol.constprop.0
  0.65%  firefox           [.] moz_xmalloc
  0.65%  libpthread-2.29.so  [.] __pthread_mutex_unlock_usercnt
  0.60%  firefox           [.] free
  0.60%  libxul.so         [.] 0x000000000241d1cd
  0.60%  [kernel]          [k] do_sys_poll
  0.58%  [kernel]          [k] menu_select
  0.56%  [kernel]          [k] _raw_spin_lock_irqsave
  0.55%  perf              [.] 0x00000000002ae0f3
```

*이 예에서 커널 함수 **do_syscall_64** 는 대부분의 **CPU** 시간을 사용합니다.*

추가 리소스

- ***perf-top(1)** 도움말 페이지*

16.3. 상위 출력의 해석

***perf** 상단 모니터링 인터페이스는 여러 열에 데이터를 표시합니다.*

"Overhead" 열

*지정된 함수에서 사용하는 **CPU**의 백분율을 표시합니다.*

"공유 오브젝트" 열

함수를 사용 중인 프로그램 또는 라이브러리 이름을 표시합니다.

<Symbol> 열

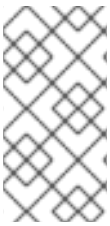
*함수 이름 또는 기호를 표시합니다. 커널 공간에 실행되는 함수는 **[k]** 에 의해 식별되며 사용자 공간으로 실행되는 함수는 **[.]** 에 의해 식별됩니다.*

16.4. PERF가 일부 함수 이름을 원시 함수 주소로 표시하는 이유

*커널 함수의 경우, **perf** 는 **/proc/kallsyms** 파일의 정보를 사용하여 샘플을 각 함수 이름 또는 기호에 매*

핑합니다. 그러나 사용자 공간으로 실행되는 함수의 경우 바이너리가 제거되었기 때문에 원시 함수 주소가 표시될 수 있습니다.

*실행 파일의 **debuginfo** 패키지를 설치하거나 실행 파일이 로컬에서 개발된 애플리케이션인 경우 이러한 상황에서 함수 이름 또는 기호를 표시하려면 애플리케이션(G의 **-g** 옵션)이 켜진 디버깅 정보로 컴파일해야 합니다.*



참고

*실행 파일과 연결된 **debuginfo** 를 설치한 후 **perf record** 명령을 다시 실행할 필요가 없습니다. **perf report** 명령을 다시 실행하면 됩니다.*

추가 리소스

- *디버깅 정보를 사용하여 디버깅 활성화*

16.5. 디버그 및 소스 리포지토리 활성화

***Red Hat Enterprise Linux**의 표준 설치는 디버그 및 소스 리포지토리를 활성화하지 않습니다. 이러한 리포지토리에는 시스템 구성 요소를 디버깅하고 성능을 측정하는 데 필요한 정보가 포함되어 있습니다.*

절차

- *소스 및 디버그 정보 패키지 채널 활성화: **$(uname -i)** 부분은 시스템의 아키텍처에 대해 일치하는 값으로 자동 교체됩니다.*

아키텍처 이름	값
64비트 Intel 및 AMD	x86_64
64비트 ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

16.6. GDB를 사용하여 애플리케이션 또는 라이브러리를 위한 DEBUGINFO 패키지 가져오기

*코드를 디버깅하려면 디버깅 정보가 필요합니다. 패키지에서 설치된 코드의 경우 **GNU Debugger(GDB)**는 누락된 디버그 정보를 자동으로 인식하고 패키지 이름을 확인하고 패키지를 가져오는*

방법에 대한 구체적인 조언을 제공합니다.

사전 요구 사항

- *디버그하려는 애플리케이션 또는 라이브러리가 시스템에 설치되어 있어야 합니다.*
- ***GDB** 및 **debuginfo-install** 툴이 시스템에 설치되어 있어야 합니다. 자세한 내용은 애플리케이션 디버그 설정을 참조하십시오.*
- ***debuginfo** 및 **debugsource** 패키지를 제공하는 채널은 시스템에서 구성하고 활성화해야 합니다.*

절차

1. *디버깅하려는 애플리케이션 또는 라이브러리에 **GDB** 연결을 시작합니다. **GDB**는 누락된 디버깅 정보를 자동으로 인식하며 실행할 명령을 제안합니다.*

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no
debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. ***exit GDB: q** 를 입력하고 **Enter** 를 사용하여 확인합니다.*

```
(gdb) q
```

3. ***GDB**에서 제안한 명령을 실행하여 필요한 **debuginfo** 패키지를 설치합니다.*

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

   ***dnf** 패키지 관리 도구는 변경 사항에 대한 요약을 제공하고 확인을 요청하며 확인 후 필요한 모든 파일을 다운로드하여 설치합니다.*

4. ***case GDB**는 **debuginfo** 패키지를 제안할 수 없으며 애플리케이션 또는 라이브러리에 대한 **debuginfo** 패키지 가져오기에 설명된 절차를 수동으로 수행합니다.*

추가 리소스

- ***Red Hat Developer Toolset** 사용자 가이드, 디버깅 정보 설치*
- ***RHEL** 시스템용 **debuginfo** 패키지를 다운로드하거나 설치하려면 어떻게 해야 합니까?* - ***Red Hat Knowledgebase** 솔루션*

17장. PERF STAT를 사용하여 프로세스 실행 중 이벤트 계산

perf stat 명령을 사용하여 프로세스 실행 중에 하드웨어 및 소프트웨어 이벤트를 계산할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

17.1. PERF 통계의 목적

perf stat 명령은 지정된 명령을 실행하고, 명령을 실행하는 동안 실행 중인 하드웨어 및 소프트웨어 이벤트 수를 유지하고, 이러한 개수의 통계를 생성합니다. 이벤트를 지정하지 않으면 **perf stat** 는 공통 하드웨어 및 소프트웨어 이벤트 세트를 계산합니다.

17.2. 통계로 이벤트 계산

perf stat 를 사용하여 명령 실행 중에 하드웨어 및 소프트웨어 이벤트 발생을 계산하고 이러한 개수의 통계를 생성할 수 있습니다. 기본적으로 **perf stat** 는 스레드별 모드에서 작동합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- 이벤트를 계산합니다.
  - 루트 액세스 권한 없이 **perf stat** 명령을 실행하면 사용자 영역에서만 발생하는 이벤트 수입니다.

```
$ perf stat ls
```

예 17.1. 루트 액세스없이 실행되었던 **perf stat**의 출력

```
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

 Performance counter stats for 'ls':

              1.28 msec task-clock:u              #    0.165 CPUs utilized
```

```
         0      context-switches:u        #    0.000 M/sec
         0      cpu-migrations:u          #    0.000 K/sec
       104      page-faults:u             #    0.081 M/sec
 1,054,302      cycles:u                  #    0.823 GHz
 1,136,989      instructions:u            #    1.08  insn per cycle
   228,531      branches:u                #  178.447 M/sec
    11,331      branch-misses:u           #    4.96% of all branches

0.007754312 seconds time elapsed

0.000000000 seconds user
0.007717000 seconds sys
```

*이전 예에서 볼 수 있듯이, **perf stat** 가 **root** 액세스없이 실행되는 경우 이벤트 이름 뒤에 **:u** 가 뒤따르면 이러한 이벤트가 사용자 공간에만 계산되었음을 나타냅니다.*

- *사용자 공간 및 커널 공간 이벤트를 모두 계산하려면 **stat** 를 실행할 때 **root** 액세스 권한이 있어야 합니다.*

```
# perf stat ls
```

*예 **17.2. perf stat**의 출력은 루트 액세스 권한으로 실행됨*

```
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

 Performance counter stats for 'ls':

         3.09 msec task-clock              #    0.119 CPUs utilized
           18      context-switches        #    0.006 M/sec
            3      cpu-migrations          #    0.969 K/sec
          108      page-faults             #    0.035 M/sec
    6,576,004      cycles                  #    2.125 GHz
    5,694,223      instructions            #    0.87  insn per cycle
    1,092,372      branches                #  352.960 M/sec
       31,515      branch-misses           #    2.89% of all branches

0.026020043 seconds time elapsed

0.000000000 seconds user
0.014061000 seconds sys
```

- *기본적으로 **perf stat** 는 스레드별 모드에서 작동합니다. **CPU** 전체 이벤트 수로 변경하려면 **-a** 옵션을 **perf stat** 에 전달합니다. **CPU** 전체 이벤트를 계산하려면 **root** 액세스 권한이 필요합니다.*

```
# perf stat -a ls
```

추가 리소스

- **perf-stat(1)** 도움말 페이지

17.3. 통계 출력의 해석

perf stat 는 지정된 명령을 실행하고 명령을 실행하는 동안 이벤트 발생을 계산하고 세 개의 열에서 이러한 수의 통계를 표시합니다.

1. 지정된 이벤트에 계산되는 발생 횟수

2. 계산된 이벤트의 이름입니다.

3. 관련 메트릭을 사용할 수 있는 경우 오른쪽 가장 큰 열에서 해시 기호(#) 뒤에 비율 또는 백분율이 표시됩니다.

   예를 들어 기본 모드에서 실행할 때 통계 는 사이클과 명령 모두를 계산하므로 가장 오른쪽 열에 사이클당 명령을 계산하고 표시합니다. 두 이벤트가 모두 기본적으로 계산되므로 분기와 유사한 동작을 모든 분기의 백분율로 볼 수 있습니다.

17.4. 실행 중인 프로세스에 PERF STAT 연결

실행 중인 프로세스에 **perf stat** 를 연결할 수 있습니다. 이 명령은 명령을 실행하는 동안 지정된 프로세스에서만 이벤트 발생을 계산하도록 **perf stat** 에 지시합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- 실행 중인 프로세스에 **perf stat** 를 연결합니다.

```
$ perf stat -p ID1,ID2 sleep seconds
```

  이전 예제에서는 **sleep** 명령을 사용하여 지정된 초 동안 **ID1** 및 **ID2** 의 **ID**가 있는 프로세스의

이벤트를 계산합니다.

추가 리소스

- ***perf-stat(1)** 도움말 페이지*

18장. PERF를 사용하여 성능 프로파일 기록 및 분석

perf 툴을 사용하면 성능 데이터를 기록하고 나중에 분석할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

18.1. PERF 레코드의 목적

perf record 명령은 성능 데이터를 샘플링하여 다른 perf 명령으로 읽고 시각화할 수 있는 파일 perf.data 에 저장합니다. perf.data 는 현재 디렉터리에서 생성되며 나중에 다른 시스템에서 액세스할 수 있습니다.

기록하기 위해 perf 레코드에 대한 명령을 지정하지 않으면 Ctrl+C 를 눌러 프로세스를 수동으로 중지할 때까지 기록합니다. -p 옵션 다음에 하나 이상의 프로세스 ID를 전달하여 특정 프로세스에 perf 레코드를 연결할 수 있습니다. 루트 액세스 권한 없이 perf 레코드 를 실행할 수 있지만 이렇게 하면 사용자 공간의 성능 데이터만 샘플링됩니다. 기본 모드에서 perf 레코드 는 CPU 사이클을 샘플링 이벤트로 사용하며 상속 모드가 활성화된 스레드별 모드에서 작동합니다.

18.2. 루트 액세스없이 성능 프로파일 기록

사용자 공간에만 샘플 및 레코드 성능 데이터에 대한 루트 액세스 권한 없이 perf 레코드를 사용할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

- 성능 데이터를 샘플을 기록합니다.

```
$ perf record command
```

명령을 샘플 데이터 중 데이터로 바꿉니다. 명령을 지정하지 않으면 Perf 레코드 는 Ctrl+C.

추가 리소스

- **perf-record(1) man page**

18.3. 루트 액세스 권한으로 성능 프로파일 기록

perf 레코드 를 루트 액세스와 함께 사용자 공간 및 커널 공간 모두에서 샘플 및 레코드 성능 데이터를 동시에 기록할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- 루트 액세스 권한이 있습니다.

절차

- 성능 데이터를 샘플을 기록합니다.

```
# perf record command
```

명령을 샘플 데이터 중 데이터로 바꿉니다. 명령을 지정하지 않으면 **Perf** 레코드 는 **Ctrl+C**.

추가 리소스

- **perf-record(1) man page**

18.4. CPU 모드에서 성능 프로파일 기록

CPU 모드의 **perf** 레코드 를 사용하여 모니터링된 **CPU**의 모든 스레드에서 및 사용자 공간 및 커널 공간 모두에서 성능 데이터를 샘플링하고 기록할 수 있습니다. 기본적으로 **CPU**당 모드에서는 모든 온라인 **CPU**를 모니터링합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- 성능 데이터를 샘플을 기록합니다.

```
# perf record -a command
```

명령을 샘플 데이터 중 데이터로 바꿉니다. 명령을 지정하지 않으면 **Perf** 레코드 는 **Ctrl+C.**

추가 리소스

- **perf-record(1) man page**

18.5. PERF 레코드를 사용하여 호출 그래프 데이터 캡처

성능 프로필의 다른 함수를 호출하는 함수가 기록되도록 **perf** 레코드 도구를 구성할 수 있습니다. 이렇게 하면 여러 프로세스가 동일한 함수를 호출하는 경우 성능 장애를 확인하는 데 도움이 됩니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- **--call-graph** 옵션을 사용하여 샘플 및 성능 데이터를 기록합니다.

```
$ perf record --call-graph method command
```

  - 명령을 샘플 데이터 중 데이터로 바꿉니다. 명령을 지정하지 않으면 **Perf** 레코드 는 **Ctrl+C.**

  - **method** 를 다음의 **unwinding** 방법 중 하나로 바꿉니다.

fp

프레임 포인터 메서드를 사용합니다. **GCC** 옵션으로 빌드된 바이너리와 같은 컴파일러 최적화에 따라 **--fomit-frame-pointer.**

dwarf

DWARF 호출 프레임 정보를 사용하여 스택의 배율을 해제합니다.

lbr

Intel 프로세서의 마지막 분기 레코드 하드웨어를 사용합니다.

추가 리소스

- **perf-record(1) man page**

18.6. PERF 보고서를 사용하여 PERF.DATA 분석

perf 보고서를 사용하여 **perf.data** 파일을 표시하고 분석할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- 현재 디렉터리에는 **perf.data** 파일이 있습니다.
- **perf.data** 파일이 루트 액세스 권한으로 생성된 경우 루트 액세스 권한을 사용하여 **perf** 보고서를 실행해야 합니다.

절차

- 추가 분석을 위해 **perf.data** 파일의 내용을 표시합니다.

```
# perf report
```

이 명령은 다음과 유사한 출력을 표시합니다.

```
Samples: 2K of event 'cycles', Event count (approx.): 235462960
Overhead  Command          Shared Object                     Symbol
  2.36%  kswapd0          [kernel.kallsyms]                 [k] page_vma_mapped_walk
  2.13%  sssd_kcm         libc-2.28.so                      [.] memset_avx2_erms 2.13% perf
[kernel.kallsyms] [k] smp_call_function_single 1.53% gnome-shell libc-2.28.so [.]
strcmp_avx2
  1.17%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_hash_table_lookup
```

```
  0.93%  Xorg            libc-2.28.so                  [.] memmove_avx_unaligned_erms 0.89%
gnome-shell libgobject-2.0.so.0.5600.4 [.] g_object_unref 0.87% kswapd0 [kernel.kallsyms]
[k] page_referenced_one 0.86% gnome-shell libc-2.28.so [.] memmove_avx_unaligned_erms
  0.83%  Xorg            [kernel.kallsyms]             [k] alloc_vmap_area
  0.63%  gnome-shell     libglib-2.0.so.0.5600.4       [.] g_slice_alloc
  0.53%  gnome-shell     libgirepository-1.0.so.1.0.0  [.] g_base_info_unref
  0.53%  gnome-shell     ld-2.28.so                    [.] _dl_find_dso_for_object
  0.49%  kswapd0         [kernel.kallsyms]             [k] vma_interval_tree_iter_next
  0.48%  gnome-shell     libpthread-2.28.so            [.] pthread_getspecific 0.47% gnome-
shell libgirepository-1.0.so.1.0.0 [.] 0x0000000000013b1d 0.45% gnome-shell libglib-
2.0.so.0.5600.4 [.] g_slice_free1 0.45% gnome-shell libgobject-2.0.so.0.5600.4 [.]
g_type_check_instance_is_fundamentally_a 0.44% gnome-shell libc-2.28.so [.] malloc 0.41%
swapper [kernel.kallsyms] [k] apic_timer_interrupt 0.40% gnome-shell ld-2.28.so [.]
_dl_lookup_symbol_x 0.39% kswapd0 [kernel.kallsyms] [k]
raw_callee_save___pv_queued_spin_unlock
```

추가 리소스

- **perf-report(1)** 도움말 페이지

18.7. PERF 보고서 출력 해석

perf report 명령을 실행하여 표시되는 표는 데이터를 여러 열로 정렬합니다.

'Overhead' 열

이 특정 함수에서 수집된 전체 샘플의 백분율을 나타냅니다.

<명령> 열

샘플이 수집되는 프로세스에 대해 알려줍니다.

'공유 오브젝트' 열

샘플이 들어오는 **ELF** 이미지의 이름을 표시합니다(샘플에서 가져온 샘플 **[kernel.kallsyms] is used).**

<Symbol> 열

함수 이름 또는 기호를 표시합니다.

기본 모드에서 함수는 가장 높은 오버헤드가 먼저 표시된 순서를 사용하여 내림차순으로 정렬됩니다.

18.8. 다른 장치에서 읽을 수 있는 PERF.DATA 파일 생성

perf 툴을 사용하여 다른 장치에서 분석할 수 있도록 **perf.data** 파일에 성능 데이터를 기록할 수 있습니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- **kernel debuginfo** 패키지가 설치되어 있습니다. 자세한 내용은 **GDB**를 사용하여 애플리케이션 또는 라이브러리에 대한 **debuginfo** 패키지 가져오기를 참조하십시오.

절차

1. 더 조사하고자 하는 성능 데이터를 캡처합니다.

```
# perf record -a --call-graph fp sleep seconds
```

이 예제에서는 **sleep** 명령을 사용하여 지시한 시간(초) 동안 전체 시스템에 대해 **perf.data** 를 생성합니다. 또한 프레임 포인터 방법을 사용하여 호출 그래프 데이터를 캡처합니다.

2. 기록된 데이터의 디버그 기호가 포함된 아카이브 파일을 생성합니다.

```
# perf archive
```

검증 단계

- 현재 활성 디렉터리에 아카이브 파일이 생성되었는지 확인합니다.

```
# ls perf.data*
```

출력에는 현재 디렉토리의 모든 파일이 **perf.data** 로 표시됩니다. 아카이브 파일의 이름은 다음과 같습니다.

```
perf.data.tar.gz
```

또는

```
perf.data.tar.bz2
```

추가 리소스

- perf를 사용하여 성능 프로파일 기록 및 분석
- perf 레코드를 사용하여 호출 그래프 데이터 캡처

18.9. 다른 장치에서 생성된 PERF.DATA 파일 분석

perf 툴을 사용하여 다른 장치에서 생성된 perf.data 파일을 분석할 수 있습니다.

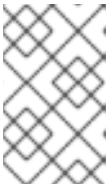
사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.
- 다른 장치에 생성된 perf.data 파일 및 연결된 아카이브 파일이 현재 장치에 있습니다.

절차

1. perf.data 파일과 아카이브 파일을 현재 활성 디렉터리에 복사합니다.
2. 아카이브 파일을 ~/.debug 로 추출합니다.

```
# mkdir -p ~/.debug
# tar xf perf.data.tar.bz2 -C ~/.debug
```



참고

아카이브 파일의 이름은 perf.data.tar.gz 로 지정됩니다.

3. 추가 분석을 위해 perf.data 파일을 엽니다.

```
# perf report
```

18.10. PERF가 일부 함수 이름을 원시 함수 주소로 표시하는 이유

커널 함수의 경우, **perf** 는 **/proc/kallsyms** 파일의 정보를 사용하여 샘플을 각 함수 이름 또는 기호에 매핑합니다. 그러나 사용자 공간으로 실행되는 함수의 경우 바이너리가 제거되었기 때문에 원시 함수 주소가 표시될 수 있습니다.

실행 파일의 **debuginfo** 패키지를 설치하거나 실행 파일이 로컬에서 개발된 애플리케이션인 경우 이러한 상황에서 함수 이름 또는 기호를 표시하려면 애플리케이션(**G**의 **-g** 옵션)이 켜진 디버깅 정보로 컴파일해야 합니다.

참고

실행 파일과 연결된 **debuginfo** 를 설치한 후 **perf record** 명령을 다시 실행할 필요가 없습니다. **perf report** 명령을 다시 실행하면 됩니다.

추가 리소스

- 디버깅 정보를 사용하여 디버깅 활성화

18.11. 디버그 및 소스 리포지토리 활성화

Red Hat Enterprise Linux의 표준 설치는 디버그 및 소스 리포지토리를 활성화하지 않습니다. 이러한 리포지토리에는 시스템 구성 요소를 디버깅하고 성능을 측정하는 데 필요한 정보가 포함되어 있습니다.

절차

- 소스 및 디버그 정보 패키지 채널 활성화: **$(uname -i)** 부분은 시스템의 아키텍처에 대해 일치하는 값으로 자동 교체됩니다.

아키텍처 이름	값
64비트 Intel 및 AMD	x86_64
64비트 ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

18.12. GDB를 사용하여 애플리케이션 또는 라이브러리를 위한 DEBUGINFO 패키지 가져오기

코드를 디버깅하려면 디버깅 정보가 필요합니다. 패키지에서 설치된 코드의 경우 **GNU Debugger(GDB)**는 누락된 디버그 정보를 자동으로 인식하고 패키지 이름을 확인하고 패키지를 가져오는 방법에 대한 구체적인 조언을 제공합니다.

사전 요구 사항

- 디버그하려는 애플리케이션 또는 라이브러리가 시스템에 설치되어 있어야 합니다.
- **GDB** 및 **debuginfo-install** 툴이 시스템에 설치되어 있어야 합니다. 자세한 내용은 애플리케이션 디버그 설정을 참조하십시오.
- **debuginfo** 및 **debugsource** 패키지를 제공하는 채널은 시스템에서 구성하고 활성화해야 합니다.

절차

1. 디버깅하려는 애플리케이션 또는 라이브러리에 **GDB** 연결을 시작합니다. **GDB**는 누락된 디버깅 정보를 자동으로 인식하며 실행할 명령을 제안합니다.

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no
debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. **exit GDB: q** 를 입력하고 **Enter** 를 사용하여 확인합니다.

```
(gdb) q
```

3. **GDB**에서 제안한 명령을 실행하여 필요한 **debuginfo** 패키지를 설치합니다.

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

dnf 패키지 관리 도구는 변경 사항에 대한 요약을 제공하고 확인을 요청하며 확인 후 필요한 모든 파일을 다운로드하여 설치합니다.

4. **case GDB**는 **debuginfo** 패키지를 제안할 수 없으며 애플리케이션 또는 라이브러리에 대한

debuginfo 패키지 가져오기에 설명된 절차를 수동으로 수행합니다.

추가 리소스

- *Red Hat Developer Toolset 사용자 가이드, 디버깅 정보 설치*
- *RHEL 시스템용 debuginfo 패키지를 다운로드하거나 설치하려면 어떻게 해야 합니까? - Red Hat Knowledgebase 솔루션*

19장. PERF로 사용 중인 CPU 조사

시스템의 성능 문제를 조사할 때 perf 툴을 사용하여 작업에 중점을 두기 위해 CPU를 식별하고 모니터링할 수 있습니다.

19.1. PERF 통계로 계산된 CPU 이벤트 표시

CPU 수 집계를 비활성화하여 perf stat 를 사용하여 계산된 CPU 이벤트를 표시할 수 있습니다. 이 기능을 사용하려면 -a 플래그를 사용하여 시스템 전체 모드에서 이벤트를 계산해야 합니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

- CPU 수 집계가 비활성화된 이벤트 수를 계산합니다.

```
# perf stat -a -A sleep seconds
```

이전 예제는 CPU0 부터 각 개별 CPU를 통해 sleep 명령을 사용하여 지시한 대로 몇 초 동안 기록된 기본 하드웨어 및 소프트웨어 이벤트 세트의 수를 표시합니다. 따라서 사이클과 같은 이벤트를 지정하는 것이 유용할 수 있습니다.

```
# perf stat -a -A -e cycles sleep seconds
```

19.2. PERF 보고서에서 가져온 CPU 샘플 표시

perf record 명령은 성능 데이터를 샘플링하고 이 데이터를 perf report 명령으로 읽을 수 있는 perf.data 파일에 저장합니다. perf record 명령은 항상 수행된 CPU 샘플을 기록합니다. 이 정보를 표시하도록 perf 보고서를 구성할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

- 현재 디렉토리에 perf 레코드로 생성된 perf.data 파일이 있습니다. perf.data 파일이 루트 액세스 권한으로 생성된 경우 루트 액세스 권한을 사용하여 perf 보고서를 실행해야 합니다.

절차

- CPU를 기준으로 정렬하는 동안 추가 분석을 위해 perf.data 파일의 내용을 표시합니다.

```
# perf report --sort cpu
```

  - CPU 및 명령으로 정렬하여 CPU 시간이 소비되는 위치에 대한 자세한 정보를 표시할 수 있습니다.

```
# perf report --sort cpu,comm
```

    이 예제에서는 오버헤드 사용량의 내림차순으로 총 오버헤드로 모니터링되는 모든 CPU의 명령을 나열하고 명령이 실행된 CPU를 식별합니다.

추가 리소스

- perf를 사용하여 성능 프로파일 기록 및 분석

19.3. PERF를 사용하여 프로파일링하는 동안 특정 CPU 표시

시스템을 실시간으로 프로파일링 하는 동안 특정 CPU 및 관련 사용량을 표시하도록 상단 을 구성할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

- CPU를 기준으로 정렬하는 동안 perf 최상위 인터페이스를 시작합니다.

```
# perf top --sort cpu
```

    이 예제에서는 CPU 및 해당 오버헤드 사용량을 실시간으로 내림차순으로 나열합니다.

  -

CPU 및 명령으로 CPU 시간이 소비되는 위치에 대한 자세한 정보를 정렬할 수 있습니다.

```
# perf top --sort cpu,comm
```

이 예제에서는 오버헤드 사용량의 내림차순으로 총 오버헤드별 명령을 나열하고 실시간으로 명령이 실행된 CPU를 식별합니다.

19.4. PERF 레코드 및 PERF 보고서를 사용하여 특정 CPU 모니터링

관심 있는 특정 CPU만 샘플하도록 perf 레코드 를 구성하고 추가 분석을 위해 perf 보고서를 사용하여 생성된 perf.data 파일을 분석할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

1. 샘플과 특정 CPU의 성능 데이터를 기록하여 perf.data 파일을 생성합니다.

   - 쉼표로 구분된 CPU 목록 사용:

     ```
     # perf record -C 0,1 sleep seconds
     ```

     이전 예제에서는 sleep 명령을 사용하여 지시한 시간 동안 CPU 0 및 1의 데이터를 몇 초 동안 샘플링하고 기록합니다.

   - 다양한 CPU 사용:

     ```
     # perf record -C 0-2 sleep seconds
     ```

     이전 예제에서는 sleep 명령의 사용에 따라 지정된 시간 동안 CPU 0에서 2 사이의 모든 CPU의 데이터를 샘플링하고 기록합니다.

2. *추가 분석을 위해 **perf.data** 파일의 내용을 표시합니다.*

```
# perf report
```

*이 예제에서는 **perf.data** 의 내용을 표시합니다. 여러 **CPU**를 모니터링하고 샘플한 **CPU** 데이터를 알고자 하는 경우, **perf** 보고서에서 가져온 **CPU** 샘플 표시를 참조하십시오.*

20장. PERF로 애플리케이션 성능 모니터링

이 섹션에서는 **perf** 툴을 사용하여 애플리케이션 성능을 모니터링하는 방법에 대해 설명합니다.

20.1. 실행 중인 프로세스에 PERF 레코드 연결

실행 중인 프로세스에 **perf** 레코드 를 연결할 수 있습니다. 그러면 지정된 프로세스에 샘플 및 레코드 성능 데이터만 내리도록 **perf** 레코드에 지시합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

- 실행 중인 프로세스에 **perf** 레코드 를 연결합니다.

```
$ perf record -p ID1,ID2 sleep seconds
```

이전 예제에서는 **sleep** 명령을 사용하여 지시한 시간 동안 프로세스 **ID**의 **ID1** 및 **ID2** 를 사용하여 프로세스의 성능 데이터를 샘플링하고 기록합니다. 특정 스레드에서 이벤트를 기록하도록 **perf** 를 구성할 수도 있습니다.

```
$ perf record -t ID1,ID2 sleep seconds
```

참고

-t 플래그를 사용하고 스레드 **ID**를 오케스트레이션하는 경우, **perf** 는 기본적으로 상속을 비활성화합니다. **--inherit** 옵션을 추가하여 상속을 활성화할 수 있습니다.

20.2. PERF 레코드를 사용하여 호출 그래프 데이터 캡처

성능 프로필의 다른 함수를 호출하는 함수가 기록되도록 **perf** 레코드 도구를 구성할 수 있습니다. 이렇게 하면 여러 프로세스가 동일한 함수를 호출하는 경우 성능 장애를 확인하는 데 도움이 됩니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

- --call-graph 옵션을 사용하여 샘플 및 성능 데이터를 기록합니다.

```
$ perf record --call-graph method command
```

  - 명령을 샘플 데이터 중 데이터로 바꿉니다. 명령을 지정하지 않으면 Perf 레코드 는 Ctrl+C.

  - method 를 다음의 unwinding 방법 중 하나로 바꿉니다.

    fp

    프레임 포인터 메서드를 사용합니다. GCC 옵션으로 빌드된 바이너리와 같은 컴파일러 최적화에 따라 --fomit-frame-pointer.

    dwarf

    DWARF 호출 프레임 정보를 사용하여 스택의 배율을 해제합니다.

    lbr

    Intel 프로세서의 마지막 분기 레코드 하드웨어를 사용합니다.

추가 리소스

- perf-record(1) man page

20.3. PERF 보고서를 사용하여 PERF.DATA 분석

perf 보고서를 사용하여 perf.data 파일을 표시하고 분석할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

- *현재 디렉터리에는 **perf.data** 파일이 있습니다.*

- ***perf.data** 파일이 루트 액세스 권한으로 생성된 경우 루트 액세스 권한을 사용하여 **perf** 보고서를 실행해야 합니다.*

절차

- *추가 분석을 위해 **perf.data** 파일의 내용을 표시합니다.*

```
# perf report
```

이 명령은 다음과 유사한 출력을 표시합니다.

```
Samples: 2K of event 'cycles', Event count (approx.): 235462960
Overhead  Command          Shared Object                     Symbol
   2.36%  kswapd0          [kernel.kallsyms]                 [k] page_vma_mapped_walk
   2.13%  sssd_kcm         libc-2.28.so                      [.] memset_avx2_erms 2.13% perf
[kernel.kallsyms] [k] smp_call_function_single 1.53% gnome-shell libc-2.28.so [.]
strcmp_avx2
   1.17%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_hash_table_lookup
   0.93%  Xorg             libc-2.28.so                      [.] memmove_avx_unaligned_erms 0.89%
gnome-shell libgobject-2.0.so.0.5600.4 [.] g_object_unref 0.87% kswapd0 [kernel.kallsyms]
[k] page_referenced_one 0.86% gnome-shell libc-2.28.so [.] memmove_avx_unaligned_erms
   0.83%  Xorg             [kernel.kallsyms]                 [k] alloc_vmap_area
   0.63%  gnome-shell      libglib-2.0.so.0.5600.4           [.] g_slice_alloc
   0.53%  gnome-shell      libgirepository-1.0.so.1.0.0      [.] g_base_info_unref
   0.53%  gnome-shell      ld-2.28.so                        [.] _dl_find_dso_for_object
   0.49%  kswapd0          [kernel.kallsyms]                 [k] vma_interval_tree_iter_next
   0.48%  gnome-shell      libpthread-2.28.so                [.] pthread_getspecific 0.47% gnome-
shell libgirepository-1.0.so.1.0.0 [.] 0x0000000000013b1d 0.45% gnome-shell libglib-
2.0.so.0.5600.4 [.] g_slice_free1 0.45% gnome-shell libgobject-2.0.so.0.5600.4 [.]
g_type_check_instance_is_fundamentally_a 0.44% gnome-shell libc-2.28.so [.] malloc 0.41%
swapper [kernel.kallsyms] [k] apic_timer_interrupt 0.40% gnome-shell ld-2.28.so [.]
_dl_lookup_symbol_x 0.39% kswapd0 [kernel.kallsyms] [k]
raw_callee_save___pv_queued_spin_unlock
```

추가 리소스

- ***perf-report(1)** 도움말 페이지*

21장. PERF를 사용하여 UPROBES 생성

21.1. PERF를 사용하여 함수 수준에서 UPROBES 생성

perf 도구를 사용하여 프로세스 또는 애플리케이션의 임의의 시점에서 동적 추적 포인트를 생성할 수 있습니다. 그런 다음 이러한 추적점은 프로세스 또는 애플리케이션 동작을 더 잘 이해하기 위해 perf stat 및 perf 레코드와 같은 다른 perf 툴과 함께 사용할 수 있습니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

1. 프로세스 또는 애플리케이션 내에서 관심 있는 위치에서 모니터링에 관심이 있는 프로세스 또는 애플리케이션에 uprobe를 생성합니다.

```
# perf probe -x /path/to/executable -a function
Added new event:
  probe_executable:function   (on function in /path/to/executable)

You can now use it in all perf tools, such as:

        perf record -e probe_executable:function -aR sleep 1
```

추가 리소스

- perf-probe 매뉴얼 페이지
- perf를 사용하여 성능 프로파일 기록 및 분석
- perf stat를 사용하여 프로세스 실행 중 이벤트 계산

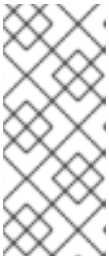
21.2. PERF를 사용하여 함수 내에서 줄에 UPROBES 생성

그런 다음 이러한 추적점은 프로세스 또는 애플리케이션 동작을 더 잘 이해하기 위해 perf stat 및 perf 레코드와 같은 다른 perf 툴과 함께 사용할 수 있습니다.

사전 요구 사항

- *perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.*
- *실행 파일의 디버깅 기호를 가져옵니다.Gets the debugging symbols for your executable:*

```
# objdump -t ./your_executable | head
```



참고

이렇게 하려면 실행 파일의 debuginfo 패키지를 설치하거나, 실행 파일이 로컬에서 개발한 애플리케이션인 경우, 디버깅 정보를 사용하여 애플리케이션을 컴파일해야 하며 GCC의 -g 옵션을 사용해야 합니다.

절차

1. *uprobe를 배치할 수 있는 함수 행을 확인합니다.*

```
$ perf probe -x ./your_executable -L main
```

이 명령의 출력은 다음과 유사합니다.

```
<main@/home/user/my_executable:0>
          0  int main(int argc, const char **argv)
          1  {
                 int err;
                 const char *cmd;
                 char sbuf[STRERR_BUFSIZE];

                 /* libsubcmd init */
          7      exec_cmd_init("perf", PREFIX, PERF_EXEC_PATH,
EXEC_PATH_ENVIRONMENT);
          8      pager_init(PERF_PAGER_ENVIRONMENT);
```

2. *원하는 함수 행에 대한 uprobe를 생성합니다.*

```
# perf probe -x ./my_executable main:8
Added new event:
      probe_my_executable:main_L8   (on main:8 in /home/user/my_executable)
```

> You can now use it in all perf tools, such as:
>
> perf record -e probe_my_executable:main_L8 -aR sleep 1

21.3. UPROBES에서 기록된 데이터의 PERF 스크립트 출력

uprobes를 사용하여 수집된 데이터를 분석하는 일반적인 방법은 perf script 명령을 사용하여 perf.data 파일을 읽고 기록된 워크로드의 세부 추적을 표시하는 것입니다.

perf 스크립트 예제 출력에서 * uprobe가 함수에 isprime() 가 추가되었습니다. my_prog * a 는 uprobe에 추가된 함수 인수입니다. 또는 uprobe를 추가하는 코드 범위에 표시되는 임의의 변수일 수 있습니다.

```
# perf script
   my_prog  1367 [007] 10802159.906593: probe_my_prog:isprime: (400551) a=2
   my_prog  1367 [007] 10802159.906623: probe_my_prog:isprime: (400551) a=3
   my_prog  1367 [007] 10802159.906625: probe_my_prog:isprime: (400551) a=4
   my_prog  1367 [007] 10802159.906627: probe_my_prog:isprime: (400551) a=5
   my_prog  1367 [007] 10802159.906629: probe_my_prog:isprime: (400551) a=6
   my_prog  1367 [007] 10802159.906631: probe_my_prog:isprime: (400551) a=7
   my_prog  1367 [007] 10802159.906633: probe_my_prog:isprime: (400551) a=13
   my_prog  1367 [007] 10802159.906635: probe_my_prog:isprime: (400551) a=17
   my_prog  1367 [007] 10802159.906637: probe_my_prog:isprime: (400551) a=19
```

22장. PERF MEM를 사용하여 메모리 액세스 프로파일링

perf mem 명령을 사용하여 시스템의 메모리 액세스를 샘플링할 수 있습니다.

22.1. PERF MEM의 목적

perf 툴의 mem 하위 명령을 사용하면 메모리 액세스 샘플링(로드 및 저장소)을 사용할 수 있습니다. perf mem 명령은 메모리 대기 시간, 메모리 액세스 유형, 캐시 적중 및 누락을 유발하는 기능, 데이터 기호를 기록하여 이러한 히트 및 누락에 대한 메모리 위치를 제공합니다.

22.2. PERF MEM를 사용하여 샘플링 메모리 액세스

이 절차에서는 perf mem 명령을 사용하여 시스템에서 메모리 액세스를 샘플하는 방법을 설명합니다. 이 명령은 perf 레코드 및 perf 보고서와 동일한 옵션을 사용하며 mem 하위 명령에 독점적인 일부 옵션을 사용합니다. 기록된 데이터는 나중에 분석을 위해 현재 디렉터리에 있는 perf.data 파일에 저장됩니다.

사전 요구 사항

- perf 설치에 설명된 대로 perf 사용자 공간 도구가 설치되어 있습니다.

절차

1. 메모리 액세스 샘플:

```
# perf mem record -a sleep seconds
```

이 예제에서는 sleep 명령으로 지정된 대로 모든 CPU에서 몇 초 동안 메모리에 액세스합니다. 메모리 액세스 데이터를 샘플하려는 명령에 대해 sleep 명령을 교체할 수 있습니다. 기본적으로 perf mem 는 메모리가 로드 및 저장소를 모두 샘플링합니다. t 옵션을 사용하여 하나의 메모리 작업만 선택하고 perf mem 와 레코드 간에 "load" 또는 "store"를 지정할 수 있습니다. 로드의 경우 메모리 계층 수준, TLB 메모리 액세스, 버스 스노크 및 메모리 잠금에 대한 정보가 캡처됩니다.

2. 분석을 위해 perf.data 파일을 엽니다.

```
# perf mem report
```

예제 명령을 사용한 경우 출력은 다음과 같습니다.

```
Available samples
35k cpu/mem-loads,ldlat=30/P
54k cpu/mem-stores/P
```

***cpu/mem-loads,ldlat=30/P** 행은 메모리 로드를 통해 수집된 데이터를 나타내며 **cpu/mem-stores/P** 행은 메모리 저장소를 통해 수집된 데이터를 나타냅니다. 관심 카테고리를 강조 표시하고 **Enter** 를 눌러 데이터를 확인합니다.*

```
Samples: 35K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 4067062
Overhead       Samples  Local Weight  Memory access            Symbol
Shared Object                Data Symbol                                  Data Object
Snoop       TLB access           Locked
  0.07%           29  98          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0         [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit         No
  0.06%           26  97          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0         [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit         No
  0.06%           25  96          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0         [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit         No
  0.06%           1  2325        Uncached or N/A hit     [k] pci_azx_readl
[kernel.kallsyms]            [k] 0xffffb092c06e9084
[kernel.kallsyms]                 None        L1 or L2 hit         No
  0.06%           1  2247        Uncached or N/A hit     [k] pci_azx_readl
[kernel.kallsyms]            [k] 0xffffb092c06e8164
[kernel.kallsyms]                 None        L1 or L2 hit         No
  0.05%           1  2166        L1 or L1 hit            [.] 0x00000000038140d6
libxul.so                    [.] 0x00007ffd7b84b4a8                       [stack]
None        L1 or L2 hit         No
  0.05%           1  2117        Uncached or N/A hit     [k] check_for_unclaimed_mmio
[kernel.kallsyms]            [k] 0xffffb092c1842300
[kernel.kallsyms]                 None        L1 or L2 hit         No
  0.05%           22  95          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0         [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit         No
  0.05%           1  1898        L1 or L1 hit            [.] 0x0000000002a30e07
libxul.so                    [.] 0x00007f610422e0e0                       anon
None        L1 or L2 hit         No
  0.05%           1  1878        Uncached or N/A hit     [k] pci_azx_readl
[kernel.kallsyms]            [k] 0xffffb092c06e8164
[kernel.kallsyms]                 None        L2 miss              No
  0.04%           18  94          L1 or L1 hit            [.] 0x000000000000a255
libspeexdsp.so.1.5.0         [.] 0x00007f697a3cd0f0                       anon
None        L1 or L2 hit         No
  0.04%           1  1593        Local RAM or RAM hit    [.] 0x00000000026f907d
libxul.so                    [.] 0x00007f3336d50a80                       anon
Hit         L2 miss              No
  0.03%           1  1399        L1 or L1 hit            [.] 0x00000000037cb5f1
libxul.so                    [.] 0x00007fbe81ef5d78                       libxul.so
```

```
None       L1 or L2 hit        No
  0.03%          1  1229       LFB or LFB hit         [.] 0x0000000002962aad
libxul.so                [.] 0x00007fb6f1be2b28                          anon
None       L2 miss             No
  0.03%          1  1202       LFB or LFB hit         [.] __pthread_mutex_lock
libpthread-2.29.so       [.] 0x00007fb75583ef20                          anon
None       L1 or L2 hit        No
  0.03%          1  1193       Uncached or N/A hit    [k] pci_azx_readl
[kernel.kallsyms]        [k] 0xffffb092c06e9164
[kernel.kallsyms]             None       L2 miss             No
  0.03%          1  1191       L1 or L1 hit           [k] azx_get_delay_from_lpib
[kernel.kallsyms]        [k] 0xffffb092ca7efcf0
[kernel.kallsyms]             None       L1 or L2 hit        No
```

또는 데이터를 표시할 때 다양한 관심 측면을 조사하도록 결과를 정렬할 수 있습니다. 예를 들어, 메모리 로드를 통해 데이터를 샘플링 기간 동안 처리하는 오버헤드 순서의 내림차순으로 발생하는 메모리 액세스 유형별로 정렬하려면 다음을 수행합니다.

```
# perf mem -t load report --sort=mem
```

예를 들어 출력은 다음과 같습니다.

```
Samples: 35K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 40670
Overhead       Samples  Memory access
  31.53%          9725  LFB or LFB hit
  29.70%         12201  L1 or L1 hit
  23.03%          9725  L3 or L3 hit
  12.91%          2316  Local RAM or RAM hit
   2.37%           743  L2 or L2 hit
   0.34%             9  Uncached or N/A hit
   0.10%            69  I/O or N/A hit
   0.02%           825  L3 miss
```

추가 리소스

- **perf-mem(1)** 도움말 페이지.

22.3. PERF MEM 보고서 출력 해석

수정자 없이 **perf mem report** 명령을 실행하여 표시되는 표는 여러 열로 데이터를 정렬합니다.

'Overhead' 열

이 특정 함수에서 수집된 전체 샘플의 백분율을 나타냅니다.

<Samples> 열

해당 행에 의해 계산된 샘플 수를 표시합니다.

<Local Weight> 열

프로세서 코어 사이클에서 액세스 대기 시간을 표시합니다.

'Memory Access' 열

발생한 메모리 액세스 유형을 표시합니다.

<Symbol> 열

함수 이름 또는 기호를 표시합니다.

'공유 오브젝트' 열

샘플이 들어오는 **ELF** 이미지의 이름을 표시합니다(샘플에서 가져온 샘플 **[kernel.kallsyms] is used).**

'Data Symbol' 열

행이 대상으로 한 메모리 위치의 주소를 표시합니다.

중요

일반적으로 액세스하는 메모리 또는 스택 메모리의 동적 할당으로 인해 **'Data Symbol'** 열에 원시 주소가 표시됩니다.

<Snoop> 열

버스 트랜잭션을 표시합니다.

'TLB 액세스' 열

TLB 메모리 액세스를 표시합니다.

<Locked> 열

함수가 잠겼거나 메모리가 잠겼는지 여부를 나타냅니다.

기본 모드에서 함수는 가장 높은 오버헤드가 먼저 표시된 순서를 사용하여 내림차순으로 정렬됩니다.

23장. 잘못된 공유 감지

잘못된 공유는 **Symmetric Multi Processing(SMP)** 시스템의 프로세서 코어가 다른 프로세서에서 사용 중인 동일한 캐시 라인의 데이터 항목을 수정하여 프로세서 간에 공유되지 않는 다른 데이터 항목에 액세스할 때 발생합니다.

이러한 초기 수정을 위해서는 캐시 라인을 사용하는 다른 프로세서가 해당 사본을 무효화하고 업데이트된 프로세서를 요청할 필요가 없으며 수정된 데이터 항목의 업데이트된 버전에 액세스할 필요가 없습니다.

perf c2c 명령을 사용하여 잘못된 공유를 탐지할 수 있습니다.

23.1. PERF C2C의 목적

perf 툴의 **c2c** 하위 명령을 사용하면 **C2C(공유 데이터 캐시)** 분석을 활성화합니다. **perf c2c** 명령을 사용하여 캐시 라인 경합을 검사하여 **true** 및 **false** 공유를 모두 탐지할 수 있습니다.

캐시 라인 경합은 **SMP(Symmetric Multi Processing)** 시스템의 프로세서 코어가 다른 프로세서에서 사용 중인 동일한 캐시 라인의 데이터 항목을 수정할 때 발생합니다. 그런 다음 이 캐시 라인을 사용하는 다른 모든 프로세서는 사본을 무효화하고 업데이트된 프로세서를 요청해야 합니다. 이로 인해 성능이 저하될 수 있습니다.

perf c2c 명령은 다음 정보를 제공합니다.

- 경합이 감지된 캐시 라인
- 데이터 읽기 및 쓰기 프로세스
- 경합을 유발하는 지침
- 경합과 관련된 **NUMA(Non-Uniform Memory Access)** 노드

23.2. PERF C2C를 사용하여 캐시 라인 경합 감지

perf c2c 명령을 사용하여 시스템의 캐시 라인 경합을 감지합니다.

perf c2c 명령은 **c2c** 하위 명령에만 사용할 수 있는 일부 옵션과 동일한 옵션을 지원합니다. 기록된 데이터는 나중에 분석을 위해 현재 디렉터리에 있는 **perf.data** 파일에 저장됩니다.

사전 요구 사항

- **perf** 사용자 공간 도구가 설치됩니다. 자세한 내용은 perf 설치를 참조하십시오.

절차

- **perf c2c** 를 사용하여 캐시 라인 경합을 감지합니다.

```
# perf c2c record -a sleep seconds
```

이 예제에서는 **sleep** 명령으로 지정된 시간 동안 모든 CPU에서 캐시 라인 경합 데이터를 샘플링하고 기록합니다. **sleep** 명령을 통해 캐시 라인 경합 데이터를 수집하려는 모든 명령으로 교체할 수 있습니다.

추가 리소스

- **perf-c2c(1) man page**

23.3. PERF C2C 레코드로 기록된 PERF.DATA 파일 시각화

이 절차에서는 **perf c2c** 명령을 사용하여 기록된 **perf.data** 파일을 시각화하는 방법을 설명합니다.

사전 요구 사항

- **perf** 사용자 공간 도구가 설치됩니다. 자세한 내용은 **Installing perf** 를 참조하십시오.

- **perf c2c** 명령을 사용하여 기록된 **perf.data** 파일은 현재 디렉터리에서 사용할 수 있습니다. 자세한 내용은 **perf c2c**를 사용하여 캐시 라인 경합 감지 를 참조하십시오.

절차

1.

*추가 분석을 위해 **perf.data** 파일을 엽니다.*

```
# perf c2c report --stdio
```

*이 명령은 **perf.data** 파일을 터미널 내의 여러 그래프로 시각화합니다.*

```
=================================================
        Trace Event Information
=================================================
Total records                     :     329219
Locked Load/Store Operations      :      14654
Load Operations                   :      69679
Loads - uncacheable               :          0
Loads - IO                        :          0
Loads - Miss                      :       3972
Loads - no mapping                :          0
Load Fill Buffer Hit              :      11958
Load L1D hit                      :      17235
Load L2D hit                      :         21
Load LLC hit                      :      14219
Load Local HITM                   :       3402
Load Remote HITM                  :      12757
Load Remote HIT                   :       5295
Load Local DRAM                   :        976
Load Remote DRAM                  :       3246
Load MESI State Exclusive         :       4222
Load MESI State Shared            :          0
Load LLC Misses                   :      22274
LLC Misses to Local DRAM          :       4.4%
LLC Misses to Remote DRAM         :      14.6%
LLC Misses to Remote cache (HIT)  :      23.8%
LLC Misses to Remote cache (HITM) :      57.3%
Store Operations                  :     259539
Store - uncacheable               :          0
Store - no mapping                :         11
Store L1D Hit                     :     256696
Store L1D Miss                    :       2832
No Page Map Rejects               :       2376
Unable to parse data source       :          1

=================================================
   Global Shared Cache Line Event Information
=================================================
Total Shared Cache Lines          :         55
Load HITs on shared lines         :      55454
Fill Buffer Hits on shared lines  :      10635
L1D hits on shared lines          :      16415
L2D hits on shared lines          :          0
LLC hits on shared lines          :       8501
Locked Access on shared lines     :      14351
Store HITs on shared lines        :     109953
Store L1D hits on shared lines    :     109449
```

```
 Total Merged records            :     126112

=================================================
            c2c details
=================================================
 Events                         : cpu/mem-loads,ldlat=30/P
                                : cpu/mem-stores/P
 Cachelines sort on             : Remote HITMs
 Cacheline data groupping       : offset,pid,iaddr

=================================================
   Shared Data Cache Line Table
=================================================
#
#                        Total      Rmt  ----- LLC Load Hitm -----  ---- Store Reference ----  ---
Load Dram ----      LLC    Total  ----- Core Load Hit -----  -- LLC Load Hit --
# Index          Cacheline  records     Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss
Lcl       Rmt  Ld Miss    Loads       FB       L1        L2       Llc       Rmt
# .....  ..................  .......  .......  .......  .......  .......  .......  .......  ........  ........  .......  .......  .......
.......  .......  ........  ........
#
    0          0x602180   149904   77.09%    12103     2269     9834   109504   109036      468
727     2657    13747    40400     5355    16154        0     2875       529
    1          0x602100    12128   22.20%     3951     1119     2832        0        0        0       65
200     3749    12128     5096      108        0     2056      652
    2  0xffff883ffb6a7e80      260    0.09%       15        3       12      161      161        0        1
1       15       99       25       50        0        6        1
    3  0xffffffff81aec000      157    0.07%        9        0        9        1        0        1        0        7
20      156       50       59        0       27        4
    4  0xffffffff81e3f540      179    0.06%        9        1        8      117       97       20        0
10       25       62       11        1        0       24        7

=================================================
    Shared Cache Line Distribution Pareto
=================================================
#
#       ----- HITM -----  -- Store Refs --        Data address                          ---------- cycles
----------       cpu                                Shared
#   Num      Rmt      Lcl   L1 Hit  L1 Miss              Offset      Pid        Code address  rmt
hitm  lcl hitm      load       cnt              Symbol              Object                Source:Line
Node{cpu list}
# .....  .......  .......  .......  .......  ...................  .......  ...................  ........  ........  ........  ........
.....................  .....................  ............................  ....
#
  ------------------------------------------------------------
    0     9834     2269   109036      468           0x602180
  ------------------------------------------------------------
        65.51%   55.88%   75.20%    0.00%                0x0    14604          0x400b4f     27161
26039     26017        9  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:144   0{0-1,4}  1{24-25,120}  2{48,54}  3{169}
    0.41%    0.35%    0.00%    0.00%                0x0    14604          0x400b56     18088
12601     26671        9  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:145   0{0-1,4}  1{24-25,120}  2{48,54}  3{169}
    0.00%    0.00%   24.80%  100.00%                0x0    14604          0x400b61         0        0
0         9  [.] read_write_func  no_false_sharing.exe  false_sharing_example.c:145   0{0-
1,4}  1{24-25,120}  2{48,54}  3{169}
```

```
   7.50%   9.92%   0.00%   0.00%            0x20    14604         0x400ba7     2470
1729     1897       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:154   1{122}  2{144}
  17.61%  20.89%   0.00%   0.00%            0x28    14604         0x400bc1     2294
1575     1649       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:158   2{53}  3{170}
   8.97%  12.96%   0.00%   0.00%            0x30    14604         0x400bdb     2325
1897     1828       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:162   0{96}  3{171}

 -------------------------------------------------------------
    1    2832     1119       0       0          0x602100
 -------------------------------------------------------------
  29.13%  36.19%   0.00%   0.00%            0x20    14604         0x400bb3     1964
1230     1788       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:155   1{122}  2{144}
  43.68%  34.41%   0.00%   0.00%            0x28    14604         0x400bcd     2274
1566     1793       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:159   2{53}  3{170}
  27.19%  29.40%   0.00%   0.00%            0x30    14604         0x400be7     2045
1247     2011       2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:163   0{96}  3{171}
```

23.4. PERF C2C 보고서 출력의 해석

이 섹션에서는 **perf c2c report** 명령의 출력을 해석하는 방법을 설명합니다.

perf c2c 보고서를 실행하여 표시되는 시각화 **--stdio** 명령은 데이터를 여러 테이블로 정렬합니다.

추적 이벤트 정보

이 표는 모든 로드 및 저장소 샘플에 대한 높은 수준의 요약을 제공하며, 이는 **perf c2c record** 명령으로 수집됩니다.

글로벌 공유 캐시 라인 이벤트 정보

이 테이블은 공유 캐시 라인에 대한 통계를 제공합니다.

c2c 세부 정보

이 표는 샘플된 이벤트 및 시각화에서 **perf c2c** 보고서 데이터를 구성하는 방법에 대한 정보를 제공합니다.

공유 데이터 캐시 라인 테이블

이 표는 잘못된 공유가 감지되고 기본적으로 캐시 라인별로 감지된 원격 **Hitm** 의 양으로 내림차순으로 정렬된 **hottest** 캐시 행에 대한 한 줄 요약을 제공합니다.

공유 캐시 라인 배포 Pareto

이 표는 경합이 발생하는 각 캐시 라인에 대한 다양한 정보를 제공합니다.

- 캐시 줄은 NUM 열에서 번호가 매겨지며 0 부터 시작합니다.
- 각 캐시 라인의 가상 주소는 데이터 주소 Offset 열에 포함되어 있으며 그 다음에 다른 액세스가 발생한 캐시 라인의 오프셋이 포함됩니다.
- Pid 열에는 프로세스 ID가 포함됩니다.
- Code Address 열에는 명령 포인터 코드 주소가 포함되어 있습니다.
- 사이클 레이블 아래의 열에는 평균 부하 대기 시간이 표시됩니다.
- cpu cnt 열에는 제공된 CPU 샘플 수(즉, 지정된 위치에서 인덱싱된 데이터를 기다리는 CPU 수)가 표시됩니다.
- Symbol 열에 함수 이름 또는 기호가 표시됩니다.
- Shared Object 열에는 샘플이 들어오는 ELF 이미지의 이름이 표시됩니다(이름 [kernel.kallsyms]은 커널에서 가져올 때 사용됩니다).
- Source:Line 열에는 소스 파일 및 행 번호가 표시됩니다.
- Node{cpu list} 열에는 각 노드에 대해 제공된 특정 CPU 샘플이 표시됩니다.

23.5. PERF C2C로 잘못된 공유 탐지

이 절차에서는 perf c2c 명령을 사용하여 잘못된 공유를 감지하는 방법을 설명합니다.

사전 요구 사항

- *perf 사용자 공간 도구가 설치됩니다. 자세한 내용은 perf 설치를 참조하십시오.*

- *perf c2c 명령을 사용하여 기록된 perf.data 파일은 현재 디렉터리에서 사용할 수 있습니다. 자세한 내용은 perf c2c를 사용하여 캐시 라인 경합 감지 를 참조하십시오.*

절차

1. *추가 분석을 위해 perf.data 파일을 엽니다.*

```
# perf c2c report --stdio
```

그러면 터미널에서 perf.data 파일이 열립니다.

2. *"Trace Event Information" 표에서 NFD Mises to Remote Cache (HITM)에 대한 값이 포함된 행을 찾습니다.*

NFD Mises to Remote Cache (HITM) 행의 값 열에 있는 백분율은 수정된 캐시 줄의 NUMA 노드에서 발생했으며 키 표시기 거짓 공유가 발생했음을 나타내는 <.> 누락의 백분율을 나타냅니다.

```
=================================================
        Trace Event Information
=================================================
 Total records                 :     329219
 Locked Load/Store Operations  :      14654
 Load Operations               :      69679
 Loads - uncacheable           :          0
 Loads - IO                    :          0
 Loads - Miss                  :       3972
 Loads - no mapping            :          0
 Load Fill Buffer Hit          :      11958
 Load L1D hit                  :      17235
 Load L2D hit                  :         21
 Load LLC hit                  :      14219
 Load Local HITM               :       3402
 Load Remote HITM              :      12757
 Load Remote HIT               :       5295
 Load Local DRAM               :        976
 Load Remote DRAM              :       3246
 Load MESI State Exclusive     :       4222
 Load MESI State Shared        :          0
 Load LLC Misses               :      22274
 LLC Misses to Local DRAM      :       4.4%
 LLC Misses to Remote DRAM     :      14.6%
```

```
LLC Misses to Remote cache (HIT)  :      23.8%
LLC Misses to Remote cache (HITM) : 57.3%
Store Operations                  :    259539
Store - uncacheable               :         0
Store - no mapping                :        11
Store L1D Hit                     :    256696
Store L1D Miss                    :      2832
No Page Map Rejects               :      2376
Unable to parse data source       :         1
```

3. *공유 데이터 캐시 라인 테이블의* **NFD Load Hitm** *필드의* **Rmt** *열을 검사합니다.*

```
=================================================
        Shared Data Cache Line Table
=================================================
#
#                        Total      Rmt  ----- LLC Load Hitm -----  ---- Store Reference ----  ---
Load Dram ----      LLC    Total  ----- Core Load Hit -----  -- LLC Load Hit --
# Index          Cacheline  records    Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss
Lcl       Rmt  Ld Miss    Loads       FB       L1       L2       Llc       Rmt
# ..... ................... ....... ....... ....... ....... ....... ....... ....... ....... ........ ........ ....... ....... .......
....... ....... ........ ........
#
     0          0x602180   149904   77.09%    12103     2269     9834   109504   109036
468      727     2657    13747    40400     5355    16154       0     2875      529
     1          0x602100    12128   22.20%     3951     1119     2832        0        0        0       65
200     3749    12128     5096      108        0     2056      652
     2  0xffff883ffb6a7e80      260    0.09%       15        3       12      161      161        0        1
1       15       99       25       50        0        6        1
     3  0xffffffff81aec000      157    0.07%        9        0        9        1        0        1        0        7
20      156       50       59        0       27        4
     4  0xffffffff81e3f540      179    0.06%        9        1        8      117       97       20        0
10       25       62       11        1        0       24        7
```

이 테이블은 캐시 줄당 감지된 원격 **Hitm** *의 양으로 내림차순으로 정렬됩니다.* **NFD Load Hitm** *섹션의* **Rmt** *열에 있는 높은 수는* **false** *공유를 나타내며 잘못된 공유 활동을 디버깅하는 데 발생한 캐시 라인을 추가로 검사해야 합니다.*

24장. FIREAMEGRAPHS 시작하기

시스템 관리자는 **flamegraphs** 를 사용하여 **perf** 툴로 기록된 시스템 성능 데이터의 시각화를 생성할 수 있습니다. 소프트웨어 개발자는 **fireame graphs** 를 사용하여 **perf** 도구로 기록된 애플리케이션 성능 데이터의 시각화를 생성할 수 있습니다.

샘플링 스택 추적은 **perf** 툴을 사용하여 **CPU** 성능을 프로파일링하는 일반적인 기술입니다. 유감스럽게도, **perf** 를 사용한 스택 추적의 결과는 분석하는 데 매우 상세하고 인건비 집약적일 수 있습니다. **Flaamegraphs** 는 핫 코드 경로를 더 빠르고 쉽게 식별할 수 있도록 **perf** 로 기록된 데이터에서 생성되는 시각화입니다.

24.1. FIREAMEGRAPHS 설치

flamegraphs 사용을 시작하려면 필요한 패키지를 설치하십시오.

절차

- **firearn egraphs** 패키지를 설치합니다.

```
# dnf install js-d3-flame-graph
```

24.2. 전체 시스템에 불AMEGRAPHS 생성

이 절차에서는 **fireame graphs** 를 사용하여 전체 시스템에 대해 기록된 성능 데이터를 시각화하는 방법을 설명합니다.

사전 요구 사항

- 불**amegraphs** 는 불**amegraphs** *설치에* 설명된 대로 설치됩니다.
- **perf** 툴은 **perf** *설치에* 설명된 대로 설치됩니다.

절차

- 데이터를 기록하고 시각화를 생성합니다.

```
# perf script flamegraph -a -F 99 sleep 60
```

*이 명령은 **sleep** 명령을 사용하여 설명하는 것처럼 전체 시스템에 대해 성능 데이터를 샘플링하고 기록한 다음, 현재 활성 디렉터리에 저장될 시각화를 구성했습니다. 이 명령은 기본적으로 호출 그래프 데이터를 샘플링하고 **perf** 툴과 동일한 인수를 사용합니다. 이 경우 다음과 같습니다.*

-a

전체 시스템에 대한 데이터를 기록하도록 설정합니다.

-F

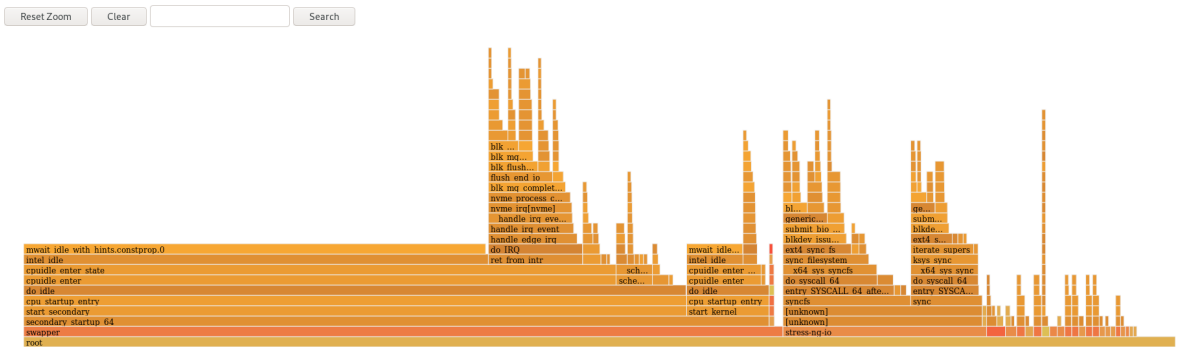
초당 샘플링 빈도를 설정하려면 다음을 수행합니다.

검증 단계

- *분석을 위해 생성된 시각화를 확인합니다.*

```
# xdg-open flamegraph.html
```

이 명령은 기본 브라우저에서 시각화를 엽니다.



24.3. 특정 프로세스에 대한 불AMEGRAPHS 생성

***flamegraphs** 를 사용하여 실행 중인 특정 프로세스에 대해 기록된 성능 데이터를 시각화할 수 있습니다.*

사전 요구 사항

- 불**amegraphs** 는 불**amegraphs** *설치에* 설명된 대로 설치됩니다.

- **perf** 툴은 **perf** *설치에* 설명된 대로 설치됩니다.

절차

- 데이터를 기록하고 시각화를 생성합니다.

```
# perf script flamegraph -a -F 99 -p ID1,ID2 sleep 60
```

이 명령은 **sleep** 명령을 사용하여 설명하는 대로 프로세스 **ID1** 및 **ID2**로 프로세스 **ID1** 및 **ID2**를 사용하여 프로세스의 성능 데이터를 기록한 다음 현재 활성 디렉터리에 저장될 시각화를 구성하며, 현재 활성 디렉터리에 저장된 시각화를 **coname graph.html** 로 구성합니다. 이 명령은 기본적으로 호출 그래프 데이터를 샘플링하고 **perf** 툴과 동일한 인수를 사용합니다. 이 경우 다음과 같습니다.

-a

전체 시스템에 대한 데이터를 기록하도록 설정합니다.

-F

초당 샘플링 빈도를 설정하려면 다음을 수행합니다.

-p

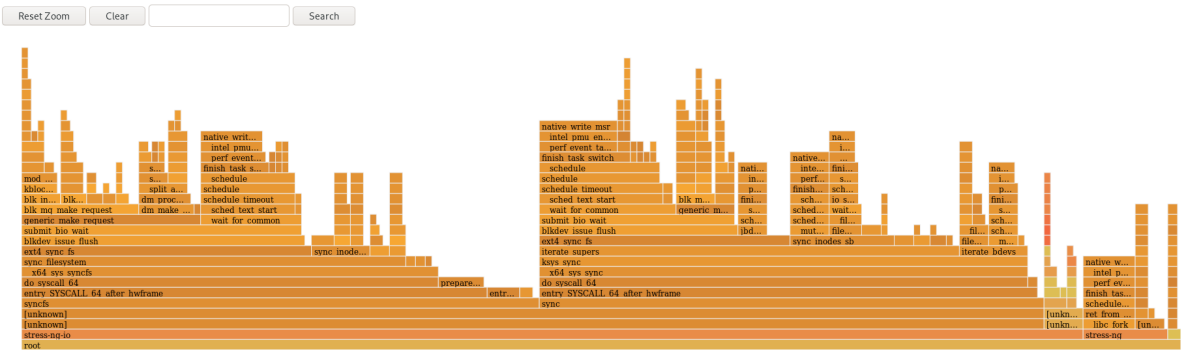
특정 프로세스 **ID**를 샘플로 지정하고 데이터를 기록하려면 다음을 수행합니다.

검증 단계

- 분석을 위해 생성된 시각화를 확인합니다.

```
# xdg-open flamegraph.html
```

이 명령은 기본 브라우저에서 시각화를 엽니다.



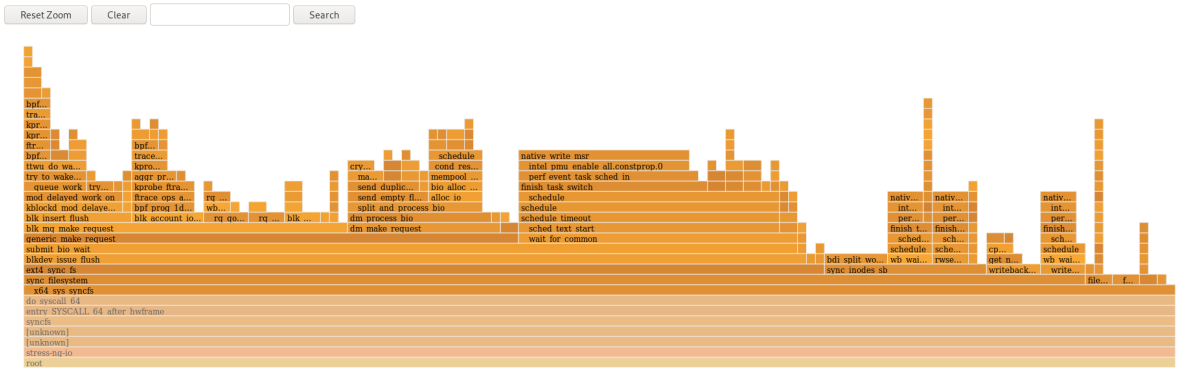
24.4. 불AMEGRAPHS 해석

*플로브그의 각 박스는 스택의 다른 기능을 나타냅니다. **y**축은 각 스택의 가장 높은 박스가 있는 스택의 깊이를 보여줍니다. 각 스택은 실제로 **on-CPU**이고 그 아래의 모든 것이 **ancestry**인 함수입니다. **X**축은 샘플 호출-그래픽 데이터의 채우기를 표시합니다.*

*지정된 행에 스택의 자식은 **x** 축을 따라 각 함수의 내림차순으로 가져온 샘플 수에 따라 표시됩니다. **x** 축은 통과 시간을 나타내지 않습니다. 개별 상자가 더 넓어지면 데이터가 샘플링되는 시점에 온-CPU 또는 온-**CPU ancestry**의 일부가되었습니다.*

절차

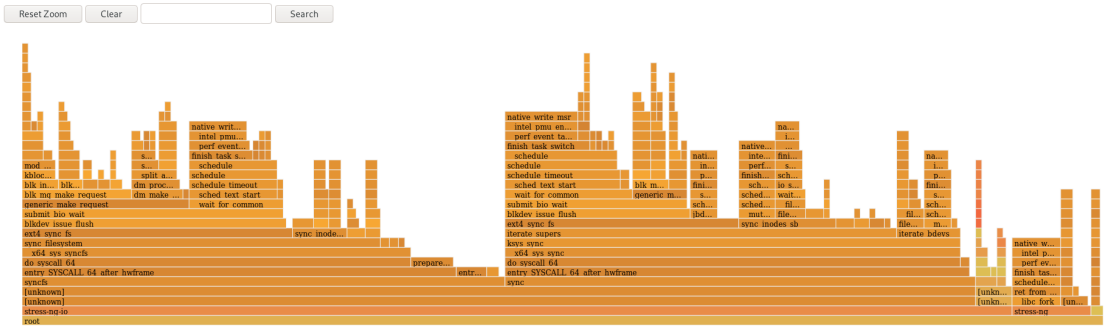
- *이전에 표시되지 않을 수 있는 함수 이름을 표시하고, 그 지정된 위치에서 스택을 확대하기 위해 플로브리 내의 상자에서 데이터 클릭을 추가로 조사합니다.*



- ***fireamegraph**의 기본 보기로 돌아가려면 확대/축소 재설정 을 클릭합니다.*

중요

*사용자 공간 함수를 나타내는 박스는 함수의 바이너리가 제거되었기 때문에 플로 즈 에서 알 수 없는 것으로 표시될 수 있습니다. 실행 파일의 **debuginfo** 패키지를 설치하거나 로컬에서 개발한 애플리케이션인 경우 디버깅 정보를 사용하여 애플리케이션을 컴파일해야 합니다. GCC에서 **-g** 옵션을 사용하여 이러한 상황에서 함수 이름 또는 기호를 표시합니다.*



추가 리소스

- *perf가 일부 함수 이름을 원시 함수 주소로 표시하는 이유*
- *디버깅 정보를 사용하여 디버깅 활성화*

25장. PERF 순환 버퍼를 사용하여 성능 병목 현상을 위한 프로세스 모니터링

시스템에서 실행되는 애플리케이션 일부 또는 특정 프로세스의 성능 병목 현상을 모니터링하기 위해 **perf** 툴을 사용하여 이벤트별 데이터 스냅샷을 사용하는 원형 버퍼를 생성할 수 있습니다. 이러한 경우, **perf** 는 지정된 이벤트가 감지된 경우에만 이후 분석을 위해 **perf.data** 파일에 데이터를 씁니다.

25.1. PERF를 사용하여 순환 버퍼 및 이벤트별 스냅샷

프로세스 또는 **perf** 의 적용에서 성능 문제를 조사할 때 특정 관심 발생 시 몇 시간 전의 데이터를 기록하는 것이 저렴하거나 적절하지 않을 수 있습니다. 이러한 경우, **perf** 레코드 를 사용하여 특정 이벤트 후 스냅샷을 가져오는 사용자 지정 순환 버퍼를 생성할 수 있습니다.

--overwrite 옵션을 사용하면 **perf** 레코드 가 모든 데이터를 덮어쓸 수 있는 순환 버퍼에 저장합니다. 버퍼가 가득 차면 **perf** 레코드 가 가장 오래된 레코드를 자동으로 덮어쓰므로 **perf.data** 파일에 기록되지 않습니다.

--overwrite 및 **--switch-output-event** 옵션을 함께 사용하면 **--switch-output-event** 트리거 이벤트를 탐지할 때까지 데이터를 기록하고 덤프하는 순환 버퍼를 구성합니다. 사용자에 대한 관심 있는 항목이 발생했으며 순환 버퍼의 데이터를 **perf.data** 파일에 기록하기 위해 **perf** 레코드에 대한 트리거 이벤트 신호입니다. 이렇게 하면 관심 있는 특정 데이터를 수집하지만 **perf.data** 파일에 원하지 않는 데이터를 작성하지 않고 실행 중인 **perf** 프로세스 오버헤드를 동시에 줄일 수 있습니다.

25.2. PERF CIRCULAR BUFFERS를 사용하여 성능 병목 현상을 모니터링하기 위해 특정 데이터를 수집합니다.

perf 도구를 사용하면 관심 있는 데이터만 수집하기 위해 지정하는 이벤트에 의해 트리거되는 순환 버퍼를 생성할 수 있습니다. 이벤트별 데이터를 수집하는 순환 버퍼를 생성하려면 **perf** 에 **--overwrite** 및 **--switch-output-event** 옵션을 사용합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- 프로세스 또는 애플리케이션 내 관심 위치에서 모니터링에 관심이 있는 프로세스 또는 애플리케이션에 **uprobe**를 배치했습니다.

```
# perf probe -x /path/to/executable -a function
Added new event:
  probe_executable:function   (on function in /path/to/executable)
```

```
You can now use it in all perf tools, such as:

        perf record -e probe_executable:function -aR sleep 1
```

절차

- **uprobe**를 트리거 이벤트로 사용하여 원형 버퍼를 생성합니다.

```
# perf record --overwrite -e cycles --switch-output-event probe_executable:function
./executable
[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012231959 ]
[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012232008 ]
^C[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012232082 ]
[ perf record: Captured and wrote 5.621 MB perf.data.<timestamp> ]
```

이 예제에서는 실행 파일을 시작하고 **-e** 옵션 뒤에 지정된 **cpu** 사이클을 수집합니다. **perf** 가 **uprobe**를 탐지할 때까지 **--switch-output-event** 옵션 뒤에 지정된 트리거 이벤트입니다. 이 시점에서 **perf** 는 순환 버퍼에 있는 모든 데이터의 스냅샷을 사용하여 타임스탬프로 식별되는 고유한 **perf.data** 파일에 저장합니다. 이 예제에서는 총 **2**개의 스냅샷을 생성했으며 마지막 **perf.data** 파일은 **Ctrl+c** 를 눌러 강제 실행되었습니다.

26장. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에서 추적 지점 추가 및 제거

제어 파이프 인터페이스를 사용하여 실행 중인 **perf** 수집기에서 다른 추적 포인트를 활성화 및 비활성화함으로써 **perf** 를 중지하거나 재시작하지 않고도 수집하는 데이터를 동적으로 조정할 수 있습니다. 이렇게 하면 중지 또는 다시 시작 프로세스 중에 기록 된 성능 데이터가 손실되지 않습니다.

26.1. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에 추적 지점 추가

제어 파이프 인터페이스를 사용하여 실행 중인 **perf** 수집기에 추적 포인트를 추가하여 **perf** 를 중지하고 성능 데이터를 손실하지 않고도 기록 중인 데이터를 조정합니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.

절차

1. 제어 파이프 인터페이스를 구성합니다.

```
# mkfifo control ack perf.pipe
```

2. 관심 있는 컨트롤 파일 설정 및 이벤트를 사용하여 **perf** 레코드 를 실행합니다.

```
# perf record --control=fifo:control,ack -D -1 --no-buffering -e 'sched:*' -o - > perf.pipe
```

이 예제에서 **-e** 옵션 뒤에 **'sched:*'** 를 선언하면 스케줄러 이벤트가 포함된**f** 레코드 가 시작됩니다.

3. 두 번째 터미널에서 제어 파이프의 읽기 측면을 시작합니다.

```
# cat perf.pipe | perf --no-pager script -i -
```

제어 파이프의 읽기 측면을 시작하면 첫 번째 터미널에서 다음 메시지가 트리거됩니다.

```
Events disabled
```

4. 세 번째 터미널에서 제어 파일을 사용하여 추적 지점을 활성화합니다.

```
# echo 'enable sched:sched_process_fork' > control
```

이 명령은 **perf** 를 트리거하여 선언된 이벤트에 대해 제어 파일의 현재 이벤트 목록을 스캔합니다. 이벤트가 있으면 **tracepoint**가 활성화되어 첫 번째 터미널에 다음 메시지가 표시됩니다.

```
event sched:sched_process_fork enabled
```

추적 지점이 활성화되면 두 번째 터미널은 추적 지점을 감지하는 **perf** 의 출력을 표시합니다.

```
bash 33349 [034] 149587.674295: sched:sched_process_fork: comm=bash pid=33349
child_comm=bash child_pid=34056
```

26.2. PERF를 중지하거나 다시 시작하지 않고 실행 중인 PERF 수집기에서 추적 지점 제거

컨트롤 파이프 인터페이스를 사용하여 실행 중인 **perf** 수집기에서 추적 지점을 제거하여 모음별을 중지하고 성능 데이터를 손실하지 않고도 수집 중인 데이터 범위를 줄입니다.

사전 요구 사항

- **perf** 설치에 설명된 대로 **perf** 사용자 공간 도구가 설치되어 있습니다.
- 제어 파이프 인터페이스를 통해 실행 중인 **perf** 수집기에 추적 포인트를 추가했습니다. 자세한 내용은 **perf**를 중지하거나 다시 시작하지 않고 실행 중인 **perf** 수집기에 추적 지점 추가를 참조하십시오.

절차

- 추적 지점을 제거합니다.

```
# echo 'disable sched:sched_process_fork' > control
```

참고

이 예제에서는 이전에 스케줄러 이벤트를 제어 파일에 로드했으며 **tracepoint sched_process_fork**를 활성화했다고 가정합니다.

*이 명령은 **perf** 를 트리거하여 선언된 이벤트에 대해 제어 파일의 현재 이벤트 목록을 스캔합니다. 이벤트가 있는 경우 추적 지점이 비활성화되어 있으며 컨트롤 파이프를 구성하는 데 사용되는 터미널에 다음 메시지가 표시됩니다.*

```
event sched:sched_process_fork disabled
```

27장. NUMASTAT로 메모리 할당 프로파일링

numastat 툴을 사용하면 시스템의 메모리 할당에 대한 통계를 표시할 수 있습니다.

numastat 툴은 각 **NUMA** 노드의 데이터를 별도로 표시합니다. 이 정보를 사용하여 시스템의 메모리 성능 또는 시스템상의 다양한 메모리 정책의 효과를 조사할 수 있습니다.

27.1. 기본 NUMASTAT 통계

기본적으로 **numastat** 툴은 각 **NUMA** 노드의 이러한 데이터 카테고리에 대한 통계를 표시합니다.

numa_hit

이 노드에 성공적으로 할당된 페이지 수입니다.

numa_miss

예상 노드의 메모리가 부족하기 때문에 이 노드에 할당된 페이지 수입니다. 각 **numa_miss** 이벤트에는 다른 노드에 해당 **numa_foreign** 이벤트가 있습니다.

numa_foreign

이 노드의 원래는 다른 노드에 할당된 페이지 수입니다. 각 **numa_foreign** 이벤트에는 다른 노드에서 해당 **numa_miss** 이벤트가 있습니다.

interleave_hit

이 노드에 성공적으로 할당된 중간 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.

참고

numa_hit 값과 낮은 **numa_miss** 값(**other**과 동일)은 최적의 성능을 나타냅니다.

27.2. NUMASTAT로 메모리 할당 보기

numastat 툴을 사용하여 시스템의 메모리 할당을 볼 수 있습니다.

사전 요구 사항

- **numactl** 패키지를 설치합니다.

```
# dnf install numactl
```

절차

- 시스템의 메모리 할당을 확인합니다.

```
$ numastat
                           node0           node1
numa_hit                76557759        92126519
numa_miss               30772308        30827638
numa_foreign            30827638        30772308
interleave_hit            106507          103832
local_node              76502227        92086995
other_node              30827840        30867162
```

추가 리소스

- **numastat(8)** 도움말 페이지

28장. CPU 사용률을 최적화하도록 운영 체제 구성

이 섹션에서는 워크로드 전반에서 CPU 사용률을 최적화하도록 운영 체제를 구성하는 방법을 설명합니다.

28.1. 프로세서 문제를 모니터링 및 진단하기 위한 툴

다음은 Red Hat Enterprise Linux 9에서 프로세서 관련 성능 문제를 모니터링하고 진단할 수 있는 툴입니다.

- turbostat 툴은 관리자가 과도한 전원 사용과 같은 서버에서 예기치 않은 동작을 식별할 수 있도록 카운터 결과를 출력하고, 과도한 절전 상태를 입력하지 못하거나 시스템 관리 인터럽트(SMI)가 불필요하게 생성됩니다.
- numactl 유틸리티는 프로세서 및 메모리 선호도를 관리하는 다양한 옵션을 제공합니다. numactl 패키지에는 커널에서 지원하는 NUMA 정책에 간단한 프로그래밍 인터페이스를 제공하는 libnuma 라이브러리가 포함되어 있으며 numactl 애플리케이션보다 더 세분화된 튜닝에 사용할 수 있습니다.
- numastat 툴은 운영 체제 및 프로세스의 각 NUMAA 노드 메모리 통계를 표시하고, 프로세스 메모리가 시스템 전체에 분산되어 있는지 또는 특정 노드에 중앙 집중화되는지 관리자에게 표시합니다. 이 툴은 numactl 패키지에서 제공합니다.
- numad 는 자동 NUMA 선호도 관리 데몬입니다. NUMA 리소스 할당 및 관리를 동적으로 개선하기 위해 시스템 내에서 NUMA 토폴로지 및 리소스 사용량을 모니터링합니다.
- /proc/interrupts 파일은 인터럽트 요청(IRQ) 번호, 시스템의 각 프로세서가 처리하는 유사한 인터럽트 요청 수, 전송된 인터럽트 유형, 나열된 인터럽트 요청에 응답하는 쉼표로 구분된 장치 목록을 표시합니다.
- pqos 유틸리티는 intel-cmt-cat 패키지에서 사용할 수 있습니다. 최근 Intel 프로세서에서 CPU 캐시 및 메모리 대역폭을 모니터링합니다. 다음을 모니터링합니다.
  - 언어별 지침(IPC)입니다.
  - 마지막 수준 캐시 MiSSES의 수입니다.

- 지정된 CPU에서 프로그램이 실행되는 킬로바이트 크기(kilobytes)입니다.
- 로컬 메모리(MBL)의 대역폭입니다.
- 원격 메모리(MBR)의 대역폭입니다.

- x86_energy_perf_policy 툴을 통해 관리자는 성능 및 에너지 효율성의 상대적 중요성을 정의할 수 있습니다. 그런 다음 이 정보를 사용하여 성능과 전력 효율성 사이에서 거래되는 옵션을 선택할 때 이러한 기능을 지원하는 프로세서에 영향을 미칠 수 있습니다.

- taskset 툴은 util-linux 패키지에서 제공합니다. 관리자는 이를 통해 실행 중인 프로세스의 프로세서 선호도를 검색하고 설정하거나 지정된 프로세서 선호도로 프로세스를 시작할 수 있습니다.

추가 리소스

- turbostat(8), numactl(8), numastat(8), numa(7), numad(8), pqos(8), x86_energy_perf_policy(8) 및 taskset(1) 매뉴얼 페이지

28.2. 시스템 토폴로지 유형

최신 컴퓨팅에서는 대부분의 최신 시스템에 여러 프로세서가 있으므로 CPU에 대한 아이디어가 오를 수 있습니다. 시스템의 토폴로지는 이러한 프로세서가 서로 연결되고 다른 시스템 리소스에 연결하는 방식입니다. 이는 시스템 및 애플리케이션 성능과 시스템 튜닝 고려 사항에 영향을 미칠 수 있습니다.

다음은 최신 컴퓨팅에 사용되는 두 가지 기본 토폴로지 유형입니다.

대칭 MP(Multi-Processor) 토폴로지

NetNamespace 토폴로지를 사용하면 모든 프로세서가 동일한 시간 내에 메모리에 액세스할 수 있습니다. 그러나 공유 및 동일한 메모리 액세스가 본질적으로 모든 CPU에서 직렬화된 메모리 액세스를 강제 적용하기 때문에 ETL 시스템 확장 제약 조건이 일반적으로 예기치 않은 것으로 표시됩니다. 이러한 이유로 사실상 모든 최신 서버 시스템은 NUMA 시스템입니다.

NUMA(Non-Uniform Memory Access) 토폴로지

NUMA 토폴로지는 weekly 토폴로지보다 최근에 개발되었습니다. NUMA 시스템에서는 여러 개의 프로세서가 소켓에 물리적으로 그룹화됩니다. 각 소켓에는 해당 메모리에 대한 로컬 액세스 권한이 있는 전용 메모리 및 프로세서 영역이 있으며 이러한 영역은 노드라고 합니다. 동일한 노드의 프로세서

는 해당 노드의 메모리 뱅크에 빠르게 액세스할 수 있으며 노드에 없는 메모리 은행에 대한 액세스 속도가 느려집니다.

따라서 로컬이 아닌 메모리에 액세스할 때 성능이 저하됩니다. 따라서 **NUMA** 토폴로지가 있는 시스템에서 성능에 민감한 애플리케이션은 애플리케이션을 실행하는 프로세서와 동일한 노드에 있는 메모리에 액세스해야 하며 가능한 경우 원격 메모리에 액세스하지 않아야 합니다.

성능에 민감한 다중 스레드 애플리케이션은 특정 프로세서가 아닌 특정 **NUMA** 노드에서 실행되도록 구성된 이점을 얻을 수 있습니다. 이것이 적합한지 여부는 시스템 및 애플리케이션의 요구 사항에 따라 달라집니다. 여러 애플리케이션 스레드가 동일한 캐시된 데이터에 액세스하는 경우 동일한 프로세서에서 실행되도록 해당 스레드를 구성하는 것이 적합할 수 있습니다. 그러나 서로 다른 데이터를 액세스 및 캐시하는 여러 스레드가 동일한 프로세서에서 실행되는 경우 각 스레드는 이전 스레드에서 액세스한 캐시된 데이터를 제거할 수 있습니다.**However, if multiple threads that access and cache different data execute on the same processor, each thread may remove cached data accessed by a previous thread.** 즉, 각 스레드가 캐시를 허용하여 메모리에서 데이터를 가져와 캐시에서 교체하는 실행 시간을 소비합니다. **perf** 툴을 사용하여 과도한 캐시 누락 수를 확인합니다.

28.2.1. 시스템 토폴로지 표시

시스템의 토폴로지를 이해하는 데 도움이 되는 여러 명령이 있습니다. 다음 절차에서는 시스템 토폴로지를 결정하는 방법을 설명합니다.

절차

- 시스템 토폴로지 개요를 표시하려면 다음을 수행합니다.

```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
[...]
```

- **CPU** 아키텍처 정보(예: **CPU**, 스레드, 코어, 소켓, **NUMA** 노드 수)를 수집하려면 다음을 수행합니다.

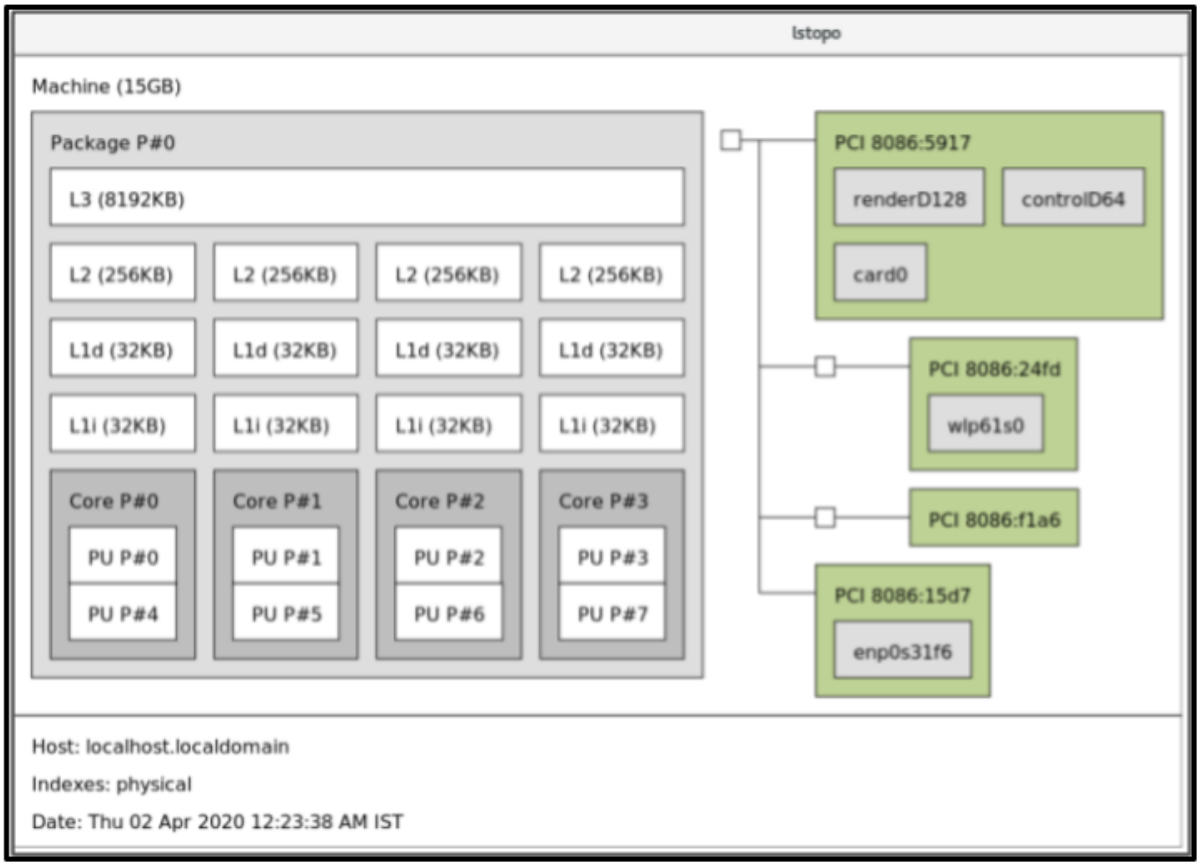
```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    10
```

```
Socket(s):          4
NUMA node(s):       4
Vendor ID:          GenuineIntel
CPU family:         6
Model:              47
Model name:         Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:           2
CPU MHz:            2394.204
BogoMIPS:           4787.85
Virtualization:     VT-x
L1d cache:          32K
L1i cache:          32K
L2 cache:           256K
L3 cache:           30720K
NUMA node0 CPU(s):  0,4,8,12,16,20,24,28,32,36
NUMA node1 CPU(s):  2,6,10,14,18,22,26,30,34,38
NUMA node2 CPU(s):  1,5,9,13,17,21,25,29,33,37
NUMA node3 CPU(s):  3,7,11,15,19,23,27,31,35,39
```

- *시스템의 그래픽 표시를 보려면 다음을 수행합니다.*

```
# dnf install hwloc-gui
# lstopo
```

*그림 **28.1. lstopo** 출력*



- *자세한 **symbols** 출력을 보려면 다음을 수행합니다.*

```
# dnf install hwloc
# lstopo-no-graphics
Machine (15GB)
  Package L#0 + L3 L#0 (8192KB)
    L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
      PU L#0 (P#0)
      PU L#1 (P#4)
     HostBridge L#0
    PCI 8086:5917
      GPU L#0 "renderD128"
      GPU L#1 "controlD64"
      GPU L#2 "card0"
    PCIBridge
      PCI 8086:24fd
        Net L#3 "wlp61s0"
    PCIBridge
      PCI 8086:f1a6
    PCI 8086:15d7
      Net L#4 "enp0s31f6"
```

추가 리소스

- **numactl(8), lscpu(1), lstopo(1)** 매뉴얼 페이지

28.3. 커널 틱 시간 구성

기본적으로 **Red Hat Enterprise Linux 9**는 유휴 **CPU**를 중단하지 않는 틱리스 커널을 사용하여 전력 사용량을 줄이고 새 프로세서가 수면 상태를 활용할 수 있도록 합니다.

Red Hat Enterprise Linux 9는 또한 고성능 컴퓨팅 또는 실시간 컴퓨팅과 같은 대기 시간에 민감한 워크로드에 유용한 동적 틱리스 옵션을 제공합니다. 기본적으로 동적 틱리스 옵션은 비활성화되어 있습니다. **cpu-partitioning TuneD** 프로필을 사용하여 **isolated_cores** 로 지정된 코어의 동적 틱리스 옵션을 활성화하는 것이 좋습니다.

이 절차에서는 동적 틱리스 동작을 수동으로 활성화하는 방법을 설명합니다.

절차

1. 특정 코어에서 동적 틱리스 동작을 활성화하려면 **nohz_full** 매개변수를 사용하여 커널 명령줄에 코어를 지정합니다. **16** 코어 시스템의 경우 **/etc/default/grub** 파일의 **GRUB_CMDLINE_LINUX** 옵션에 이 매개 변수를 추가합니다.

```
nohz_full=1-15
```

*이를 통해 코어 **1** 에서 **15** 까지 동적 틱 수없는 동작을 가능하게하고 모든 시간 유지를 지정하지 않은 코어 (코어 **0**)로 이동합니다.*

2. *동적 틱리스 동작을 영구적으로 활성화하려면 편집한 기본 파일을 사용하여 **GRUB2** 설정을 다시 생성합니다. **BIOS** 펌웨어가 있는 시스템에서 다음 명령을 실행합니다.*

```
# grub2-mkconfig -o /etc/grub2.cfg
```

__UEFI__ 펌웨어가 있는 시스템에서 다음 명령을 실행합니다.

```
# grub2-mkconfig -o /etc/grub2-efi.cfg
```

3. *시스템이 부팅될 때 **rcu** 스레드를 대기 시간이 아닌 코어로 수동으로 이동합니다. 이 경우 코어 **0**:*

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

4. *선택 사항: 커널 명령줄에서 **isolcpus** 매개변수를 사용하여 특정 코어를 사용자 공간 작업으로부터 분리합니다.*

5. *선택 사항: 커널의 **write-back bdi-flush** 스레드의 **CPU** 선호도를 하우스키핑 코어로 설정합니다.*

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

검증 단계

- *시스템이 재부팅되면 **dynticks** 가 활성화되었는지 확인합니다.*

```
# journalctl -xe | grep dynticks
Mar 15 18:34:54 rhel-server kernel: NO_HZ: Full dynticks CPUs: 1-15.
```

- *동적 틱 없는 구성이 제대로 작동하는지 확인합니다.*

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
```

*이 명령은 **CPU 1**을 유휴 상태로 지정하는 동안 **CPU 1**이 **3**초 동안 켜지도록 하는 동안 **CPU**의*

틱을 측정합니다.

- 기본 커널 타이머 설정은 일반 **CPU**에서 약 **3100**개의 틱을 보여줍니다.

```
# perf stat -C 0 -e irq_vectors:local_timer_entry taskset -c 0 sleep 3

Performance counter stats for 'CPU(s) 0':

             3,107      irq_vectors:local_timer_entry

       3.001342790 seconds time elapsed
```

- 동적 틱리스 커널이 구성된 경우, 약 **4**개의 틱을 볼 수 있습니다.

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3

Performance counter stats for 'CPU(s) 1':

                 4      irq_vectors:local_timer_entry

       3.001544078 seconds time elapsed
```

추가 리소스

- **perf(1)** 및 **cpuset442)** 매뉴얼 페이지
- **nohz_full** 커널 매개변수 **Red Hat Knowledgebase** 문서
- **sysfs**의 **"isolated"** 및 **"nohz_full" CPU** 정보를 확인하는 방법은 무엇입니까**? Red Hat Knowledgebase** 문서

28.4. 인터럽트 요청 개요

인터럽트 요청 또는 **IRQ**는 하드웨어에서 프로세서로 전송되는 즉각적인 주의를 위한 신호입니다. 시스템의 각 장치에는 고유한 인터럽트를 보낼 수 있는 **IRQ** 번호가 하나 이상 할당됩니다. 인터럽트가 활성화되면 인터럽트 요청을 수신하는 프로세서는 인터럽트 요청을 처리하기 위해 현재 애플리케이션 스레드의 실행을 즉시 일시 중지합니다.

인터럽트가 정상적인 작동을 중단하므로 인터럽트 비율이 심각하게 시스템 성능이 저하될 수 있습니다. 인터럽트 선호도를 설정하거나 일괄 처리에서 여러 우선 순위 인터럽트를 전송하여 인터럽트에서 걸

리는 시간을 줄일 수 있습니다(마운트 인터럽트 수 참조).

인터럽트 요청에는 인터럽트 요청을 처리하는 프로세서를 정의하는 **smp_affinity** 속성 **smp_affinity**이 있습니다. 애플리케이션 성능을 개선하기 위해 인터럽트 선호도 및 프로세스 선호도를 동일한 프로세서 또는 동일한 코어의 프로세서에 할당합니다. 이렇게 하면 지정된 인터럽트 및 애플리케이션 스레드가 캐시 라인을 공유할 수 있습니다.

인터럽트 **steering**을 지원하는 시스템에서는 인터럽트 요청의 **smp_affinity** 속성을 수정하여 하드웨어가 설정되므로 특정 프로세서를 통한 중단을 위한 결정이 커널에서 개입하지 않고 하드웨어 수준에서 이루어집니다.

28.4.1. 인터럽트 수동 밸런싱

BIOS가 **NUMA** 토폴로지를 내보내는 경우 **irqbalance** 서비스는 하드웨어 요청 서비스에 로컬인 노드의 인터럽트 요청을 자동으로 제공할 수 있습니다.

절차

1. 구성할 인터럽트 요청에 해당하는 장치를 확인합니다.

2. 플랫폼의 하드웨어 사양을 찾으십시오. 시스템의 칩셋이 인터럽트 배포를 지원하는지 확인합니다.

   a. 이 경우 다음 단계에 설명된 대로 인터럽트 전달을 구성할 수 있습니다. 또한 칩셋에서 인터럽트의 균형을 조정하는 데 사용하는 알고리즘을 확인합니다. 일부 **BIOS**에는 인터럽트 전달을 구성하는 옵션이 있습니다.

   b. 그렇지 않은 경우 칩셋은 모든 인터럽트를 항상 하나의 정적 **CPU**로 라우팅합니다. 사용되는 **CPU**를 구성할 수 없습니다.

3. 시스템에서 사용 중인 **APIC(Advanced Programmable Interrupt Controller)** 모드를 확인합니다.

```
$ journalctl --dmesg | grep APIC
```

   여기,

- *시스템에서 플랫 이외의 모드를 사용하는 경우 물리적 플랫 으로 **APIC** 라우팅 설정 과 유사한 행이 표시될 수 있습니다.*

- *이러한 메시지가 표시되지 않으면 시스템은 플랫 모드를 사용합니다.*

*시스템이 **x2apic** 모드를 사용하는 경우 부트 로더 설정의 커널 명령줄에 **nox2apic** 옵션을 추가하여 이를 비활성화할 수 있습니다.*

*플랫(물리적 플랫 모드)만 인터럽트를 여러 **CPU**에 배포할 수 있도록 지원합니다. 이 모드는 **CPU**가 **8** 개까지 있는 시스템에만 사용할 수 있습니다.*

4. *__smp_affinity mask__ 를 계산합니다. __smp_affinity mask__ 를 계산하는 방법에 대한 자세한 내용은 __Setting the smp_affinity mask__ 를 참조하십시오.*

추가 리소스

- ***journalctl(1)*** *및* ***taskset(1)*** *도움말 페이지*

28.4.2. smp_affinity mask 설정

smp_affinity *값은 시스템의 모든 프로세서를 나타내는* ***16***진수 *비트 마스크로 저장됩니다. 각 비트는 다른* ***CPU***를 *구성합니다. 가장 작은 비트는* ***CPU 0***입니다.

mask의 *기본값은* ***f*** *이며, 이는 시스템의 모든 프로세서에서 인터럽트 요청을 처리할 수 있음을 의미합니다. 이 값을* ***1***로 *설정하면 프로세서* ***0***만 *인터럽트를 처리할 수 있습니다.*

절차

1. *바이너리에서 인터럽트를 처리하는* ***CPU***의 *값* ***1***을 *사용합니다. 예를 들어 인터럽트를 처리하기 위해* ***CPU 0*** *및* ***CPU 7***을 *설정하려면* ***0000000010000001*** *을 바이너리 코드로 사용합니다.*

표 ***28.1. CPU***용 *바이너리 비트*

CPU	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
바이너리	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

2. 바이너리 코드를 16진수로 변환합니다.

   예를 들어 Python을 사용하여 바이너리 코드를 변환하려면 다음을 수행합니다.

```
>>> hex(int('0000000010000001', 2))
'0x81'
```

   32개 이상의 프로세서가 있는 시스템에서는 개별 32비트 그룹의 smp_affinity 값을 줄여야 합니다. 예를 들어 64 프로세서 시스템의 첫 번째 32 프로세서만 인터럽트 요청을 서비스하려면 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffff,00000000 을 사용합니다.

3. 특정 인터럽트 요청의 인터럽트 선호도 값은 연결된 /proc/irq/irq_number/smp_affinity 파일에 저장됩니다. 이 파일에서 smp_affinity 마스크를 설정합니다.

```
# echo mask > /proc/irq/irq_number/smp_affinity
```

추가 리소스

- journalctl(1), irqbalance(1) 및 taskset(1) 매뉴얼 페이지

29장. 스케줄링 정책 튜닝

Red Hat Enterprise Linux에서 가장 작은 프로세스 실행 단위를 스레드라고 합니다. 시스템 스케줄러는 스레드를 실행하는 프로세서와 스레드가 실행되는 기간을 결정합니다. 그러나 스케줄러의 주요 문제는 시스템을 계속 사용하는 것이기 때문에 애플리케이션 성능에 가장 적합한 스레드를 예약하지 않을 수 있습니다.

예를 들어 NUMA 시스템의 애플리케이션이 노드 B의 프로세서를 사용할 수 있게 되면 노드 A에서 실행되고 있다고 가정합니다. 노드 B에서 프로세서를 사용하도록 유지하기 위해 스케줄러는 애플리케이션의 스레드 중 하나를 Node B로 이동합니다. 그러나 애플리케이션 스레드는 여전히 Node A의 메모리에 액세스해야 합니다. 그러나 스레드가 현재 노드 B에서 실행되고 있는 스레드가 더 이상 스레드에서 실행되고 있지 않기 때문에 이 메모리는 더 이상 액세스할 수 없습니다. 따라서 노드 A에서 프로세서를 사용할 수 있을 때까지 대기한 다음 로컬 메모리 액세스 권한이 있는 원래 노드에서 스레드를 실행하는 데 걸리는 것보다 노드 B에서 실행을 완료하는 데 시간이 더 오래 걸릴 수 있습니다.

29.1. 스케줄링 정책의 범주

성능에 민감한 애플리케이션은 종종 설계자 또는 관리자가 스레드가 실행되는 위치를 결정하는 데 도움이 됩니다. Linux 스케줄러는 스레드가 실행되는 위치와 기간을 결정하는 여러 스케줄링 정책을 구현합니다.

다음은 스케줄링 정책의 두 가지 주요 범주입니다.

일반 정책

일반 스레드는 일반적인 우선 순위의 작업에 사용됩니다.

실시간 정책

실시간 정책은 중단 없이 완료해야 하는 시간에 민감한 작업에 사용됩니다. 실시간 스레드는 시간 분할의 영향을 받지 않습니다. 즉, 스레드는 차단, 종료, 자발적으로 산출되거나 더 높은 우선순위 스레드에 의해 선점될 때까지 실행됩니다.

가장 낮은 우선 순위 실시간 스레드는 일반 정책이 있는 스레드보다 먼저 예약됩니다. 자세한 내용은 nfsnobody _RR 를 사용한 Static 우선순위 스케줄링 을 참조하십시오.

추가 리소스

- sched -02- , sched_setaffinity(2), sched_getaffinity(2), sched_setscheduler(2) 및 sched_getscheduler(2) 도움말 페이지

29.2. NFSNOBODY_DPDK를 사용한 정적 우선 순위 예약

static 우선순위 예약이라고도 하는 NetNamespace _ databind 는 각 스레드에 대해 고정된 우선 순위를 정의하는 실시간 정책입니다. 관리자는 이 정책을 통해 이벤트 응답 시간을 개선하고 대기 시간을 줄일 수 있습니다. 시간에 민감한 작업의 장기적인 기간 동안 이 정책을 실행하지 않는 것이 좋습니다.

NetNamespace _ ECDSA가 사용 중인 경우 스케줄러는 우선 순위로 모든 DASD _ ECDSA 스레드 목록을 스캔하고 실행할 준비가 된 가장 높은 우선 순위 스레드를 예약합니다. jenkinsfile _ databind 스레드의 우선순위 수준은 1 에서 99까지의 모든 정수일 수 있으며 99 는 가장 높은 우선 순위로 취급됩니다. Red Hat은 대기 시간 문제를 식별할 때만 더 낮은 번호로 시작하고 우선 순위를 높이는 것이 좋습니다.

주의

실시간 스레드는 시간 분할에 영향을 받지 않으므로 Red Hat은 우선 순위를 99로 설정하는 것을 권장하지 않습니다. 이렇게 하면 프로세스가 마이그레이션 및 워치독 스레드와 동일한 우선 순위 수준으로 유지됩니다. 스레드가 계산 루프에 들어가면 이러한 스레드가 차단되면 실행할 수 없습니다. 단일 프로세서가 있는 시스템은 결국 이러한 상황에서 중단될 것입니다.

관리자는 realtime 애플리케이션 프로그래머가 프로세서를 모국화하는 실시간 작업을 시작하는 것을 방지하기 위해 NetNamespace_ ptp 대역폭을 제한할 수 있습니다.

다음은 이 정책에서 사용되는 몇 가지 매개 변수입니다.

/proc/sys/kernel/sched_rt_period_us

이 매개 변수는 프로세서 대역폭의 100 %로 간주되는 시간 기간을 마이크로초 단위로 정의합니다. 기본값은 1000000 databinds 또는 1초입니다.

/proc/sys/kernel/sched_rt_runtime_us

이 매개 변수는 실시간 스레드를 실행하는 데 사용되는 시간 기간을 마이크로초 단위로 정의합니다. 기본값은 95% 0000 databinds 또는 0.95초입니다.

29.3. DASD_RR을 사용한 라운드 로빈 우선순위 스케줄링

Net Namespace_RR 은 cryptsetup _databind 의 라운드 로빈 변형입니다. 이 정책은 여러 스레드를

동일한 우선 순위 수준에서 실행해야 하는 경우에 유용합니다.

Net Namespace_databind 와마찬가지로, **cryptsetup_RR** 은 각 스레드에 대해 고정된 우선 순위를 정의하는 실시간 정책입니다. 스케줄러는 우선 순위에 따라 모든 **DASD_RR** 스레드 목록을 스캔하고 실행할 준비가 된 가장 높은 우선 순위 스레드를 예약합니다. 그러나 **precedence** 가 동일한 우선 순위가 같은 스레드는 특정 시간 슬라이스 내에서 라운드 로빈 스타일로 예약됩니다.

/proc/sys/kernel/sched_rr_timeslice_ms 파일에서 **sched_rr_timeslice_ms** 커널 매개 변수를 사용하여 이 시간 슬라이스의 값을 밀리초 단위로 설정할 수 있습니다. 가장 작은 값은 **1**밀리초 입니다.

29.4. NFSNOBODY_OTHER를 사용한 일반 스케줄링

Net Namespace_OTHER 는 **Red Hat Enterprise Linux 9**의 기본 스케줄링 정책입니다. 이 정책은 **CFS(Completely Fair Scheduler)**를 사용하여 이 정책으로 예약된 모든 스레드에 대한 공정한 프로세서 액세스를 허용합니다. 이 정책은 시간이 지남에 따라 스레드를 보다 효율적으로 예약할 수 있으므로 많은 수의 스레드가 있거나 데이터 처리량이 우선 순위인 경우 가장 유용합니다.

이 정책이 사용 중인 경우 스케줄러는 각 프로세스 스레드의 선호도 값에 따라 부분적으로 동적 우선 순위 목록을 생성합니다. 관리자는 프로세스의 **niceness** 값을 변경할 수 있지만 스케줄러의 동적 우선 순위 목록을 직접 변경할 수는 없습니다.

29.5. 스케줄러 정책 설정

chrt 명령줄 툴을 사용하여 스케줄러 정책 및 우선 순위를 확인하고 조정합니다. 원하는 속성을 사용하여 새 프로세스를 시작하거나 실행 중인 프로세스의 속성을 변경할 수 있습니다. 런타임 시 정책을 설정하는 데 사용할 수도 있습니다.

절차

1. 활성 프로세스의 **PID(프로세스 ID)**를 확인합니다.

```
# ps
```

ps 명령과 함께 **--pid** 또는 **-p** 옵션을 사용하여 특정 **PID**의 세부 정보를 확인합니다.

2. 특정 프로세스의 스케줄링 정책, **PID** 및 우선 순위를 확인합니다.

```
# chrt -p 468
pid 468's current scheduling policy: SCHED_FIFO
pid 468's current scheduling priority: 85

# chrt -p 476
pid 476's current scheduling policy: SCHED_OTHER
pid 476's current scheduling priority: 0
```

*여기서 **468** 및 **476** 은 프로세스의 **PID**입니다.*

3. *프로세스의 스케줄링 정책을 설정합니다.*

   a. *예를 들어 **PID 1000** 인 프로세스를 우선 순위 **50** 개로 설정하려면 다음을 수행합니다.*

   ```
   # chrt -f -p 50 1000
   ```

   b. *예를 들어 **PID 1000** 이 있는 프로세스를 **priority _OTHER** 로 설정하고 우선 순위가 **0** 으로 설정하려면 다음을 실행합니다.*

   ```
   # chrt -o -p 0 1000
   ```

   c. *예를 들어 **PID 1000** 인 프로세스를 **priority _RR** 로 설정하고 우선 순위 **10:***

   ```
   # chrt -r -p 10 1000
   ```

   d. *특정 정책 및 우선 순위로 새 애플리케이션을 시작하려면 애플리케이션 이름을 지정합니다.*

   ```
   # chrt -f 36 /bin/my-app
   ```

추가 리소스

- ***chrt(1)** 도움말 페이지*
- ***chrt** 명령의 정책 옵션*

- *부팅 프로세스 중 서비스 우선 순위 변경*

29.6. CHRT 명령의 정책 옵션

chrt 명령을 사용하면 프로세스의 스케줄링 정책을 보고 설정할 수 있습니다.

다음 표에서는 프로세스의 스케줄링 정책을 설정하는 데 사용할 수 있는 적절한 정책 옵션을 설명합니다.

표 29.1. chrt 명령의 정책 옵션

짧은 옵션	긴 옵션	설명
-f	**--fifo**	schedule을 nfsnobody **_databind**로 설정
-o	**--other**	schedule을 NetNamespace **_OTHER**로 설정
-r	**--rr**	schedule을 DASD **_RR**로 설정

29.7. 부팅 프로세스 중 서비스 우선 순위 변경

systemd 서비스를 사용하면 부팅 프로세스 중에 시작된 서비스에 대한 실시간 우선 순위를 설정할 수 있습니다. 단위 구성 지시문 은 부팅 프로세스 중에 서비스의 우선 순위를 변경하는 데 사용됩니다.

부팅 프로세스 우선 순위 변경은 service 섹션에서 다음 지시문을 사용하여 수행됩니다.

CPUSchedulingPolicy=

실행된 프로세스에 대한 CPU 스케줄링 정책을 설정합니다. 다른,fifo 및 rr 정책을 설정하는 데 사용됩니다.

CPUSchedulingPriority=

실행된 프로세스의 CPU 스케줄링 우선 순위를 설정합니다. 사용 가능한 우선 순위 범위는 선택한 CPU 스케줄링 정책에 따라 다릅니다. 실시간 스케줄링 정책의 경우 1 (최저 우선 순위)과 99 (최고 우선 순위) 사이의 정수를 사용할 수 있습니다.

다음 절차에서는 mcelog 서비스를 사용하여 부팅 프로세스 중에 서비스의 우선 순위를 변경하는 방법

을 설명합니다.

사전 요구 사항

1. ***TuneD** 패키지를 설치합니다.*

```
# dnf install tuned
```

2. ***TuneD** 서비스를 활성화하고 시작합니다.*

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

절차

1. *실행 중인 스레드의 스케줄링 우선 순위를 확인합니다.*

```
# tuna --show_threads
                    thread       ctxt_switches
   pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
  1     OTHER    0    0xff      3181          292         systemd
  2     OTHER    0    0xff       254            0        kthreadd
  3     OTHER    0    0xff         2            0         rcu_gp
  4     OTHER    0    0xff         2            0      rcu_par_gp
  6     OTHER    0       0         9            0 kworker/0:0H-kblockd
  7     OTHER    0    0xff      1301            1 kworker/u16:0-events_unbound
  8     OTHER    0    0xff         2            0    mm_percpu_wq
  9     OTHER    0       0       266            0     ksoftirqd/0
[...]
```

2. *보조 **mcelog** 서비스 구성 디렉터리 파일을 생성하고 이 파일에 정책 이름과 우선 순위를 삽입합니다.*

```
# cat <<-EOF > /etc/systemd/system/mcelog.system.d/priority.conf

>
[SERVICE]
CPUSchedulingPolicy=_fifo_
CPUSchedulingPriority=_20_
EOF
```

3. ***systemd** 스크립트 구성을 다시 로드합니다.*

```
# systemctl daemon-reload
```

-

4. **mcelog 서비스를 다시 시작하십시오.**

```
# systemctl restart mcelog
```

검증 단계

- **systemd 문제로 설정된 mcelog 우선 순위를 표시합니다.**

```
# tuna -t mcelog -P
thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
826     FIFO    20  0,1,2,3        13             0          mcelog
```

추가 리소스

- **systemd(1) 및 tuna(8) 매뉴얼 페이지**
- *우선순위 범위에 대한 설명*

29.8. 우선순위 맵

우선순위는 특정 커널 기능 전용인 그룹으로 정의됩니다. 실시간 스케줄링 정책의 경우 1 (최저 우선 순위)과 99 (최고 우선 순위) 사이의 정수를 사용할 수 있습니다.

다음 표에서는 프로세스의 스케줄링 정책을 설정하는 동안 사용할 수 있는 우선순위 범위를 설명합니다.

표 29.2. 우선순위 범위에 대한 설명

우선 순위	스레드	설명
1	낮은 우선 순위 커널 스레드	일반적으로 이 우선순위는 deadline _OTHER 위에 있어야 하는 작업에 대해 예약되어 있습니다.
2 - 49	사용 가능	일반적인 애플리케이션 우선 순위에 사용되는 범위입니다.
50	기본 hard-IRQ 값	

우선 순위	스레드	설명
51 - 98	높은 우선 순위 스레드	주기적으로 실행되고 빠른 응답 시간이 있어야 하는 스레드에 이 범위를 사용합니다. 인터럽트를 시작할 때 CPU 바인딩 스레드에 이 범위를 사용하지 마십시오.
99	워치독 및 마이그레이션	가장 높은 우선순위로 실행해야 하는 시스템 스레드입니다.

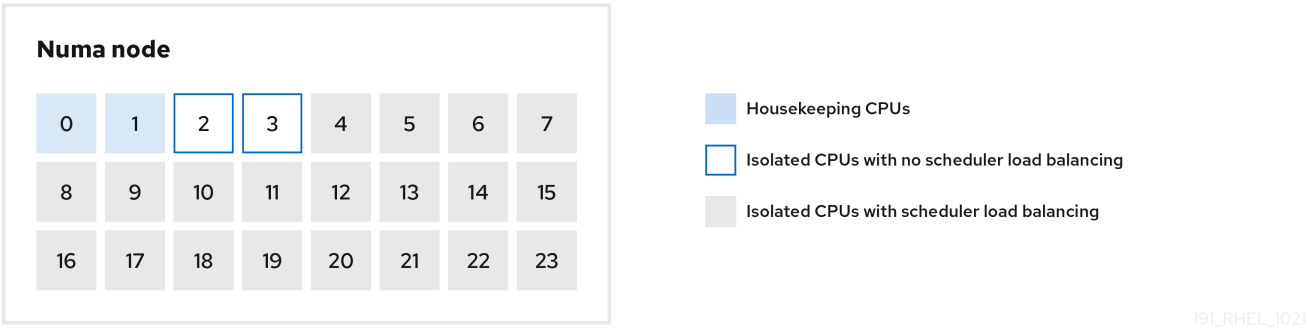
29.9. TUNED CPU-PARTITIONING 프로필

*대기 시간에 민감한 워크로드에 맞게 **Red Hat Enterprise Linux 9**를 튜닝하려면 **cpu-partitioning TuneD** 프로필을 사용하는 것이 좋습니다.*

***Red Hat Enterprise Linux 9** 이전에는 대기 시간이 짧은 **Red Hat** 문서에서는 대기 시간이 짧은 튜닝을 수행하는 데 필요한 수많은 낮은 수준의 단계를 설명했습니다. **Red Hat Enterprise Linux 9**에서는 **cpu-partitioning TuneD** 프로필을 사용하여 대기 시간이 짧은 튜닝을 보다 효율적으로 수행할 수 있습니다. 이 프로필은 대기 시간이 짧은 개별 애플리케이션의 요구 사항에 따라 쉽게 사용자 지정할 수 있습니다.*

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

*그림 **29.1. figure cpu-partitioning***



*다음 구성 옵션을 사용하여 **/etc/tuned/cpu-partitioning-ECDHEs.conf** 파일에서 **cpu-partitioning** 프로필을 구성할 수 있습니다.*

*로드 밸런싱이 있는 격리된 **CPU***

***cpu-partitioning figure**에서 **4**에서 **23**으로 번호가 매겨진 블록은 기본 분리된 **CPU**입니다. 커널*

스케줄러의 프로세스 부하 분산이 이러한 CPU에서 활성화됩니다. 커널 스케줄러 부하 분산이 필요한 여러 스레드가 있는 대기 시간이 짧은 프로세스를 위해 설계되었습니다.

kernel 스케줄러 로드 밸런싱을 사용할 CPU를 나열하는 isolated_cores= cpu-list 옵션을 사용하여 /etc/tuned/cpu-partitions.conf 파일에서 cpu-partitioning 프로필을 구성할 수 있습니다.

분리된 CPU 목록은 쉼표로 구분하거나 dash(예:)를 사용하여 범위를 지정할 수 있습니다. 이 옵션은 필수입니다. 이 목록에서 누락된 CPU는 자동으로 하우스키핑 CPU로 간주됩니다.

로드 밸런싱이 없는 격리된 CPU

cpu-partitioning 그림에서 번호가 지정된 블록 2와 3은 추가 커널 스케줄러 프로세스 부하 분산을 제공하지 않는 격리된 CPU입니다.

커널 스케줄러 로드 밸런싱을 사용하지 않는 CPU를 나열하는 no_balance_cores= cpu-list 옵션을 사용하여 /etc/tuned/cpu-partitions.conf 파일에서 cpu-partitioning 프로필을 구성할 수 있습니다.

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

이러한 CPU를 사용하는 애플리케이션 스레드를 각 CPU에 개별적으로 고정해야 합니다.

하우스키핑 CPU

cpu-partitioning-postgresqls.conf 파일에서 분리된 CPU는 자동으로 하우스키핑 CPU로 간주됩니다. 하우스키핑 CPU에서 모든 서비스, 데몬, 사용자 프로세스, 이동 가능한 커널 스레드, 인터럽트 처리기, 커널 타이머를 실행할 수 있습니다.

추가 리소스

- tuned-profiles-cpu-partitioning(7) man page

29.10. 대기 시간이 짧은 튜닝을 위해 TUNED CPU-PARTITIONING 프로파일 사용

이 프로세스에서는 TuneD의 cpu-partitioning 프로필을 사용하여 대기 시간이 짧은 시스템을 조정하는 방법을 설명합니다. cpu-partitioning 및 cpu-partitioning 수치에 언급된 CPU 레이아웃을 사용할 수 있는 대기 시간이 짧은 애플리케이션의 예를 사용합니다.

이 경우 애플리케이션은 다음을 사용합니다.

- 네트워크에서 데이터를 읽는 하나의 전용 리더 스레드가 **CPU 2**에 고정되어 있습니다.
- 이 네트워크 데이터를 처리하는 많은 수의 스레드가 **CPU 4-23**에 고정되어 있습니다.
- 처리된 데이터를 네트워크에 쓰는 전용 작성기 스레드는 **CPU 3**에 고정되어 있습니다.

사전 요구 사항

- **dnf install tuned-profiles- cpu-partitioning** 명령을 **root**로 사용하여 **cpu-partitioning TuneD** 프로필 을 설치했습니다.

절차

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

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

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

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

3. **reboot**

재부팅 후 **cpu-partitioning figure**의 격리에 따라 대기 시간이 짧아지도록 시스템이 조정됩니다. 애플리케이션은 **taskset**을 사용하여 **reader** 및 **writer** 스레드를 **2** 및 **3**에 고정하고 **CPU 4-23**의 나머지 애플리케이션 스레드를 **CPU 2** 및 **3**에 고정할 수 있습니다.

추가 리소스

- **tuned-profiles-cpu-partitioning(7) man page**

29.11. CPU-PARTITIONING TUNED 프로파일 사용자 정의

TuneD 프로필을 확장하여 추가 튜닝을 변경할 수 있습니다.

예를 들어 cpu-partitioning 프로필은 CPU가 cstate=1 을 사용하도록 설정합니다. cpu-partitioning 프로필을 사용하지만 CPU cstate를 cstate1에서 cstate0으로 추가로 변경하려면 다음 절차에서는 cpu-partitioning 프로필을 상속하고 C state-0을 설정하는 my_profile 이라는 새 TuneD 프로필을 설명합니다.

절차

1. /etc/tuned/my_profile 디렉토리를 만듭니다.

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

2. 이 디렉터리에 tuned.conf 파일을 생성하고 다음 콘텐츠를 추가합니다.

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

3. 새 프로필을 사용합니다.

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

참고

공유 예에서는 재부팅이 필요하지 않습니다. 그러나 my_profile 프로필의 변경 사항을 적용하려면 재부팅을 수행한 다음 시스템을 재부팅합니다.

추가 리소스

- tuned-profiles-cpu-partitioning(7) man page

30장. I/O 및 파일 시스템 성능에 영향을 미치는 요소

스토리지 및 파일 시스템 성능에 적합한 설정은 스토리지 목적에 따라 크게 달라집니다.

I/O 및 파일 시스템 성능은 다음 요인의 영향을 받을 수 있습니다.

- 데이터 쓰기 또는 읽기 패턴
- 순차 또는 임의의
- 버퍼링 또는 직접 IO
- 데이터 기본 기하 도형과 정렬
- 블록 크기
- 파일 시스템 크기
- 저널 크기 및 위치
- 액세스 시간 기록
- 데이터 신뢰성 보장
- 데이터 가져오기 전
- 디스크 공간 사전 할당

- 파일 조각화
- 리소스 경합

30.1. I/O 및 파일 시스템 문제를 모니터링 및 진단하기 위한 툴

Red Hat Enterprise Linux 9에서 시스템 성능을 모니터링하고 I/O, 파일 시스템 및 해당 구성과 관련된 성능 문제를 진단하는 데 사용할 수 있는 툴은 다음과 같습니다.

- vmstat 툴은 전체 시스템의 프로세스, 메모리, 페이징, 블록 I/O, 인터럽트, CPU 활동에 대해 보고합니다. 관리자는 I/O 하위 시스템이 성능 문제를 담당하는지 여부를 결정하는 데 도움이 될 수 있습니다. vmstat 를 사용한 분석을 통해 I/O 하위 시스템이 성능 감소를 담당하는 것으로 표시되면 관리자는 iostat 툴을 사용하여 관련 I/O 장치를 결정할 수 있습니다.
- iostat 는 시스템의 I/O 장치 로드에 대해 보고합니다. NetNamespace 패키지에서 제공합니다.
- blktrace 는 I/O 하위 시스템에서 시간을 소비하는 방법에 대한 자세한 정보를 제공합니다. companion 유틸리티 blkparse 는 blktrace 에서 원시 출력을 읽고 blktrace 에서 기록한 입력 및 출력 작업에 대한 사람이 읽을 수 있는 요약을 생성합니다.
- BTT 는 blktrace 출력을 분석하고 데이터가 I/O 스택의 각 영역에 보내는 시간을 표시하므로 I/O 하위 시스템에서 병목 현상을 쉽게 찾을 수 있습니다. 이 유틸리티는 blktrace 패키지의 일부로 제공됩니다. blktrace 메커니즘에서 추적하고 btt 에서 분석한 중요한 이벤트 중 일부는 다음과 같습니다.
  - I/O 이벤트 대기열(Q)
  - I/O를 드라이버 이벤트 (D)로 디스패치
  - I/O 이벤트 완료 (C)
- iowatcher 는 시간 경과에 따른 그래프 I/O에 blktrace 출력을 사용할 수 있습니다. 디스크 I/O의 논리 블록 주소(LBA), 초당 메가바이트당 처리량, 초당 검색 수 및 초당 I/O 작업 수에 중점을

둡니다. 이렇게 하면 장치의 1초 단위 제한에 도달할 때를 식별하는 데 도움이 될 수 있습니다.

- BCC(BPF Compiler Collection)는eBPF(extended Berkeley Packet Filter) 프로그램을 쉽게 생성할 수 있는 라이브러리입니다. eBPF 프로그램은 디스크 I/O, TCP 연결 및 프로세스 생성과 같은 이벤트에서 트리거됩니다. BCC 툴은 /usr/share/bcc/tools/ 디렉터리에 설치됩니다. 다음 bcc-tools 는 성능을 분석하는 데 도움이 됩니다.

  - interactivelatency 는 히스토그램의 블록 장치 I/O(디스크 I/O)의 대기 시간을 요약합니다. 이를 통해 장치 캐시 히트 및 캐시 누락에 대한 두 가지 모드를 포함하여 배포를 연구할 수 있으며 대기 시간이 초과됩니다.

  - biosnoop 는 발행 프로세스 ID 및 I/O 대기 시간과 함께 각 I/O 이벤트를 표시하는 기본 블록 I/O 추적 툴입니다. 이 도구를 사용하면 디스크 I/O 성능 문제를 조사할 수 있습니다.

  - biotop 는 커널의 블록 i/o 작업에 사용됩니다.

  - 파일life 툴은 stat() syscall을 추적합니다.

  - 더 낮은 파일 추적 속도가 동기 파일 읽기 및 쓰기 속도가 느립니다.

  - Filetop 는 프로세스별로 파일 읽기 및 쓰기를 표시합니다.

  - ext4slower,nfs slower 는 특정 임계값보다 느린 파일 시스템 작업을 표시하는 툴이며 기본값은 10ms 입니다.

    자세한 내용은 Analyzing system performance with BPF Compiler Collection 을 참조하십시오.

- bpftace 는 성능 문제를 분석하는 데 사용되는 eBPF 의 추적 언어입니다. 또한 시스템 관찰에 대해 BCC와 같은 추적 유틸리티를 제공합니다. 이 유틸리티는 I/O 성능 문제를 조사하는 데 유용합니다.

- 다음 SystemTap 스크립트는 스토리지 또는 파일 시스템 성능 문제를 진단하는 데 유용할 수 있습니다.

- ***disktop.stp: 5초마다 디스크를 읽거나 쓰는 상태를 확인하고 해당 기간 동안 상위 10개 항목을 출력합니다.***

- ***iotime.stp: 읽기 및 쓰기 작업에 소요된 시간과 읽기 및 쓰기 바이트 수를 출력합니다.***

- ***traceio.stp: 관찰된 누적 I/O 트래픽에 따라 상위 10개의 실행 파일을 1초마다 인쇄합니다.***

- ***traceio2.stp: 지정된 장치에 대한 읽기 및 쓰기가 발생할 때 실행 가능 이름 및 프로세스 식별자를 인쇄합니다.***

- ***inodewatch.stp: 지정된 메이저 또는 마이너 장치의 지정된 inode에 읽기 또는 쓰기가 발생할 때마다 실행 가능한 이름 및 프로세스 식별자를 출력합니다.***

- ***inodewatch2.stp: 지정된 메이저 또는 마이너 장치의 지정된 inode에서 속성이 변경될 때마다 실행 가능한 이름, 프로세스 식별자 및 속성을 인쇄합니다.***

추가 리소스

- ***vmstat(8), iostat(1), blktrace(8), blkparse(1), btt(1), bpftrace, iowatcher(1) 도움말 페이지***

- ***BPF Compiler Collection을 사용하여 시스템 성능 분석***

30.2. 파일 시스템 포맷에 사용 가능한 튜닝 옵션

장치를 포맷한 후에는 일부 파일 시스템 구성 결정을 변경할 수 없습니다.

다음은 스토리지 장치를 포맷하기 전에 사용할 수 있는 옵션입니다.

크기

워크로드에 맞게 적절하게 크기 조정된 파일 시스템을 생성합니다. 크기가 작은 파일 시스템은 파일 시스템 점검을 위해 시간과 메모리를 더 적게 필요로 합니다. 그러나 파일 시스템이 너무 작으면 성능이 높은 조각화로 인해 발생합니다.

블록 크기

블록은 파일 시스템의 작업 단위입니다. 블록 크기는 단일 블록에 저장할 수 있는 데이터의 양을 결정하므로 한 번에 쓰거나 읽는 최소 데이터 양을 결정합니다.

기본 블록 크기는 대부분의 사용 사례에 적합합니다. 그러나 파일 시스템은 블록 크기 또는 여러 블록의 크기가 일반적으로 한 번에 읽거나 쓰는 데이터의 양과 동일하거나 약간 큰 경우 데이터를 더 효율적으로 수행하고 저장합니다. 작은 파일은 여전히 전체 블록을 사용합니다. 파일을 여러 블록에 분산할 수 있지만 이로 인해 추가 런타임 오버헤드가 발생할 수 있습니다.

*또한 일부 파일 시스템은 특정 블록 수로 제한되며, 이로 인해 파일 시스템의 최대 크기가 제한됩니다. 블록 크기는 **mkfs** 명령으로 장치를 포맷할 때 파일 시스템 옵션의 일부로 지정됩니다. 블록 크기를 지정하는 매개변수는 파일 시스템에 따라 다릅니다.*

기하메트리

*파일 시스템의 **Geometry**는 파일 시스템 전체에서 데이터의 분산과 관련이 있습니다. 시스템에서 **RAID**와 같이 스트라이핑된 스토리지를 사용하는 경우 장치를 포맷할 때 데이터와 메타데이터를 기본 스토리지 기하메리와 정렬하여 성능을 향상시킬 수 있습니다.*

*많은 장치는 장치가 특정 파일 시스템으로 포맷될 때 자동으로 설정되는 권장 **geometry**를 내보냅니다. 장치가 이러한 권장 사항을 내보내지 않거나 권장 설정을 변경하려면 **mkfs** 명령을 사용하여 장치를 포맷할 때 **geometry**를 수동으로 지정해야 합니다.*

파일 시스템을 지정하는 매개 변수는 파일 시스템에 따라 다릅니다.

외부 저널

저널링 파일 시스템은 작업이 실행되기 전에 저널 파일에서 쓰기 작업 중에 수행할 변경 사항을 문서화합니다. 이렇게 하면 시스템 충돌 또는 정전 시 스토리지 장치가 손상될 가능성이 줄어들고 복구 프로세스가 빨라집니다.

참고

***Red Hat**은 외부 저널 옵션을 사용하지 않는 것이 좋습니다.*

메타데이터 사용량이 많은 워크로드에는 저널에 대한 매우 빈번한 업데이트가 포함됩니다. 더 큰 저널은 메모리를 더 사용하지만 쓰기 작업 빈도를 줄입니다. 또한 기본 스토리지보다 빠르고 빠른 전용 스토리지에 저널을 배치하여 메타데이터 집약적인 워크로드로 장치의 검색 시간을 개선할 수 있습니다.

주의

외부 저널이 신뢰할 수 있는지 확인합니다. 외부 저널 장치를 손실하면 파일 시스템이 손상됩니다. 외부 저널은 형식으로 생성해야 하며 마운트 시 저널 장치를 지정해야 합니다.

추가 리소스

- **mkfs(8)** 및 **mount(8)** 도움말 페이지
- 사용 가능한 파일 시스템 개요

30.3. 파일 시스템 마운트에 사용 가능한 튜닝 옵션

다음은 대부분의 파일 시스템에서 사용 가능한 옵션입니다. 장치가 마운트될 때 지정할 수 있습니다.

액세스 시간

파일을 읽을 때마다 해당 메타데이터는 액세스가 발생한 시점(시간)으로 업데이트됩니다. 여기에는 추가 쓰기 **I/O**가 포함됩니다. **relatime** 은 대부분의 파일 시스템에 대한 기본 **atime** 설정입니다.

그러나 이 메타데이터를 업데이트하는 데 시간이 소요되며 정확한 액세스 시간 데이터가 필요하지 않은 경우 **noatime** 마운트 옵션으로 파일 시스템을 마운트할 수 있습니다. 이 명령은 파일을 읽을 때 메타데이터 업데이트를 비활성화합니다. 또한 디렉터리를 읽을 때 메타데이터 업데이트를 비활성화하는 **nodiratime** 동작을 활성화합니다.

참고

no atime 마운트 옵션을 사용하여 일정 시간 업데이트를 비활성화하면 백업 프로그램(예: 백업 프로그램)에 의존하는 애플리케이션이 중단될 수 있습니다.

read-ahead

미리 필요한 데이터를 미리 가져와서 페이지 캐시에 로드할 수 있는 파일 액세스 권한의 속도가 빨라지므로 디스크에 있는 것보다 더 빠르게 검색할 수 있습니다. **read-ahead** 값이 클수록 시스템이 데

이터를 미리 가져옵니다.

Red Hat Enterprise Linux는 파일 시스템에 대해 감지한 내용에 따라 적절한 읽기-부트 값을 설정하려고 합니다. 그러나 정확한 탐지가 항상 가능한 것은 아닙니다. 예를 들어 스토리지 배열이 시스템에 단일 **LUN**으로 자신을 표시하는 경우 시스템은 단일 **LUN**을 감지하고 배열에 대한 적절한 읽기-마이마 값을 설정하지 않습니다.

순차적 **I/O** 스트리밍이 많은 워크로드에서는 종종 읽기/출력의 높은 읽기 값의 이점을 얻을 수 있습니다. **Red Hat Enterprise Linux**와 함께 제공되는 스토리지 관련 **tuned** 프로필은 **LVM** 스트라이핑을 사용하는 것처럼 미리 읽기 값을 높일 수 있지만 이러한 조정이 모든 워크로드에 대해 항상 충분하지는 않습니다.

추가 리소스

- **mount(8), xfs(5)** 및 **ext4(5)** 매뉴얼 페이지

30.4. 사용되지 않는 블록 삭제 유형

파일 시스템에서 사용하지 않는 블록을 정기적으로 삭제하는 것은 솔리드 스테이트 디스크와 씬 프로비저닝된 스토리지에 권장되는 방법입니다.

다음은 사용되지 않는 블록을 삭제하는 두 가지 방법입니다.

배치 삭제

이러한 유형의 삭제는 **fstrim** 명령의 일부입니다. 관리자가 지정한 조건과 일치하는 파일 시스템에서 사용되지 않은 모든 블록을 삭제합니다. **Red Hat Enterprise Linux 9**는 물리적 삭제 작업을 지원하는 **XFS** 및 **ext4** 형식의 장치에서 배치 삭제 기능을 지원합니다.

온라인 삭제

이 유형의 삭제 작업은 **discard** 옵션을 사용하여 마운트 시 구성되며 사용자 개입 없이 실시간으로 실행됩니다. 그러나 에서 무료로 전환되는 블록만 삭제합니다. **Red Hat Enterprise Linux 9**는 **XFS** 및 **ext4** 형식의 장치에서 온라인 삭제 기능을 지원합니다.

Red Hat은 성능을 유지하기 위해 온라인 삭제가 필요한 경우 또는 배치 삭제가 시스템 워크로드에 적합하지 않은 경우를 제외하고 배치 삭제를 권장합니다.

사전 할당은 해당 공간에 데이터를 쓰지 않고 파일에 디스크 공간이 할당되어 있음을 나타냅니다. 이는

데이터 조각화 및 읽기 성능이 저하되는 데 유용할 수 있습니다. Red Hat Enterprise Linux 9는 XFS, ext4 및 GFS2 파일 시스템의 사전 할당 공간을 지원합니다. 애플리케이션은 fallocate(2) glibc 호출을 사용하여 공간을 미리 배치하면 이점을 얻을 수 있습니다.

추가 리소스

- mount(8) 및 fallocate(2) 매뉴얼 페이지

30.5. 솔리드 스테이트 디스크 튜닝 고려 사항

SSD(솔리드 스테이트 디스크)는 영구 데이터를 저장하기 위해 회전하는 마운드 플래터 대신 NAND Flash chip을 사용합니다. SSD는 전체 논리 블록 주소 범위에서 데이터에 대한 일정한 액세스 시간을 제공하며 회전하는 상대와 같은 비용을 측정할 수 없습니다. 1GB의 스토리지 공간당 비용이 많이 들고 스토리지 밀도가 줄어들지만, memory보다 대기 시간이 단축되고 처리량이 향상됩니다.

SSD에서 사용된 블록이 디스크의 용량에 접근하므로 일반적으로 성능이 저하됩니다. 성능 저하 수준은 벤더에 따라 달라지지만 모든 장치는 이러한 성능 저하를 경험할 수 있습니다. 삭제 동작을 활성화하면 이러한 성능 저하를 완화하는 데 도움이 될 수 있습니다. 자세한 내용은 사용되지 않는 블록 삭제 유형을 참조하십시오.

기본 I/O 스케줄러 및 가상 메모리 옵션은 SSD와 함께 사용하기에 적합합니다. SSD 성능에 영향을 줄 수 있는 설정을 구성할 때 다음 요소를 고려하십시오.

I/O 스케줄러

모든 I/O 스케줄러는 대부분의 SSD에서 제대로 작동할 것으로 예상됩니다. 그러나 다른 스토리지 유형과 마찬가지로 Red Hat은 지정된 워크로드에 대한 최적의 구성을 확인하기 위해 벤치마킹을 권장합니다. SSD를 사용하는 경우 Red Hat은 특정 워크로드의 벤치마킹을 위해서만 I/O 스케줄러 변경을 권장합니다. I/O 스케줄러 간에 전환하는 방법에 대한 지침은 /usr/share/doc/kernel-version/Documentation/block/sched.txt 파일을 참조하십시오.

단일 대기열 HBA의 경우 기본 I/O 스케줄러는 데드라인 입니다. 다중 대기열 HBA의 경우 기본 I/O 스케줄러는 none 입니다. I/O 스케줄러 설정 방법에 대한 자세한 내용은 디스크 스케줄러 설정을 참조하십시오.

가상 메모리

I/O 스케줄러와 마찬가지로 VM(가상 메모리) 하위 시스템에는 특별한 튜닝이 필요하지 않습니다. SSD의 I/O의 빠른 특성을 고려할 때 vm_dirty_backfield_ratio 및 vm_dirty_ratio 설정을 끄면 일반적으로 디스크의 다른 작업 대기 시간에 부정적인 영향을 미치지 않습니다. 그러나 이러한 튜닝은 전체 I/O를 더 많이 생성할 수 있으므로 일반적으로 워크로드별 테스트 없이 권장되지 않습니다.

swap

SSD도 스왑 장치로 사용할 수 있으며 좋은 페이지 아웃 및 페이지인 성능을 생성할 수 있습니다.

30.6. 일반 블록 장치 튜닝 매개변수

이 섹션에 나열된 일반 튜닝 매개 변수는 **/sys/block/sdX/queue/** 디렉터리에서 사용할 수 있습니다.

다음 나열된 튜닝 매개변수는 **I/O** 스케줄러 튜닝과 다르며 모든 **I/O** 스케줄러에 적용할 수 있습니다.

add_random

일부 **I/O** 이벤트는 **/dev/random** 의 엔트로피 풀에 기여합니다. 이러한 기여의 오버헤드가 측정 가능한 경우 이 매개변수를 **0** 으로 설정할 수 있습니다.

iostats

기본적으로 **iostats** 는 활성화되어 있으며 기본값은 **1** 입니다. **iostats** 값을 **0** 으로 설정하면 장치에 대한 **I/O** 통계 수집이 비활성화되어 **I/O** 경로에 적은 오버헤드가 제거됩니다. **iostats** 를 **0** 으로 설정하면 특정 **NVMe** 솔리드 스테이트 스토리지 장치와 같은 매우 고성능 장치의 성능이 약간 향상될 수 있습니다. 벤더가 지정된 스토리지 모델에 달리 지정하지 않는 한 **iostats** 를 활성화 상태로 두는 것이 좋습니다.

iostats 를 비활성화하면 장치에 대한 **I/O** 통계가 더 이상 **/proc/diskstats** 파일에 존재하지 않습니다. **/sys/diskstats** 파일의 내용은 **sar** 또는 **iostats** 와 같은 **I/O** 툴 모니터링을 위한 **I/O** 정보의 소스입니다. 따라서 장치에 대한 **iostats** 매개변수를 비활성화하면 장치가 더 이상 **I/O** 모니터링 툴 출력에 표시되지 않습니다.

max_sectors_kb

I/O 요청의 최대 크기를 킬로바이트로 지정합니다. 기본값은 **512 KB**입니다. 이 매개 변수의 최소값은 스토리지 장치의 논리 블록 크기에 따라 결정됩니다. 이 매개변수의 최대값은 **max_hw_sectors_kb** 의 값으로 결정됩니다.

Red Hat은 **max_sectors_kb** 가 항상 최적의 **I/O** 크기의 여러 개이며 내부 클리어 블록 크기가 클 것을 권장합니다. 스토리지 장치에 의해 **0**이거나 지정되지 않은 경우 매개 변수에 **logical_block_size** 값을 사용합니다.

nomerges

대부분의 워크로드는 요청 병합을 통해 이점을 얻을 수 있습니다. 그러나 병합을 비활성화하면 디버깅에 유용할 수 있습니다. 기본적으로 **nomerges** 매개변수는 **0** 으로 설정되어 병합이 가능합니다. 간단한 일회성 병합을 비활성화하려면 **nomerges** 를 **1** 로 설정합니다. 모든 유형의 병합을 비활성화하려면 **nomerges** 를 **2** 로 설정합니다.

nr_requests

*대기 중인 **I/O**의 허용된 최대 수입니다. 현재 **I/O** 스케줄러가 **none** 이면 이 수를 줄일 수 있습니다. 그렇지 않으면 수를 늘리거나 줄일 수 있습니다.*

optimal_io_size

*일부 스토리지 장치는 이 매개변수를 통해 최적의 **I/O** 크기를 보고합니다. 이 값이 보고되면 **Red Hat**은 가능한 경우 최적의 **I/O** 크기와 일치하는 **I/O**와 관련된 **I/O**를 실행하는 것이 좋습니다.*

read_ahead_kb

*연속 읽기 작업 중에 운영 체제가 미리 읽을 수 있는 최대 킬로바이트 수를 정의합니다. 결과적으로 다음 순차 읽기의 커널 페이지 캐시에 필요한 정보가 이미 있으므로 읽기 **I/O** 성능이 향상됩니다.*

*장치 매퍼는 종종 상위 **read_ahead_kb** 값의 이점을 얻습니다. 각 장치를 매핑할 수 있는 **128 KB**는 좋은 시작점이지만, 디스크의 **queue**의 **max_sectors_kb** 값을 요청하도록 **read_ahead_kb** 값을 늘리면 대용량 파일을 순차적으로 읽을 수 있는 애플리케이션 환경에서 성능이 향상될 수 있습니다.*

rotational

*일부 솔리드 스테이트 디스크는 솔리드 스테이트 상태를 올바르게 공개하지 않으며 기존 교체 디스크로 마운트됩니다. 스케줄러에서 불필요한 검색 논리를 비활성화하려면 **rotation** 값을 **0** 으로 수동으로 설정합니다.*

rq_affinity

***rq_affinity** 의 기본값은 **1** 입니다. 실행된 **CPU** 코어의 동일한 **CPU** 그룹에 있는 하나의 **CPU** 코어에서 **I/O** 작업을 완료합니다. **I/O** 요청을 발급한 프로세서에서만 완료를 수행하려면 **rq_affinity** 를 **2** 로 설정합니다. 언급된 두 가지 능력을 비활성화하려면 **0** 으로 설정하십시오.*

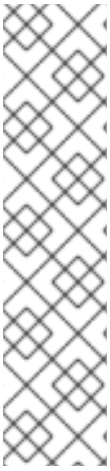
scheduler

*특정 스토리지 장치에 대한 스케줄러 또는 스케줄러 기본 순서를 설정하려면 **/sys/block/devname/queue/scheduler** 파일을 편집합니다. 여기서 **devname** 은 구성하려는 장치의 이름입니다.*

31장. SYSTEMD를 사용하여 애플리케이션에서 사용하는 리소스 관리

RHEL 9에서는 cgroup 계층 구조의 시스템을 systemd 단위 트리에 바인딩하여 리소스 관리 설정을 프로세스 수준에서 애플리케이션 수준으로 이동합니다. 따라서 systemctl 명령을 사용하거나 systemd 장치 파일을 수정하여 시스템 리소스를 관리할 수 있습니다.

이를 위해 systemd 는 단위 파일에서 또는 systemctl 명령을 통해 직접 다양한 구성 옵션을 사용합니다. 그런 다음 systemd 는 Linux 커널 시스템 호출 및 cgroups 및 네임스페이스 와 같은 기능을 활용하여 특정 프로세스 그룹에 이러한 옵션을 적용합니다.



참고

다음 도움말 페이지에서 systemd 에 대한 전체 설정 옵션 세트를 검토할 수 있습니다.

- systemd.resource-control(5)
- systemd.exec(5)

31.1. SYSTEMD를 사용하여 시스템 리소스 할당

시스템 리소스의 배포를 수정하려면 다음 배포 모델 중 하나 이상을 적용할 수 있습니다.

weights

모든 하위 그룹의 가중치를 추가하고 각 하위 그룹에 합계와 일치하는 분수를 지정하여 리소스를 배포할 수 있습니다.

예를 들어 cgroup이 10개의 경우 각각 값 100의 가중치가 있으면 합계는 1000입니다. 각 cgroup은 리소스 1/10을 수신합니다.

weight는 일반적으로 상태 비저장 리소스를 배포하는 데 사용됩니다. 예를 들어 CPUWeight= 옵션은 이 리소스 배포 모델의 구현입니다.

제한

cgroup은 구성된 리소스 양까지 사용할 수 있습니다. 하위 그룹 제한의 합계는 상위 cgroup의 제한을 초과할 수 있습니다. 따라서 이 모델에서 리소스를 과다 할당할 수 있습니다.

*예를 들어 **MemoryMax=** 옵션은 이 리소스 배포 모델의 구현입니다.*

보호

__cgroup__에 대해 보호된 리소스 양을 설정할 수 있습니다. 리소스 사용량이 보호 경계 아래에 있는 경우 커널은 동일한 리소스에 대해 경쟁하는 다른 __cgroups__에 우선하여 이 __cgroup__에 영향을 미치지 않습니다. 오버 커밋도 가능합니다.

*예를 들어 **MemoryLow=** 옵션은 이 리소스 배포 모델의 구현입니다.*

할당

무한한 리소스의 절대 할당에 대한 배타적 할당입니다. 오버 커밋이 불가능합니다. __Linux__의 이러한 리소스 유형의 예로는 실시간 예산이 있습니다.

단위 파일 옵션

리소스 제어 구성 설정입니다.

*예를 들어 **CPUAccounting=** 또는 **CPUQuota=** 와 같은 옵션을 사용하여 __CPU__ 리소스를 구성할 수 있습니다. 마찬가지로 **AllowedMemoryNodes=** 및 **IOAccounting=** 와 같은 옵션을 사용하여 메모리 또는 __I/O__ 리소스를 구성할 수 있습니다.*

절차

*서비스의 단위 파일 옵션 값을 변경하려면 단위 파일의 값을 조정하거나 **systemctl** 명령을 사용할 수 있습니다.*

1. *선택한 서비스에 대해 할당된 값을 확인합니다.*

```
# systemctl show --property <unit file option> <service name>
```

2. *__CPU__ 시간 할당 정책 옵션의 필요한 값을 설정합니다.*

```
# systemctl set-property <service name> <unit file option>=<value>
```

검증 단계

- 선택한 서비스에 대해 새로 할당된 값을 확인합니다.

```
# systemctl show --property <unit file option> <service name>
```

추가 리소스

- systemd.resource-control(5), systemd.exec(5) manual pages

31.2. 리소스 관리에서 SYSTEMD의 역할

systemd 의 핵심 기능은 서비스 관리 및 감독입니다. systemd 시스템 및 서비스 관리자는 관리 서비스가 부팅 프로세스 중에 적시에 올바른 순서로 시작되도록 합니다. 기본 하드웨어 플랫폼을 최적으로 사용하려면 서비스를 원활하게 실행해야 합니다. 따라서 systemd 는 리소스 관리 정책을 정의하고 서비스의 성능을 향상시킬 수 있는 다양한 옵션을 조정하는 기능도 제공합니다.

중요

일반적으로 Red Hat은 systemd 를 사용하여 시스템 리소스 사용을 제어하는 것이 좋습니다. 특수한 경우에만 cgroup 가상 파일 시스템을 수동으로 구성해야 합니다. 예를 들어 cgroup-v2 계층 구조에는 동일하지 않은 cgroup-v1 컨트롤러를 사용해야 합니다.

31.3. CGROUPS의 SYSTEMD 계층 구조 개요

백엔드에서 systemd 시스템 및 서비스 관리자는 슬라이스, 범위 및 서비스 단위를 사용하여 제어 그룹의 프로세스를 구성하고 구성합니다. 사용자 지정 유닛 파일을 생성하거나 systemctl 명령을 사용하여 이 계층을 추가로 수정할 수 있습니다. 또한 systemd 는 /sys/fs/cgroup/ 디렉터리에 중요한 커널 리소스 컨트롤러에 대한 계층을 자동으로 마운트합니다.

리소스 제어에는 세 가지 systemd 장치 유형이 사용됩니다.

- Service - 단위 구성 파일에 따라 systemd 가 시작된 프로세스 또는 프로세스 그룹입니다. 서비스는 지정된 프로세스를 캡슐화하여 하나의 세트로 시작 및 중지할 수 있습니다. 서비스의 이름은 다음과 같이 지정됩니다.

```
<name>.service
```

- 범위 - 외부에서 생성된 프로세스 그룹입니다. 범위는 fork() 함수를 통해 임의의 프로세스에서 시작 및 중지된 프로세스를 캡슐화한 다음 런타임에 systemd 에 의해 등록됩니다. 예를 들어

사용자 세션, 컨테이너 및 가상 머신은 범위로 처리됩니다. 범위는 다음과 같이 이름이 지정됩니다.

```
<name>.scope
```

- 슬라이스 - 계층적으로 구성된 단위로 이루어진 그룹입니다. 슬라이스는 범위가 배치되는 계층 구조를 구성합니다. 실제 프로세스는 범위 또는 서비스에 포함됩니다. 슬라이스 단위의 모든 이름은 계층 구조의 위치 경로에 해당합니다. 대시("-") 문자는 **-.slice** 루트 슬라이스의 슬라이스에 대한 경로 구성 요소의 구분자 역할을 합니다. 다음 예에서:

```
<parent-name>.slice
```

parent-name.slice 는 **-.slice** 루트 슬라이스의 하위 디렉터리인 **parent.slice.slice**입니다. **parent-name.slice** 에는 **parent-name-name2.slice** 라는 고유한 하위 디렉터리가 있을 수 있습니다.

서비스, 범위, 슬라이스 단위는 제어 그룹 계층 구조의 개체에 직접 매핑됩니다. 이러한 장치가 활성화되면 직접 매핑하여 장치 이름에서 구축된 그룹 경로를 제어합니다.

다음은 제어 그룹 계층 구조의 축약된 예입니다.

```
Control group /:
-.slice
├─user.slice
│ ├─user-42.slice
│ │ ├─session-c1.scope
│ │ │ ├─ 967 gdm-session-worker [pam/gdm-launch-environment]
│ │ │ ├─1035 /usr/libexec/gdm-x-session gnome-session --autostart
/usr/share/gdm/greeter/autostart
│ │ │ ├─1054 /usr/libexec/Xorg vt1 -displayfd 3 -auth /run/user/42/gdm/Xauthority -
background none -noreset -keeptty -verbose 3
│ │ │ ├─1212 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart
│ │ │ ├─1369 /usr/bin/gnome-shell
│ │ │ ├─1732 ibus-daemon --xim --panel disable
│ │ │ ├─1752 /usr/libexec/ibus-dconf
│ │ │ ├─1762 /usr/libexec/ibus-x11 --kill-daemon
│ │ │ ├─1912 /usr/libexec/gsd-xsettings
│ │ │ ├─1917 /usr/libexec/gsd-a11y-settings
│ │ │ ├─1920 /usr/libexec/gsd-clipboard
…
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
  ├─rngd.service
  │ └─800 /sbin/rngd -f
  ├─systemd-udevd.service
```

```
│ └─659 /usr/lib/systemd/systemd-udevd
├─chronyd.service
│ └─823 /usr/sbin/chronyd
├─auditd.service
│ ├─761 /sbin/auditd
│ └─763 /usr/sbin/sedispatch
├─accounts-daemon.service
│ └─876 /usr/libexec/accounts-daemon
├─example.service
│ ├─ 929 /bin/bash /home/jdoe/example.sh
│ └─4902 sleep 1
…
```

위의 예제에서는 서비스 및 범위에 프로세스가 포함되어 있지 않은 슬라이스에 배치되어 있음을 보여줍니다.

추가 리소스

- Red Hat Enterprise Linux에서 기본 시스템 설정 구성
- 커널 리소스 컨트롤러란?
- systemd.resource-control(5), systemd.exec(5), cgroups(7), fork (), fork(2) 매뉴얼 페이지
- cgroups 이해

31.4. SYSTEMD 단위 나열

다음 절차에서는 **systemd** 시스템 및 서비스 관리자를 사용하여 단위를 나열하는 방법을 설명합니다.

절차

- 시스템의 모든 활성 단위를 나열하려면 **# systemctl** 명령을 실행하면 터미널이 다음 예제와 유사한 출력을 반환합니다.

```
# systemctl
UNIT                                LOAD   ACTIVE SUB       DESCRIPTION
…
init.scope                            loaded active running   System and Service Manager
session-2.scope                         loaded active running   Session 2 of user jdoe
```

```
abrt-ccpp.service                          loaded active exited    Install ABRT coredump
hook
abrt-oops.service                          loaded active running   ABRT kernel log watcher
abrt-vmcore.service                         loaded active exited    Harvest vmcores for
ABRT
abrt-xorg.service                          loaded active running   ABRT Xorg log watcher
...
-.slice                                  loaded active active    Root Slice
machine.slice                              loaded active active    Virtual Machine and
Container Slice system-getty.slice                                        loaded
active active    system-getty.slice
system-lvm2\x2dpvscan.slice                    loaded active active    system-
lvm2\x2dpvscan.slice
system-sshd\x2dkeygen.slice                    loaded active active    system-
sshd\x2dkeygen.slice
system-systemd\x2dhibernate\x2dresume.slice        loaded active active    system-
systemd\x2dhibernate\x2dresume>
system-user\x2druntime\x2ddir.slice               loaded active active    system-
user\x2druntime\x2ddir.slice
system.slice                               loaded active active    System Slice
user-1000.slice                             loaded active active    User Slice of UID 1000
user-42.slice                              loaded active active    User Slice of UID 42
user.slice                                loaded active active    User and Session Slice
...
```

- *UNIT - 제어 그룹 계층 구조의 단위 위치도 반영하는 단위의 이름입니다. 리소스 제어와 관련된 단위는 슬라이스, 범위, 서비스 입니다.*

- *LOAD - 단위 구성 파일이 제대로 로드되었는지 여부를 나타냅니다. 단위 파일을 로드하는 데 실패한 경우 필드에 로드된 대신 상태 오류가 포함됩니다. 기타 단위 로드 상태는 스텁,병합된, 마스킹 입니다.*

- *Net Namespace - SUB 를 일반화한 고급 단위 활성화 상태입니다.*

- *SUB - 낮은 수준의 유닛 활성화 상태입니다. 가능한 값의 범위는 단위 유형에 따라 다릅니다.*

- *DESCRIPTION - 단위 콘텐츠 및 기능에 대한 설명입니다.*

- *비활성 단위를 나열하려면 다음을 실행합니다.*

```
# systemctl --all
```

- 출력의 정보 양을 제한하려면 다음을 실행합니다.

```
# systemctl --type service,masked
```

--type 옵션에는 서비스 및 슬라이스 와 같은 단위 유형 또는 로드 및 마스크 와 같은 단위 로드 상태 목록이 쉼표로 구분되어 있어야 합니다.

추가 리소스

- **RHEL**에서 기본 시스템 설정 구성
- **systemd.resource-control(5), systemd.exec(5) manual pages**

31.5. SYSTEMD 제어 그룹 계층 구조 보기

다음 절차에서는 특정 **cgroups** 에서 실행되는 제어 그룹(**cgroups**) 계층 구조 및 프로세스를 표시하는 방법을 설명합니다.

절차

- 시스템의 전체 **cgroups** 계층 구조를 표시하려면 **# systemd-cgls** 를 실행합니다.

```
# systemd-cgls
Control group /:
-.slice
├─user.slice
│ ├─user-42.slice
│ │ ├─session-c1.scope
│ │ │ ├─ 965 gdm-session-worker [pam/gdm-launch-environment]
│ │ │ ├─1040 /usr/libexec/gdm-x-session gnome-session --autostart
/usr/share/gdm/greeter/autostart
...
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
  ...
  ├─example.service
  │ ├─6882 /bin/bash /home/jdoe/example.sh
  │ └─6902 sleep 1
  ├─systemd-journald.service
  │ └─629 /usr/lib/systemd/systemd-journald
  ...
```

*예제 출력은 전체 **cgroups** 계층을 반환하며, 가장 높은 수준은 슬라이스 순으로 구성됩니다.*

- *리소스 컨트롤러에서 필터링한 **cgroups** 계층 구조를 표시하려면 **# systemd-cgls <resource_controller > :**을 실행합니다.*

```
# systemd-cgls memory
Controller memory; Control group /:
├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
├─user.slice
│ ├─user-42.slice
│ │ ├─session-c1.scope
│ │ │ ├─ 965 gdm-session-worker [pam/gdm-launch-environment]
...
└─system.slice
  |
  ...
  ├─chronyd.service
  │ └─844 /usr/sbin/chronyd
  ├─example.service
  │ ├─8914 /bin/bash /home/jdoe/example.sh
  │ └─8916 sleep 1
  ...
```

위의 명령의 예제 출력에는 선택한 컨트롤러와 상호 작용하는 서비스가 나열됩니다.

- *특정 단위 및 **cgroup** 계층 구조의 일부에 대한 자세한 정보를 표시하려면 **# systemctl status <system_unit>:***

```
# systemctl status example.service
● example.service - My example service
   Loaded: loaded (/usr/lib/systemd/system/example.service; enabled; vendor preset:
disabled)
   Active: active (running) since Tue 2019-04-16 12:12:39 CEST; 3s ago
 Main PID: 17737 (bash)
    Tasks: 2 (limit: 11522)
   Memory: 496.0K (limit: 1.5M)
   CGroup: /system.slice/example.service
           ├─17737 /bin/bash /home/jdoe/example.sh
           └─17743 sleep 1
Apr 16 12:12:39 redhat systemd[1]: Started My example service.
Apr 16 12:12:39 redhat bash[17737]: The current time is Tue Apr 16 12:12:39 CEST
2019
Apr 16 12:12:40 redhat bash[17737]: The current time is Tue Apr 16 12:12:40 CEST
2019
```

추가 리소스

- *커널 리소스 컨트롤러란?*
- ***systemd.resource-control(5), cgroups(7) manual pages***

31.6. 프로세스 CGROUPS 보기

*다음 절차에서는 프로세스가 속한 **C Group(**컨트롤그룹 **)**을 확인하는 방법을 설명합니다. 그런 다음 **cgroup** 을 확인하여 사용하는 컨트롤러 및 컨트롤러별 구성을 확인할 수 있습니다.*

절차

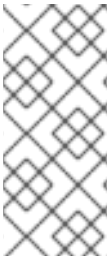
1. *프로세스가 속한 **cgroup** 을 보려면 **# cat proc/<PID>/cgroup** 명령을 실행합니다.*

```
# cat /proc/2467/cgroup
0::/system.slice/example.service
```

*예제 출력은 관심 프로세스와 관련이 있습니다. 이 경우 **PID 2467** 에 의해 식별되는 프로세스이며 **example.service** 단위에 속합니다. 프로세스가 **systemd** 장치 파일 사양에 정의된 대로 올바른 제어 그룹에 배치되었는지 여부를 확인할 수 있습니다.*

2. ***cgroup** 이 사용하는 컨트롤러와 각 구성 파일을 표시하려면 **cgroup** 디렉터리를 확인합니다.*

```
# cat /sys/fs/cgroup/system.slice/example.service/cgroup.controllers
memory pids

# ls /sys/fs/cgroup/system.slice/example.service/
cgroup.controllers
cgroup.events
...
cpu.pressure
cpu.stat
io.pressure
memory.current
memory.events
...
pids.current
pids.events
pids.max
```



참고

cgroup 의 버전 1 계층 구조에서는 컨트롤러별 모델을 사용합니다. 따라서 /proc/PID/cgroup 파일의 출력은 PID가 속한 각 컨트롤러 아래에 있는 cgroup 을 표시합니다. 해당 cgroup 은 /sys/fs/cgroup/ <controller_name>/ 에서 확인할 수 있습니다.

추가 리소스

- cgroups ImageCon 매뉴얼 페이지
- 커널 리소스 컨트롤러란?
- /usr/share/doc/kernel-doc-<kernel_version>/Documentation/admin-guide/cgroup-v2.rst 파일에 있는 문서(kernel-doc 패키지를 설치한 후)

31.7. 리소스 사용 모니터링

다음 절차에서는 현재 실행 중인 제어 그룹(cgroups) 및 리소스 소비 목록을 실시간으로 확인하는 방법을 설명합니다.

절차

1. 현재 실행 중인 cgroups 의 동적 계정을 보려면 # systemd-cgtop 명령을 실행합니다.

```
# systemd-cgtop
Control Group                          Tasks   %CPU   Memory  Input/s Output/s
/                                        607   29.8     1.5G        -        -
/system.slice                            125      -   428.7M        -        -
/system.slice/ModemManager.service         3      -     8.6M        -        -
/system.slice/NetworkManager.service       3      -    12.8M        -        -
/system.slice/accounts-daemon.service      3      -     1.8M        -        -
/system.slice/boot.mount                   -      -    48.0K        -        -
/system.slice/chronyd.service              1      -     2.0M        -        -
/system.slice/cockpit.socket               -      -     1.3M        -        -
/system.slice/colord.service               3      -     3.5M        -        -
/system.slice/crond.service                1      -     1.8M        -        -
/system.slice/cups.service                 1      -     3.1M        -        -
/system.slice/dev-hugepages.mount          -      -   244.0K        -        -
/system.slice/dev-mapper-rhel\x2dswap.swap  -      -   912.0K        -        -
/system.slice/dev-mqueue.mount             -      -    48.0K        -        -
/system.slice/example.service              2      -     2.0M        -        -
/system.slice/firewalld.service            2      -    28.8M        -        -
...
```

예제 출력에는 리소스 사용량(CPU, 메모리, 디스크 I/O 로드)으로 정렬된 현재 실행 중인 cgroup 이 표시됩니다. 목록은 기본적으로 1초마다 새로 고쳐집니다. 따라서 각 제어 그룹의 실제 리소스 사용에 대한 동적 통찰력을 제공합니다.

추가 리소스

- systemd-cgtop(1) 매뉴얼 페이지

31.8. SYSTEMD 장치 파일을 사용하여 애플리케이션 제한 설정

기존 또는 실행 중인 각 단위는 systemd 에 의해 감독되며 이를 위해 제어 그룹도 생성합니다. 유닛에는 /usr/lib/systemd/system/ 디렉터리에 구성 파일이 있습니다. 장치 파일을 수동으로 수정하여 제한 사항을 설정하거나 프로세스 그룹의 하드웨어 리소스에 대한 액세스를 제어할 수 있습니다.

사전 요구 사항

- 루트 권한이 있어야 합니다.

절차

1. /usr/lib/systemd/system/example.service 파일을 수정하여 서비스의 메모리 사용량을 제한합니다.

```
...
[Service]
MemoryMax=1500K
...
```

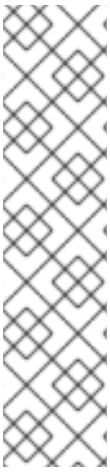
위의 구성은 최대 메모리 제한을 배치합니다. 이 제한은 제어 그룹의 프로세스가 초과할 수 없습니다. example.service 서비스는 제한 사항이 있는 제어 그룹의 일부입니다. 접미사 K, M, G 또는 T를 사용하여 Kilobyte, Megabyte, 기가바이트 또는 Terabyte를 측정 단위로 식별할 수 있습니다.

2. 모든 단위 구성 파일을 다시 로드합니다.

```
# systemctl daemon-reload
```

3. 서비스를 다시 시작하십시오.

```
# systemctl restart example.service
```



참고

다음 도움말 페이지에서 **systemd** 에 대한 전체 설정 옵션 세트를 검토할 수 있습니다.

- **systemd.resource-control(5)**
- **systemd.exec(5)**

검증

1. 변경 사항이 적용되었는지 확인합니다.

```
# cat /sys/fs/cgroup/system.slice/example.service/memory.max
1536000
```

예제 출력에서는 메모리 사용량이 약 **1,500KB**로 제한되었음을 보여줍니다.

추가 리소스

- *cgroups 이해*
- **Red Hat Enterprise Linux**에서 *기본 시스템 설정 구성*
- **systemd.resource-control(5), systemd.exec(5), cgroups(7) manual pages**

31.9. SYSTEMCTL 명령을 사용하여 애플리케이션 제한 설정

CPU 선호도 설정은 특정 프로세스의 액세스를 일부 **CPU**로 제한하는 데 도움이 됩니다. 효과적으로 **CPU** 스케줄러는 프로세스의 유사성 마스크에 없는 **CPU**에서 실행되도록 프로세스를 예약하지 않습니다.

기본 **CPU** 선호도 마스크는 **systemd** 에서 관리하는 모든 서비스에 적용됩니다.

*특정 **systemd** 서비스에 대한 **CPU** 선호도 마스크를 구성하기 위해 **systemd** 는 **/etc/systemd/system.conf** 파일에 있는 장치 파일 옵션과 관리자 구성 옵션으로 **CPUAffinity=** 을 제공합니다.*

***CPUAffinity=** 장치 파일 옵션은 병합되어 선호도 마스크로 사용되는 **CPU** 또는 **CPU** 범위 목록을 설정합니다.*

*특정 **systemd** 서비스에 대한 **CPU** 선호도 마스크를 구성한 후 서비스를 다시 시작하여 변경 사항을 적용해야 합니다.*

절차

***CPU Affinity** 장치 파일 옵션을 사용하여 특정 **systemd** 서비스의 **CPU** 선호도 마스크를 설정하려면 다음을 수행합니다.*

1. *선택한 서비스에서 **CPUAffinity** 단위 파일 옵션 값을 확인합니다.*

```
$ systemctl show --property <CPU affinity configuration option> <service name>
```

2. *루트로서 선호도 마스크로 사용되는 **CPU** 범위에 대해 **CPUAffinity** 단위 파일 옵션의 필요한 값을 설정합니다.*

```
# systemctl set-property <service name> CPUAffinity=<value>
```

3. *서비스를 다시 시작하여 변경 사항을 적용합니다.*

```
# systemctl restart <service name>
```

참고

*다음 도움말 페이지에서 **systemd** 에 대한 전체 설정 옵션 세트를 검토할 수 있습니다.*

- ***systemd.resource-control(5)***
- ***systemd.exec(5)***

31.10. 관리자 구성을 통해 글로벌 기본 CPU 선호도 설정

/etc/systemd/system.conf 파일의 CPUAffinity 옵션은 PID(프로세스 ID) 1과 PID1에서 분기된 모든 프로세스에 대한 선호도 마스크를 정의합니다. 그런 다음 서비스별로 CPUAffinity 를 덮어쓸 수 있습니다.

관리자 구성 옵션을 사용하여 모든 systemd 서비스에 대한 기본 CPU 선호도 마스크를 설정하려면 다음을 수행합니다.

1. /etc/systemd/system.conf 파일에서 CPUAffinity= 옵션의 CPU 번호를 설정합니다.
2. 편집한 파일을 저장하고 systemd 서비스를 다시 로드합니다.

```
# systemctl daemon-reload
```

3. 서버를 재부팅하여 변경 사항을 적용합니다.

참고

다음 도움말 페이지에서 systemd 에 대한 전체 설정 옵션 세트를 검토할 수 있습니다.

- systemd.resource-control(5)
- systemd.exec(5)

31.11. SYSTEMD를 사용하여 NUMA 정책 구성

NUMA(Non-Uniform Memory Access)는 컴퓨터 메모리 하위 시스템 설계로, 프로세서와 관련된 실제 메모리 위치에 따라 메모리 액세스 시간이 달라집니다.

CPU에 가까운 메모리는 다른 CPU (foreign memory)에 대해 로컬되거나 CPU 세트 간에 공유되는 메모리보다 대기 시간(로컬 메모리)이 더 적습니다.

Linux 커널 측면에서 NUMA 정책은 커널이 프로세스에 실제 메모리 페이지를 할당하는 위치(예:

NUMA 노드)를 관리합니다.

systemd 는 서비스의 메모리 할당 정책을 제어하기 위해 장치 파일 옵션 NUMAPolicy 및 NUMAMask 를 제공합니다.

절차

NUMA Policy 장치 파일 옵션을 통해 NUMA 메모리 정책을 설정하려면 다음을 수행합니다.

1. *선택한 서비스에서 NUMAPolicy 단위 파일 옵션 값을 확인합니다.*

```
$ systemctl show --property <NUMA policy configuration option> <service name>
```

2. *루트로서 NUMAPolicy 단위 파일 옵션의 필수 정책 유형을 설정합니다.*

```
# systemctl set-property <service name> NUMAPolicy=<value>
```

3. *서비스를 다시 시작하여 변경 사항을 적용합니다.*

```
# systemctl restart <service name>
```

관리자 구성 옵션을 통해 글로벌 NUMAPolicy 설정을 설정하려면 다음을 수행합니다.

1. */etc/systemd/system.conf 파일에서 NUMAPolicy 옵션을 검색합니다.*

2. *정책 유형을 편집하고 파일을 저장합니다.*

3. *systemd 구성을 다시 로드합니다.*

```
# systemd daemon-reload
```

4. *서버를 재부팅합니다.*

중요

엄격한 NUMA 정책을 구성할 때(예: 바인딩)는 CPUAffinity= 단위 파일 옵션도 적절히 설정해야 합니다.

추가 리소스

- systemctl 명령을 사용하여 애플리케이션 제한 설정
- systemd.resource-control(5), systemd.exec(5), set_mempolicy(2) manual pages.

31.12. SYSTEMD에 대한 NUMA 정책 구성 옵션

systemd는 NUMA 정책을 구성하는 다음 옵션을 제공합니다.

NUMAPolicy

실행된 프로세스의 NUMA 메모리 정책을 제어합니다. 다음과 같은 정책 유형을 사용할 수 있습니다.

- default
- preferred
- bind
- interleave
- 로컬

NUMAMask

선택한 NUMA 정책과 연결된 NUMA 노드 목록을 제어합니다.

다음 정책에 대해 **NUMAMask** 옵션을 지정할 필요가 없습니다.

- **default**
- 로컬

기본 정책의 경우 목록은 단일 **NUMA** 노드만 지정합니다.

추가 리소스

- **systemd.resource-control(5), systemd.exec(5), and set_mempolicy(2) manual pages**

31.13. SYSTEMD-RUN 명령을 사용하여 일시적인 CGROUPS 생성

일시적인 **cgroups** 는 런타임 중 단위(서비스 또는 범위)에서 사용하는 리소스에 대한 제한을 설정합니다.

절차

- 일시적인 제어 그룹을 생성하려면 다음 형식으로 **systemd-run** 명령을 사용합니다.

```
# systemd-run --unit=<name> --slice=<name>.slice <command>
```

이 명령은 일시적인 서비스 또는 범위 장치를 생성 및 시작하고 이러한 단위로 사용자 지정 명령을 실행합니다.

  - **--unit=<name& gt;** 옵션은 유닛 이름을 제공합니다. **--unit** 을 지정하지 않으면 이름이 자동으로 생성됩니다.
  - **--slice=<name>.slice** 옵션을 사용하면 서비스 또는 범위 단위를 지정된 슬라이스의 멤버로 만들 수 있습니다. **< name>.slice** 를 기존 슬라이스 이름(예: **systemctl -t slice**)의 이름으로 바꾸거나 고유한 이름을 전달하여 새 슬라이스를 생성합니다. 기본적으로 서비스 및 범위는 **system.slice** 의 멤버로 생성됩니다.

- & lt;command >를 서비스 또는 범위 단위로 실행하려는 명령으로 바꿉니다.

다음 메시지가 표시되어 서비스 또는 범위를 성공적으로 생성 및 시작했는지 확인합니다.

```
# Running as unit <name>.service
```

- 필요한 경우 프로세스가 완료된 후 단위를 실행하여 런타임 정보를 수집할 수 있습니다.

```
# systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>
```

이 명령은 일시적인 서비스 장치를 생성 및 시작하고 이러한 장치에서 사용자 지정 명령을 실행합니다. **--remain-after-exit** 옵션을 사용하면 프로세스가 완료된 후 서비스가 계속 실행되도록 합니다.

추가 리소스

- 제어 그룹 이해
- RHEL에서 기본 시스템 설정 구성
- **systemd-run(1)** 매뉴얼 페이지

31.14. 일시적인 제어 그룹 제거

systemd 시스템 및 서비스 관리자를 사용하여 더 이상 제한, 우선 순위 또는 프로세스 그룹의 하드웨어 리소스에 대한 액세스를 제어할 필요가 없는 경우**cgroup**(임시 제어 그룹)을 제거할 수 있습니다.

서비스 또는 범위 단위에 포함된 모든 프로세스가 완료되면 임시 **cgroup** 이 자동으로 해제됩니다.

절차

- 모든 프로세스가 있는 서비스 장치를 중지하려면 다음을 실행합니다.

```
# systemctl stop name.service
```

- 단위 프로세스 중 하나 이상을 종료하려면 다음을 실행합니다.

```
# systemctl kill name.service --kill-who=PID,... --signal=<signal>
```

위의 명령은 **--kill-who** 옵션을 사용하여 종료하려는 제어 그룹에서 **process(es)**를 선택합니다. 여러 프로세스를 동시에 종료하려면 쉼표로 구분된 **PID** 목록을 전달합니다. **--signal** 옵션은 지정된 프로세스에 보낼 **POSIX** 신호 유형을 결정합니다. 기본 신호는 **SIGTERM** 입니다.

추가 리소스

- 제어 그룹 이해
- 커널 리소스 컨트롤러란?
- **systemd.resource-control(5), cgroups(7) manual pages**
- 제어 그룹에서 **systemd**의 역할
- **RHEL**에서 기본 시스템 설정 구성

32장. CGROUPS 이해

제어 그룹 (cgroups) 커널 기능을 사용하여 제한, 우선 순위 또는 프로세스의 하드웨어 리소스를 격리할 수 있습니다. 이를 통해 애플리케이션의 리소스 사용량을 세부적으로 제어하여 보다 효율적으로 활용할 수 있습니다.

32.1. 제어 그룹 이해

제어 그룹은 프로세스를 계층적으로 정렬된 그룹(cgroups)으로 구성할 수 있는 Linux 커널 기능입니다. 계층 구조(제어 그룹 트리)는 기본적으로 /sys/fs/cgroup/ 디렉터리에 마운트된 cgroups 가상 파일 시스템에 구조를 제공하여 정의됩니다. systemd 시스템 및 서비스 관리자는 cgroups 를 사용하여 관리하는 모든 단위 및 서비스를 구성합니다. 또는 /sys/fs/cgroup/ 디렉터리에서 하위 디렉터리를 생성하고 제거하여 cgroup 계층을 수동으로 관리할 수 있습니다.

그런 다음 리소스 컨트롤러(커널 구성 요소)는 시스템 리소스(예: CPU 시간, 메모리, 네트워크 대역폭 또는 다양한 조합)의 제한, 우선 순위를 지정 또는 할당하여 cgroup 의 프로세스 동작을 수정합니다.

cgroup 의 부가된 값은 프로세스 집계로, 애플리케이션 및 사용자 간 하드웨어 리소스 분할을 가능하게 합니다. 따라서 사용자 환경의 전체 효율성, 안정성 및 보안이 향상될 수 있습니다.

제어 그룹 버전 1

제어 그룹 버전 1 (cgroups-v1)은 리소스별 컨트롤러 계층을 제공합니다. 즉, 각 리소스(예: CPU, 메모리, I/O 등)에는 자체 제어 그룹 계층 구조가 있습니다. 한 컨트롤러가 각각의 리소스를 관리하는 데 다른 컨트롤러와 조정할 수 있는 방식으로 다양한 컨트롤 그룹 계층 구조를 결합할 수 있습니다. 그러나 두 컨트롤러는 서로 다른 프로세스 계층에 속할 수 있으므로 적절한 조정을 허용하지 않습니다.

cgroups-v1 컨트롤러는 대규모 기간 동안 개발되었으며 그 결과 제어 파일의 동작 및 이름 지정은 균일하지 않습니다.

제어 그룹 버전 2

계층 유연성에서 발생하는 컨트롤러 조정의 문제로 인해 제어 그룹 버전 2가 개발되었습니다.

제어 그룹 버전 2 (cgroups-v2)는 모든 리소스 컨트롤러가 마운트된 단일 제어 그룹 계층 구조를 제공합니다.

제어 파일 동작 및 이름 지정은 다양한 컨트롤러에서 일관되게 지정됩니다.

중요

RHEL 9에서는 기본적으로 cgroups-v2 를 마운트하고 사용합니다.

이 하위 섹션은 Devconf.cz 2019 프레젠테이션을 기반으로 했습니다.[2]

추가 리소스

- 커널 리소스 컨트롤러란?
- cgroups ImageCon 매뉴얼 페이지
- cgroups-v1
- cgroups-v2

32.2. 커널 리소스 컨트롤러란?

제어 그룹의 기능은 커널 리소스 컨트롤러에서 사용할 수 있습니다. RHEL 9는 제어 그룹 버전 1(cgroups-v1) 및 제어 그룹 버전 2(cgroups-v2)에 대한 다양한 컨트롤러를 지원합니다.

제어 그룹 하위 시스템이라고도 하는 리소스 컨트롤러는 CPU 시간, 메모리, 네트워크 대역폭 또는 디스크 I/O와 같은 단일 리소스를 나타내는 커널 하위 시스템입니다. Linux 커널은 systemd 시스템 및 서비스 관리자에 의해 자동으로 마운트되는 다양한 리소스 컨트롤러를 제공합니다. /proc/cgroups 파일에서 현재 마운트된 리소스 컨트롤러 목록을 찾습니다.

cgroup-v1 에서는 다음 컨트롤러를 사용할 수 있습니다.

- blkio - 블록 장치로의 입력/출력 액세스에 대한 제한을 설정할 수 있습니다.
- cpu - 제어 그룹 작업에 대해 CFS(Completely Fair Scheduler) 스케줄러의 매개 변수를 조정할 수 있습니다. 동일한 마운트에서 cpuacct 컨트롤러와 함께 마운트됩니다.

- *cpuacct - 제어 그룹의 작업에서 사용하는 CPU 리소스에 대한 자동 보고서를 생성합니다. 동일한 마운트의 cpu 컨트롤러와 함께 마운트됩니다.*

- *cpuset - 지정된 CPU 서브 세트에서만 실행되도록 제어 그룹 작업을 제한하고 작업을 지정된 메모리 노드에서만 사용하도록 지시하는 데 사용할 수 있습니다.*

- *devices - 제어 그룹의 작업에 대한 장치에 대한 액세스를 제어할 수 있습니다.*

- *freezer - 제어 그룹의 작업을 일시 중지하거나 재개하는 데 사용할 수 있습니다.*

- *memory - 제어 그룹의 작업에서 메모리 사용 제한을 설정하는 데 사용할 수 있으며 해당 작업에서 사용하는 메모리 리소스에 대한 자동 보고서를 생성할 수 있습니다.*

- *net_cls - Linux 트래픽 컨트롤러(tc 명령)가 특정 제어 그룹 작업에서 시작된 패킷을 식별하는 클래스 식별자(classid)로 네트워크 패킷 태그를 지정합니다. net_cls, net_filter (iptables)의 하위 시스템에서 이 태그를 사용하여 이러한 패킷에서 작업을 수행할 수도 있습니다. net_filter 는 Linux 방화벽(iptables 명령)을 통해 특정 제어 그룹 작업에서 시작되는 패킷을 식별할 수 있는 방화벽 식별자(fwid)로 네트워크 소켓을 태그합니다.*

- *net_prio - 네트워크 트래픽의 우선 순위를 설정합니다.*

- *PIDs - 제어 그룹의 여러 프로세스와 해당 하위 항목에 대한 제한을 설정할 수 있습니다.*

- *perf _event - 성능 모니터링 및 보고 유틸리티를 통한 모니터링 작업을 그룹화할 수 있습니다.*

- *RDMA - 제어 그룹에서 원격 직접 메모리 액세스/InfiniBand 특정 리소스에 대한 제한을 설정할 수 있습니다.*

- *hugetlb - 대규모 크기 가상 메모리 페이지의 사용을 제어 그룹의 작업으로 제한하는 데 사용할 수 있습니다.*

cgroups-v2 에서는 다음 컨트롤러를 사용할 수 있습니다.

- *io - cgroups-v1 의 blkio 에 대한 후속 조치입니다.*
- *memory - cgroups-v1 의 메모리 후속입니다.*
- *PIDs - cgroups-v 1 의 pid와 동일합니다.*
- *RDMA - cgroups-v1 의 rdma 와 동일합니다.*
- *cpu - cgroups-v1 의 cpu 및 cpuacct 에 대한 후속입니다.*
- *cpuset - 새 파티션 기능을 사용하는 core 기능(cpus{,.effective}, mems{,.effective})만 지원합니다.*
- *perf_event - 지원은 고유하고 명시적인 제어 파일이 없습니다. 해당 cgroup 내의 모든 작업을 프로파일링하는 perf 명령의 매개 변수로 v2 cgroup 을 지정할 수 있습니다.*

중요

리소스 컨트롤러는 cgroup-v1 계층 구조 또는 두 계층에서 동시에 사용할 수 없는 cgroups-v2 계층에서 사용할 수 있습니다.

추가 리소스

- *cgroups ImageCon 매뉴얼 페이지*
- */usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/ 디렉토리(kernel-doc 패키지를 설치한 후).*

32.3. 네임스페이스란?

네임스페이스는 소프트웨어 오브젝트를 구성하고 식별하는 데 가장 중요한 방법 중 하나입니다.

네임스페이스는 글로벌 리소스의 자체 격리된 인스턴스가 있는 네임스페이스 내의 프로세스에 표시되는 추상화에서 글로벌 시스템 리소스(예: 마운트 지점, 네트워크 장치 또는 호스트 이름)를 래핑합니다. 네임스페이스를 사용하는 가장 일반적인 기술 중 하나는 컨테이너입니다.

특정 글로벌 리소스에 대한 변경 사항은 해당 네임스페이스의 프로세스에만 표시되고 나머지 시스템 또는 기타 네임 스페이스에는 영향을 미치지 않습니다.

프로세스가 속한 네임스페이스를 검사하려면 /proc/<PID>/ns/ 디렉터리에서 심볼릭 링크를 확인할 수 있습니다.

다음 표에서는 격리하는 지원되는 네임스페이스 및 리소스를 보여줍니다.

네임스페이스	isolates
Mount	마운트 지점
UTS	호스트 이름 및 NIS 도메인 이름
IPC	System V IPC, POSIX 메시지 대기열
PID	프로세스 ID
네트워크	네트워크 장치, 스택, 포트 등
사용자	사용자 및 그룹 ID
제어 그룹	제어 그룹 루트 디렉터리

추가 리소스

- *namespaces ImageCon 및 cgroup_namespaces442) 매뉴얼 페이지*
- *제어 그룹 이해*

---

[2]
Linux Control Group v2 - Waiman Long의 Introduction, Devconf.cz 2019 프레젠테이션

33장. ZSWAP으로 시스템 성능 개선

zswap 커널 기능을 활성화하여 시스템 성능을 향상시킬 수 있습니다.

33.1. ZSWAP이란 무엇입니까?

이 섹션에서는 zswap 이 무엇인지, 시스템 성능 향상을 초래할 수 있는 방법을 설명합니다.

zswap 은 스왑 페이지의 압축된 RAM 캐시를 제공하는 커널 기능입니다. 메커니즘은 다음과 같이 작동합니다. zswap 은 스왑 아웃되는 프로세스에 있는 페이지를 가져와 동적으로 할당된 RAM 기반 메모리 풀에 압축하려고 합니다. 풀이 가득 차거나 RAM이 소진되면 zswap 은 LRU 기반(최근 사용된)에서 백업 스왑 장치에 대한 압축된 캐시에서 페이지를 제거합니다. 페이지의 스왑 캐시의 압축을 풀면 zswap 은 풀의 압축된 버전을 확보합니다.

zswap의 이점

- 상당한 I/O 감소
- 워크로드 성능 개선

Red Hat Enterprise Linux 9에서 zswap 은 기본적으로 활성화되어 있습니다.

추가 리소스

- Zswap이란 무엇입니까?

33.2. 런타임 시 ZSWAP 활성화

sysfs 인터페이스를 사용하여 시스템 런타임에서 zswap 기능을 활성화할 수 있습니다.

사전 요구 사항

- 루트 권한이 있어야 합니다.

절차

- *zswap 활성화:*

```
# echo 1 > /sys/module/zswap/parameters/enabled
```

검증 단계

- *zswap 이 활성화되었는지 확인합니다.*

```
# grep -r . /sys/kernel/debug/zswap

duplicate_entry:0
pool_limit_hit:13422200
pool_total_size:6184960 (pool size in total in pages)
reject_alloc_fail:5
reject_compress_poor:0
reject_kmemcache_fail:0
reject_reclaim_fail:13422200
stored_pages:4251 (pool size after compression)
written_back_pages:0
```

추가 리소스

- *Zswap 기능을 활성화하는 방법은 무엇입니까?*

33.3. ZSWAP을 영구적으로 활성화

zswap.enabled=1 커널 명령줄 매개변수를 제공하여 zswap 기능을 영구적으로 활성화할 수 있습니다.

사전 요구 사항

- *루트 권한이 있어야 합니다.*
- *grubby 또는 zipl 유틸리티가 시스템에 설치되어 있습니다.*

절차

1. *zswap 을 영구적으로 활성화합니다.*

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="zswap.enabled=1"
```

2. 시스템을 재부팅하여 변경 사항을 적용합니다.

검증 단계

- zswap 이 활성화되었는지 확인합니다.

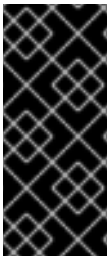
```
# cat /proc/cmdline

BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.14.0-70.5.1.el9_0.x86_64
root=/dev/mapper/rhel-root ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M
resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet
zswap.enabled=1
```

추가 리소스

- Zswap 기능을 활성화하는 방법은 무엇입니까?
- 커널 명령줄 매개변수 구성

34장. CGROUP을 사용하여 CGROUP 수동 관리

cgroup fs 가상 파일 시스템에 디렉터리를 생성하여 시스템의 cgroup 계층 구조를 관리할 수 있습니다. 파일 시스템은 기본적으로 /sys/fs/cgroup/ 디렉터리에 마운트되며 전용 제어 파일에 원하는 구성을 지정할 수 있습니다.



중요

일반적으로 Red Hat은 systemd 를 사용하여 시스템 리소스 사용을 제어하는 것이 좋습니다. 특수한 경우에만 cgroup 가상 파일 시스템을 수동으로 구성해야 합니다. 예를 들어 cgroup-v2 계층 구조에는 동일하지 않은 cgroup-v1 컨트롤러를 사용해야 합니다.

34.1. CGROUP 생성 및 CGROUPS-V2 파일 시스템에서 컨트롤러 활성화

디렉터리를 생성하거나 제거하고cgroups가상 파일 시스템의 파일에 작성하여 제어 그룹 (cgroups) 을 관리할 수 있습니다. 파일 시스템은 기본적으로 /sys/fs/cgroup/ 디렉터리에 마운트됩니다. cgroups 컨트롤러의 설정을 사용하려면 하위 cgroup 에 대해 원하는 컨트롤러도 활성화해야 합니다. 루트 cgroup 은 기본적으로 하위 cgroup 에 대해 메모리 및 pids 컨트롤러를 활성화했습니다. 따라서 Red Hat은 /sys/fs/cgroup/ root cgroup 에 두 개 이상의 하위 cgroup 을 생성하는 것이 좋습니다. 이렇게 하면 선택적으로 자식 cgroups 에서 메모리 및 pids 컨트롤러를 제거하고 cgroup 파일의 조직적인 명확성을 개선할 수 있습니다.

사전 요구 사항

- 루트 권한이 있어야 합니다.

절차

1. /sys/fs/cgroup/Example/ 디렉터리를 생성합니다.

```
# mkdir /sys/fs/cgroup/Example/
```

/sys/fs/cgroup/Example/ 디렉터리는 하위 그룹을 정의합니다. /sys/fs/cgroup/Example/ 디렉토리를 생성하면 일부 cgroups-v2 인터페이스 파일이 디렉터리에 자동으로 생성됩니다. /sys/fs/cgroup/Example/ 디렉터리에는 메모리 및 pids 컨트롤러의 컨트롤러별 파일도 포함되어 있습니다.

2. 선택적으로 새로 생성된 하위 제어 그룹을 검사합니다.

```
# ll /sys/fs/cgroup/Example/
```

```
-r—r—r--. 1 root root 0 Jun  1 10:33 cgroup.controllers
-r—r—r--. 1 root root 0 Jun  1 10:33 cgroup.events
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.freeze
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.procs
…
-rw-r—r--. 1 root root 0 Jun  1 10:33 cgroup.subtree_control
-r—r—r--. 1 root root 0 Jun  1 10:33 memory.events.local
-rw-r—r--. 1 root root 0 Jun  1 10:33 memory.high
-rw-r—r--. 1 root root 0 Jun  1 10:33 memory.low
…
-r—r—r--. 1 root root 0 Jun  1 10:33 pids.current
-r—r—r--. 1 root root 0 Jun  1 10:33 pids.events
-rw-r—r--. 1 root root 0 Jun  1 10:33 pids.max
```

*예제 출력에서는 **cgroup.procs** 또는 **cgroup.controllers** 와 같은 일반적인 **cgroup** 제어 인터페이스 파일을 보여줍니다. 이러한 파일은 활성화된 컨트롤러에 관계없이 모든 제어 그룹에 공통입니다.*

***memory.high** 및 **pids.max** 와 같은 파일은 루트 제어 그룹(**/sys/fs/cgroup/**)에 있는 메모리 및 **pids** 컨트롤러와 관련되어 있으며 **systemd** 에 의해 기본적으로 활성화됩니다.*

*기본적으로 새로 생성된 하위 그룹은 상위 **cgroup** 의 모든 설정을 상속합니다. 이 경우 **root cgroup** 에는 제한이 없습니다.*

3. *`/sys/fs/cgroup/cgroup.controllers` 파일에서 원하는 컨트롤러를 사용할 수 있는지 확인합니다.*

```
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

4. *원하는 컨트롤러를 활성화합니다. 이 예제에서는 **cpu** 및 **cpuset** 컨트롤러입니다.*

```
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

*이러한 명령을 사용하면 **/sys/fs/cgroup/** **root** 제어 그룹의 즉시 하위 그룹에 대해 **cpu** 및 **cpuset** 컨트롤러가 활성화됩니다. 새로 생성된 **Example** 제어 그룹 포함. 하위 그룹은 프로세스를 지정하고 기준에 따라 각 프로세스에 제어 검사를 적용할 수 있는 곳입니다.*

*사용자는 임의 수준에서 **cgroup.subtree_control** 파일의 내용을 읽고 즉시 하위 그룹에서 사용할 수 있는 컨트롤러를 확인할 수 있습니다.*

참고

*기본적으로 루트 제어 그룹의 **/sys/fs/cgroup/cgroup.subtree_control** 파일에는 메모리 및 **pid** 컨트롤러가 포함되어 있습니다.*

5. **Example** *제어 그룹의 하위* **cgroup** *에 대해 원하는 컨트롤러를 활성화합니다.*

```
# echo "+cpu +cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
```

이 명령을 사용하면 즉시 하위 제어 그룹에 메모리 또는 **pids** *컨트롤러가 아닌* **CPU** *시간 분배를 규제하는 데 관련된 컨트롤러 만 있습니다.*

6. **/sys/fs/cgroup/Example/tasks/** *디렉터리를 생성합니다.*

```
# mkdir /sys/fs/cgroup/Example/tasks/
```

/sys/fs/cgroup/Example/tasks/ *디렉터리는* **cpu** *및* **cpuset** *컨트롤러와 관련된 파일이 있는 하위 그룹을 정의합니다. 이제 프로세스를 이 제어 그룹에 할당하고 프로세스에* **cpu** *및* **cpuset** *컨트롤러 옵션을 사용할 수 있습니다.*

7. *필요한 경우 하위 제어 그룹을 검사합니다.*

```
# ll /sys/fs/cgroup/Example/tasks
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.controllers
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.events
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.freeze
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.max.depth
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.max.descendants
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.procs
-r—r—r--. 1 root root 0 Jun  1 11:45 cgroup.stat
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.subtree_control
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.threads
-rw-r—r--. 1 root root 0 Jun  1 11:45 cgroup.type
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.max
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.pressure
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus
-r—r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.effective
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.partition
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpuset.mems
-r—r—r--. 1 root root 0 Jun  1 11:45 cpuset.mems.effective
-r—r—r--. 1 root root 0 Jun  1 11:45 cpu.stat
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.weight
```

```
-rw-r—r--. 1 root root 0 Jun  1 11:45 cpu.weight.nice
-rw-r—r--. 1 root root 0 Jun  1 11:45 io.pressure
-rw-r—r--. 1 root root 0 Jun  1 11:45 memory.pressure
```

중요

cpu 컨트롤러는 관련 하위 제어 그룹에 단일 CPU의 시간 동안 경쟁하는 최소 2개의 프로세스가 있는 경우에만 활성화됩니다.

검증 단계

- 선택 사항: 원하는 컨트롤러가 활성화된 새 cgroup 을 생성했는지 확인합니다.

```
# cat /sys/fs/cgroup/Example/tasks/cgroup.controllers
cpuset cpu
```

추가 리소스

- 제어 그룹 이해
- 커널 리소스 컨트롤러란?
- cgroups-v1 마운트
- cgroups ImageCon, sysfs(5) 매뉴얼 페이지

34.2. CPU 가중치를 조정하여 애플리케이션의 CPU 시간 분배 제어

특정 cgroup 트리의 애플리케이션에 CPU 시간 배포를 규제하려면 cpu 컨트롤러의 관련 파일에 값을 할당해야 합니다.

사전 요구 사항

- 루트 권한이 있어야 합니다.
- CPU 시간 분배를 제어하려는 애플리케이션이 있습니다.

- 다음 예제와 같이 **/sys/fs/cgroup/ root** 제어 그룹 내에 하위 제어 그룹 의 두 가지 수준 계층을 생성했습니다.

```
...
├── Example
│   ├── g1
│   ├── g2
│   └── g3
...
```

- **cgroup** 생성에 설명된 대로 상위 제어 그룹 및 하위 제어 그룹에서 **cpu** 컨트롤러를 활성화하여 **cgroups-v2** 파일 시스템에서 컨트롤러를 활성화합니다.

절차

1. 제어 그룹 내에서 리소스 제한을 달성하도록 원하는 **CPU** 가중치를 구성합니다.

```
# echo "150" > /sys/fs/cgroup/Example/g1/cpu.weight
# echo "100" > /sys/fs/cgroup/Example/g2/cpu.weight
# echo "50" > /sys/fs/cgroup/Example/g3/cpu.weight
```

2. 애플리케이션의 **PID**를 **g1,g2, g3** 하위 그룹에 추가합니다.

```
# echo "33373" > /sys/fs/cgroup/Example/g1/cgroup.procs
# echo "33374" > /sys/fs/cgroup/Example/g2/cgroup.procs
# echo "33377" > /sys/fs/cgroup/Example/g3/cgroup.procs
```

예제 명령을 사용하면 원하는 애플리케이션이 **Example/g*/** 하위 **cgroup**의 멤버가 되고 해당 **cgroups**의 구성에 따라 **CPU** 시간을 분산시킵니다.

프로세스를 실행 중인 하위 **cgroup(g1,g2,g3)**의 가중치는 상위 **cgroup** 수준(예)으로 요약됩니다. 그런 다음 **CPU** 리소스는 각각의 가중치에 따라 비례분산됩니다.

결과적으로 모든 프로세스가 동시에 실행될 때 커널은 각각의 **cgroup**의 **cpu.weight** 파일에 따라 비례 **CPU** 시간을 할당합니다.

하위 cgroup	**cpu.weight** file	CPU 시간 할당
g1	150	~50% (150/300)

하위 cgroup	**cpu.weight** file	CPU 시간 할당
g2	100	~33% (100/300)
g3	50	~16% (50/300)

***cpu.weight** 컨트롤러 파일의 값은 백분율이 아닙니다.*

*한 프로세스가 중지되어 실행 중인 프로세스가 없는 **cgroup g2** 를 그대로 두면 계산으로 **cgroup g2** 및 **g3** 의 계정 가중치만 생략됩니다.*

하위 cgroup	**cpu.weight** file	CPU 시간 할당
g1	150	~75% (150/200)
g3	50	~25% (50/200)

중요

*하위 **cgroup**에 실행 중인 프로세스가 여러 개 있는 경우 해당 **cgroup**에 할당된 **CPU** 시간이 해당 **cgroup**의 멤버 프로세스에 동일하게 배포됩니다.*

검증

1. *애플리케이션이 지정된 제어 그룹에서 실행되는지 확인합니다.*

```
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

*명령 출력에는 **Example/g*/** 하위 **cgroups**에서 실행되는 지정된 애플리케이션의 프로세스가 표시됩니다.*

2. *스로틀된 애플리케이션의 현재 **CPU** 사용량을 검사합니다.*

```
# top
top - 05:17:18 up 1 day, 18:25,  1 user,  load average: 3.03, 3.03, 3.00
```

```
Tasks:  95 total,   4 running,  91 sleeping,   0 stopped,   0 zombie
%Cpu(s): 18.1 us, 81.6 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
MiB Mem :   3737.0 total,   3233.7 free,    132.8 used,    370.5 buff/cache
MiB Swap:   4060.0 total,   4060.0 free,      0.0 used.   3373.1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
33373 root      20   0   18720   1748   1460 R  49.5   0.0 415:05.87 sha1sum
33374 root      20   0   18720   1756   1464 R  32.9   0.0 412:58.33 sha1sum
33377 root      20   0   18720   1860   1568 R  16.3   0.0 411:03.12 sha1sum
  760 root      20   0  416620  28540  15296 S   0.3   0.7   0:10.23 tuned
    1 root      20   0  186328  14108   9484 S   0.0   0.4   0:02.00 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthread
...
```

참고

더 명확한 설명을 위해 모든 예제 프로세스를 단일 CPU에서 실행하도록 강제했습니다. CPU weight는 여러 CPU에서 사용되는 경우에도 동일한 원칙을 적용합니다.

PID 33373,PID 33374, PID 33377 의 CPU 리소스가 각 하위 cgroup에 할당한 가중치인 150, 100, 50을 기반으로 할당되었습니다. 가중치는 약 50%, 33%, 각 애플리케이션에 대한 CPU 시간이 16%에 해당합니다.

추가 리소스

- 제어 그룹 이해
- 커널 리소스 컨트롤러란?
- cgroup 생성 및 cgroups-v2 파일 시스템에서 컨트롤러 활성화
- 리소스 배포 모델
- cgroups ImageCon, sysfs(5) 매뉴얼 페이지

34.3. CGROUPS-V1 마운트

부팅 프로세스 중에 RHEL 9는 기본적으로 cgroup-v2 가상 파일 시스템을 마운트합니다. 애플리케이

션의 리소스를 제한하면서 **cgroup-v1** 기능을 활용하려면 시스템을 수동으로 구성합니다.

참고

cgroup-v1 및 **cgroup-v2** 는 모두 커널에서 완전히 활성화됩니다. 커널 보기에는 기본 제어 그룹 버전이 없으며 시작 시 마운트할 **systemd** 에 의해 결정됩니다.

사전 요구 사항

- 루트 권한이 있어야 합니다.

절차

1. 시스템 부팅 중에 **systemd** 시스템 및 서비스 관리자가 기본적으로 **cgroup-v1** 을 마운트하도록 시스템을 구성합니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

그러면 현재 부팅 항목에 필요한 커널 명령줄 매개변수가 추가됩니다.

모든 커널 부팅 항목에 동일한 매개변수를 추가하려면 다음을 수행합니다.

```
# grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

2. 시스템을 재부팅하여 변경 사항을 적용합니다.

검증

1. 필요한 경우 **cgroups-v1** 파일 시스템이 마운트되었는지 확인합니다.

```
# mount -l | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs
(ro,nosuid,nodev,noexec,seclabel,size=4096k,nr_inodes=1024,mode=755,inode64)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/syste
md-cgroups-agent,name=systemd)
```

```
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/pids type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,pids)
cgroup on /sys/fs/cgroup/cpuset type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpuset)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,hugetlb)
cgroup on /sys/fs/cgroup/memory type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,memory)
cgroup on /sys/fs/cgroup/blkio type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,blkio)
cgroup on /sys/fs/cgroup/devices type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,devices)
cgroup on /sys/fs/cgroup/misc type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,misc)
cgroup on /sys/fs/cgroup/freezer type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,freezer)
cgroup on /sys/fs/cgroup/rdma type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,rdma)
```

*다양한 **cgroup-v1** 컨트롤러에 해당하는 **cgroups-v1** 파일 시스템이 **/sys/fs/cgroup/** 디렉터리에 성공적으로 마운트되었습니다.*

2. *선택적으로 **/sys/fs/cgroup/** 디렉터리의 콘텐츠를 검사합니다.*

```
# ll /sys/fs/cgroup/
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 blkio
lrwxrwxrwx.  1 root root 11 Mar 16 09:34 cpu → cpu,cpuacct
lrwxrwxrwx.  1 root root 11 Mar 16 09:34 cpuacct → cpu,cpuacct
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 cpu,cpuacct
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 cpuset
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 devices
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 freezer
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 hugetlb
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 memory
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 misc
lrwxrwxrwx.  1 root root 16 Mar 16 09:34 net_cls → net_cls,net_prio
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 net_cls,net_prio
lrwxrwxrwx.  1 root root 16 Mar 16 09:34 net_prio → net_cls,net_prio
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 perf_event
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 pids
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 rdma
dr-xr-xr-x. 11 root root  0 Mar 16 09:34 systemd
```

*/sys/fs/cgroup/ 디렉터리(기본적으로 루트 제어 그룹 라고도 함)에는 **cpuset** 와 같은 컨트롤러별 디렉터리가 포함되어 있습니다. 또한 **systemd** 와 관련된 일부 디렉터리도 있습니다.*

추가 리소스

- 제어 그룹 이해
- 커널 리소스 컨트롤러란?
- cgroups ImageCon, sysfs(5) 매뉴얼 페이지
- RHEL 9에서 기본적으로 cgroup-v2 활성화

34.4. CGROUPS-V1을 사용하여 애플리케이션에 CPU 제한 설정

경우에 따라 애플리케이션이 많은 CPU 시간을 소비하므로 환경의 전반적인 상태에 부정적인 영향을 미칠 수 있습니다. /sys/fs/ 가상 파일 시스템을 사용하여 제어 그룹 버전 1 (cgroups-v1)을 사용하여 애플리케이션에 대한 CPU 제한을 구성합니다.

사전 요구 사항

- 루트 권한이 있어야 합니다.
- CPU 사용량이 제한하려는 애플리케이션이 있습니다.
- 시스템 부팅 중에 systemd 시스템 및 서비스 관리자가 기본적으로 cgroup-v1 을 마운트하도록 시스템을 구성하셨습니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

그러면 현재 부팅 항목에 필요한 커널 명령줄 매개변수가 추가됩니다.

절차

1.

CPU 소비에서 제한하려는 애플리케이션의 PID(프로세스 ID)를 확인합니다.

```
# top
top - 11:34:09 up 11 min,  1 user,  load average: 0.51, 0.27, 0.22
Tasks: 267 total,   3 running, 264 sleeping,   0 stopped,   0 zombie
%Cpu(s): 49.0 us,  3.3 sy,  0.0 ni, 47.5 id,  0.0 wa,  0.2 hi,  0.0 si,  0.0 st
MiB Mem :   1826.8 total,    303.4 free,   1046.8 used,    476.5 buff/cache
MiB Swap:   1536.0 total,   1396.0 free,    140.0 used.    616.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 6955 root      20   0  228440   1752   1472 R  99.3   0.1   0:32.71 sha1sum
 5760 jdoe      20   0 3603868 205188  64196 S   3.7  11.0   0:17.19 gnome-shell
 6448 jdoe      20   0  743648  30640  19488 S   0.7   1.6   0:02.73 gnome-terminal-
    1 root      20   0  245300   6568   4116 S   0.3   0.4   0:01.87 systemd
  505 root      20   0       0      0      0 I   0.3   0.0   0:00.75 kworker/u4:4-events_unbound
...
```

top 프로그램의 예제 출력은 PID 6955 (실용 애플리케이션 sha1sum)가 CPU 리소스를 많이 사용한다는 것을 보여줍니다.

2.

cpu 리소스 컨트롤러 디렉터리에 하위 디렉터리를 생성합니다.

```
# mkdir /sys/fs/cgroup/cpu/Example/
```

위의 디렉터리는 특정 프로세스를 배치하고 프로세스에 특정 CPU 제한을 적용할 수 있는 제어 그룹을 나타냅니다. 동시에 일부 cgroups-v1 인터페이스 파일과 cpu 컨트롤러별 파일이 디렉터리에 생성됩니다.

3.

선택적으로 새로 생성된 제어 그룹을 검사합니다.

```
# ll /sys/fs/cgroup/cpu/Example/
-rw-r—r--. 1 root root 0 Mar 11 11:42 cgroup.clone_children
-rw-r—r--. 1 root root 0 Mar 11 11:42 cgroup.procs
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.stat
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_all
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_sys
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_user
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_sys
-r—r—r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_user
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.cfs_period_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.cfs_quota_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.rt_period_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.rt_runtime_us
-rw-r—r--. 1 root root 0 Mar 11 11:42 cpu.shares
```

```
-r—r—r--. 1 root root 0 Mar 11 11:42 cpu.stat
-rw-r—r--. 1 root root 0 Mar 11 11:42 notify_on_release
-rw-r—r--. 1 root root 0 Mar 11 11:42 tasks
```

예제 출력에서는 **Example** 제어 그룹의 프로세스에 대해 설정할 수 있는 특정 구성 및/또는 제한을 나타내는 **cpuacct.usage,cpu.cfs._period_us** 와 같은 파일을 보여줍니다. 각 파일 이름 앞에는 자신이 속한 컨트롤 그룹 컨트롤러의 이름이 붙습니다.

기본적으로 새로 생성된 제어 그룹은 제한 없이 시스템의 전체 **CPU** 리소스에 대한 액세스를 상속합니다.

4. 제어 그룹에 대한 **CPU** 제한을 구성합니다.

```
# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
```

cpu.cfs_period_us 파일은 **CPU** 리소스에 대한 제어 그룹의 액세스 권한이 얼마나 자주 할당되어야 하는지에 대해 여기서 **"us")**로 표시되는 시간(마이크로초) 기간을 나타냅니다. 상한은 **1** 초이고 더 낮은 제한은 **1000**마이크로초입니다.

cpu.cfs_quota_us 파일은 한 기간 동안 모든 프로세스를 전체적으로 실행할 수 있는(**cpu.cfs_period_us)** 동안 모든 프로세스를 전체적으로 실행할 수 있는 마이크로초의 총 시간을 나타냅니다. 제어 그룹의 프로세스가 단일 기간 동안 할당량에 의해 지정된 모든 시간을 사용하는 즉시 나머지 기간 동안 제한되며 다음 기간까지 실행되지 않습니다. 더 낮은 제한은 **1000**마이크로초입니다.

위의 예제 명령은 예제 제어 그룹의 모든 프로세스가 **1**초(**cpu.cfs_period_us**로 정의됨)에서 **0.2**초 동안만 실행할 수 있도록 **CPU** 시간 제한을 설정합니다.

5. 필요한 경우 제한을 확인합니다.

```
# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
/sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
1000000
200000
```

6. 애플리케이션의 **PID**를 **Example** 제어 그룹에 추가합니다.

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs
```

```
or

# echo "6955" > /sys/fs/cgroup/cpu/Example/tasks
```

이전 명령은 원하는 애플리케이션이 **Example** 제어 그룹의 멤버가 되도록 하고 따라서 **Example** 제어 그룹에 대해 구성된 **CPU** 제한을 초과하지 않습니다. **PID**는 시스템의 기존 프로세스를 나타냅니다. 여기에서 **PID 6955** 는 **cpu** 컨트롤러의 사용 사례를 설명하는 데 사용되는 **sha1sum /dev/zero & amp;** 에게 할당되었습니다.

7. 애플리케이션이 지정된 제어 그룹에서 실행되는지 확인합니다.

```
# cat /proc/6955/cgroup
12:cpuset:/
11:hugetlb:/
10:net_cls,net_prio:/
9:memory:/user.slice/user-1000.slice/user@1000.service
8:devices:/user.slice
7:blkio:/
6:freezer:/
5:rdma:/
4:pids:/user.slice/user-1000.slice/user@1000.service
3:perf_event:/
2:cpu,cpuacct:/Example
1:name=systemd:/user.slice/user-1000.slice/user@1000.service/gnome-terminal-server.service
```

위의 예제 출력에서는 원하는 애플리케이션의 프로세스가 애플리케이션 프로세스에 **CPU** 제한을 적용하는 **Example** 제어 그룹에서 실행됨을 보여줍니다.

8. **throttled** 애플리케이션의 현재 **CPU** 사용량을 확인합니다.

```
# top
top - 12:28:42 up  1:06,  1 user,  load average: 1.02, 1.02, 1.00
Tasks: 266 total,   6 running, 260 sleeping,   0 stopped,   0 zombie
%Cpu(s): 11.0 us,  1.2 sy,  0.0 ni, 87.5 id,  0.0 wa,  0.2 hi,  0.0 si,  0.2 st
MiB Mem :   1826.8 total,    287.1 free,   1054.4 used,    485.3 buff/cache
MiB Swap:   1536.0 total,   1396.7 free,    139.2 used.    608.3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 6955 root      20   0  228440   1752   1472 R  20.6   0.1  47:11.43 sha1sum
 5760 jdoe      20   0 3604956 208832  65316 R   2.3  11.2   0:43.50 gnome-shell
 6448 jdoe      20   0  743836  31736  19488 S   0.7   1.7   0:08.25 gnome-terminal-
  505 root      20   0       0      0      0 I   0.3   0.0   0:03.39 kworker/u4:4-events_unbound
 4217 root      20   0   74192   1612   1320 S   0.3   0.1   0:01.19 spice-vdagentd
...
```

PID 6955 의 CPU 사용량이 99%에서 20%로 줄었습니다.

중요

cpu.cfs_period_us 및 cpu.cfs_quota_us 의 cgroup-v2 는 cpu.max 파일입니다. cpu.max 파일은 cpu 컨트롤러를 통해 사용할 수 있습니다.

추가 리소스

- *제어 그룹 이해*
- *어떤 커널 리소스 컨트롤러입니까?*
- *cgroups ImageCon, sysfs(5) 매뉴얼 페이지*

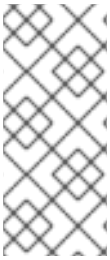
35장. BPF COMPILER COLLECTION을 사용하여 시스템 성능 분석

시스템 관리자는 BCC(BPF Compiler Collection) 라이브러리를 사용하여 Linux 운영 체제의 성능을 분석하고 정보를 수집하는 도구를 만들 수 있으며 다른 인터페이스를 통해 가져오기가 어려울 수 있습니다.

35.1. BCC 소개

BCC(BPF Compiler Collection)는 eBPF(extended Berkeley Packet Filter) 프로그램을 쉽게 생성할 수 있는 라이브러리입니다. eBPF 프로그램의 주요 유틸리티는 오버헤드 또는 보안 문제가 발생하지 않고 OS 성능 및 네트워크 성능을 분석하는 것입니다.

BCC는 사용자가 eBPF의 기술적 세부 사항을 알 필요가 없으며, 사전 생성된 eBPF 프로그램이 있는 bcc-tools 패키지와 같은 기본 시작점을 많이 제공합니다.



참고

eBPF 프로그램은 디스크 I/O, TCP 연결 및 프로세스 생성과 같은 이벤트에서 트리거됩니다. 프로그램에서 커널 충돌, 루프 또는 커널의 안전한 가상 시스템에서 실행되기 때문에 응답하지 않을 수 있습니다.

35.2. BCC-TOOLS 패키지 설치

이 섹션에서는 BCC(BPF Compiler Collection) 라이브러리를 종속성으로 설치하는 bcc-tools 패키지를 설치하는 방법을 설명합니다.

절차

1. bcc-tools 를 설치합니다.

```
# dnf install bcc-tools
```

BCC 툴은 /usr/share/bcc/tools/ 디렉터리에 설치됩니다.

2. 필요한 경우 툴을 검사합니다.

```
# ll /usr/share/bcc/tools/
```

```
...
-rwxr-xr-x. 1 root root  4198 Dec 14 17:53 dcsnoop
-rwxr-xr-x. 1 root root  3931 Dec 14 17:53 dcstat
-rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
-rw-r--r--. 1 root root  7105 Dec 14 17:53 deadlock_detector.c
drwxr-xr-x. 3 root root  8192 Mar 11 10:28 doc
-rwxr-xr-x. 1 root root  7588 Dec 14 17:53 execsnoop
-rwxr-xr-x. 1 root root  6373 Dec 14 17:53 ext4dist
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
...
```

위의 목록에 있는 **doc** 디렉토리에는 각 툴에 대한 문서가 포함되어 있습니다.

35.3. 성능 분석에 선택된 BCC-TOOLS 사용

이 섹션에서는 **BCC(BPF Compiler Collection)** 라이브러리에서 미리 생성된 특정 프로그램을 사용하여 이벤트별로 시스템 성능을 효율적이고 안전하게 분석하는 방법에 대해 설명합니다. **BCC** 라이브러리에서 미리 생성된 프로그램 세트는 추가 프로그램 생성의 예 역할을 할 수 있습니다.

사전 요구 사항

- 설치된 **bcc-tools** 패키지
- 루트 권한

execsnoop를 사용하여 시스템 프로세스 검사

1. 한 터미널에서 **execsnoop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/execsnoop
```

2. 다른 터미널 실행에서 예를 들면 다음과 같습니다.

```
$ ls /usr/share/bcc/tools/doc/
```

위 명령은 **ls** 명령의 수명이 짧은 프로세스를 생성합니다.

3. **execsnoop** 를 실행하는 터미널은 다음과 유사한 출력을 보여줍니다.

```
PCOMM PID    PPID   RET ARGS
ls    8382   8287     0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

execsnoop 프로그램은 시스템 리소스를 사용하는 새 프로세스마다 출력의 행을 출력합니다. **ls** 와 같이 매우 빠르게 실행되는 프로그램 프로세스도 감지하며 대부분의 모니터링 툴은 등록하지 않습니다.

execsnoop 출력에는 다음 필드가 표시됩니다.

- **PCOMM -** 상위 프로세스 이름. **(ls)**
- **PID -** 프로세스 **ID**입니다. **(8382)**
- **PPID -** 상위 프로세스 **ID**입니다. **(8287)**
- **RET -** 프로그램 코드를 새 프로세스에 로드하는 **exec()** 시스템 호출**(0)**의 반환 값입니다.
- **ARGS -** 인수가 포함된 시작 프로그램의 위치입니다.

execsnoop 에 대한 자세한 정보, 예제 및 옵션을 보려면 **/usr/share/bcc/tools/doc/execsnoop_example.txt** 파일을 참조하십시오.

exec() 에 대한 자세한 내용은 **exec(3)** 매뉴얼 페이지를 참조하십시오.

opennoop를 사용하여 명령에서 열리는 파일을 추적할 수 있습니다.

1. 한 터미널에서 **opensnoop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/opensnoop -n uname
```

위의 명령은 **uname** 명령 프로세스에서만 여는 파일의 출력을 출력합니다.

2. 다른 터미널에서 다음을 실행합니다.

```
$ uname
```

위의 명령은 특정 파일을 열고 다음 단계에서 캡처됩니다.

3. 실행 중인 터미널 은 다음과 유사한 출력을 보여줍니다.

```
PID    COMM  FD ERR PATH
8596   uname  3  0   /etc/ld.so.cache
8596   uname  3  0   /lib64/libc.so.6
8596   uname  3  0   /usr/lib/locale/locale-archive
...
```

opensnoop 프로그램은 전체 시스템에서 **open()** 시스템 호출을 감시하고 **uname** 이 방식을 따라 열려고 시도하는 각 파일의 출력 라인을 출력합니다.

opensnoop 출력에는 다음 필드가 표시됩니다.

- **PID -** 프로세스 **ID**입니다. **(8596)**
- **COMM -** 프로세스 이름**. (uname)**
- **FD - open()** 에서 열린 파일을 참조하기 위해 반환하는 값인 파일 설명자입니다. **(3)**
- **ERR -** 모든 오류
- **PATH - open()** 열기를 시도하는 파일의 위치입니다.

명령이 존재하지 않는 파일을 읽으려고 하면 **FD** 열은 **-1** 을 반환하고 **ERR** 열에는 관련 오류에 해당하는 값이 출력됩니다. 따라서 **opensnoop** 는 제대로 작동하지 않는 애플리케이션을 식별하는 데 도움이 될 수 있습니다.

opensnoop 에 대한 자세한 정보, 예제 및 옵션을 보려면

/usr/share/bcc/tools/doc/opensnoop_example.txt 파일을 참조하십시오.

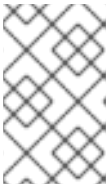
open() 에 대한 자세한 내용은 open(2) 매뉴얼 페이지를 참조하십시오.

BIOStop를 사용하여 디스크의 I/O 작업 검사

1. *한 터미널에서 biotop 프로그램을 실행합니다.*

```
# /usr/share/bcc/tools/biotop 30
```

명령을 사용하면 디스크에서 I/O 작업을 수행하는 최상위 프로세스를 모니터링할 수 있습니다. 인수를 사용하면 명령이 30초 요약을 생성합니다.



참고

인수를 제공하지 않으면 기본적으로 출력 화면이 1초마다 새로 고쳐집니다.

2. *다른 터미널 실행 예제에서는 다음과 같습니다.*

```
# dd if=/dev/vda of=/dev/zero
```

위의 명령은 로컬 하드 디스크 장치에서 콘텐츠를 읽고 출력을 /dev/zero 파일에 씁니다. 이 단계는 BIOS 를 설명하기 위해 특정 I/O 트래픽을 생성합니다.

3. *biotop 실행 중인 터미널은 다음과 유사한 출력을 보여줍니다.*

```
PID    COMM             D MAJ MIN DISK       I/O  Kbytes     AVGms
9568   dd               R 252 0   vda      16294 14440636.0  3.69
48     kswapd0          W 252 0   vda       1763 120696.0    1.65
7571   gnome-shell      R 252 0   vda        834 83612.0     0.33
1891   gnome-shell      R 252 0   vda       1379 19792.0     0.15
7515   Xorg             R 252 0   vda        280  9940.0     0.28
7579   llvmpipe-1       R 252 0   vda        228  6928.0     0.19
9515   gnome-control-c  R 252 0   vda         62  6444.0     0.43
8112   gnome-terminal-  R 252 0   vda         67  2572.0     1.54
7807   gnome-software   R 252 0   vda         31  2336.0     0.73
9578   awk              R 252 0   vda         17  2228.0     0.66
7578   llvmpipe-0       R 252 0   vda        156  2204.0     0.07
9581   pgrep            R 252 0   vda         58  1748.0     0.42
7531   InputThread      R 252 0   vda         30  1200.0     0.48
```

```
7504   gdbus          R 252 0   vda        3  1164.0     0.30
1983   llvmpipe-1     R 252 0   vda        39   724.0     0.08
1982   llvmpipe-0     R 252 0   vda        36   652.0     0.06
...
```

biotop 출력에는 다음 필드가 표시됩니다.

- **PID -** 프로세스 **ID**입니다. **(9568)**
- **COMM -** 프로세스 이름**(dd)**
- **DISK -** 읽기 작업을 수행하는 디스크입니다. **(vda)**
- **I/O -** 읽기 작업의 수입니다. **(16294)**
- **kbytes -** 읽기 작업에서 도달한 **Kbytes** 의 양입니다. **(14,440,636)**
- **AVGms -** 읽기 작업의 평균 **I/O** 시간입니다. **(3.69)**

biotop 에 대한 자세한 정보, 예제 및 옵션을 보려면 **/usr/share/bcc/tools/doc/biotop_example.txt** 파일을 참조하십시오.

dd 에 대한 자세한 내용은 **dd(1)** 매뉴얼 페이지를 참조하십시오.

xfsslower를 사용하여 예기치 않게 느린 파일 시스템 작업 노출

1. 한 터미널에서 **xfsslower** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/xfsslower 1
```

위의 명령은 **XFS** 파일 시스템이 읽기, 쓰기, 열기 또는 동기화**(fsync)** 작업을 수행하는 데 사용하는 시간을 측정합니다. **1** 인수는 프로그램이 **1ms**보다 느린 작업만 표시하도록 합니다.

참고

인수가 제공되지 않으면 **xfsslower** 는 기본적으로 **10ms**보다 느린 작업을 표시합니다.

2. 다른 터미널에서 다음을 실행합니다. 예를 들면 다음과 같습니다.

```
$ vim text
```

위의 명령은 **ovs** 편집기에 텍스트 파일을 생성하여 **XFS** 파일 시스템과의 특정 상호 작용을 시작합니다.

3. **xfsslower** 를 실행하는 터미널은 이전 단계에서 파일을 저장할 때 유사한 것을 보여줍니다.

```
TIME     COMM          PID    T BYTES   OFF_KB   LAT(ms) FILENAME
13:07:14 b'bash'       4754   R 256     0          7.11 b'vim'
13:07:14 b'vim'        4754   R 832     0          4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'        4754   R 32      20         1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'        4754   R 1982    0          2.30 b'vimrc'
13:07:14 b'vim'        4754   R 1393    0          2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'        4754   S 0       0          6.71 b'text'
13:07:45 b'pool'       2588   R 16      0          5.58 b'text'
...
```

위의 각 줄은 파일 시스템에서 작업을 나타내며 특정 임계값보다 더 많은 시간이 걸렸습니다. **xfsslower** 는 예기치 않은 느린 작업을 수행할 수 있는 파일 시스템 문제를 노출하는 데 유용합니다.

xfsslower 출력에는 다음 필드가 표시됩니다.

- **COMM -** 프로세스 이름. **(b'bash')**
- **t -** 작업 유형입니다. **(R)**
  - **Read**

- *Write*
- *Sync*

- *OFF_KB - KB의 파일 오프셋입니다. (0)*
- *파일 이름 - 읽기, 쓰기 또는 동기화되는 파일입니다.*

xfsslower 에 대한 세부 정보, 예제 및 옵션을 보려면 /usr/share/bcc/tools/doc/xfsslower_example.txt 파일을 참조하십시오.

fsync 에 대한 자세한 내용은 fsync(2) 매뉴얼 페이지를 참조하십시오.

36장. 대규모 페이지 구성

물리 메모리는 페이지라는 고정된 크기 청크로 관리됩니다. Red Hat Enterprise Linux 9에서 지원되는 x86_64 아키텍처에서 메모리 페이지의 기본 크기는 4KB 입니다. 이 기본 페이지 크기는 다양한 종류의 워크로드를 지원하는 Red Hat Enterprise Linux와 같은 범용 운영 체제에 적합합니다.

그러나 특정 애플리케이션에서는 특정 상황에서 더 큰 페이지 크기를 사용하는 것이 유용할 수 있습니다. 예를 들어, 수백 메가바이트 또는 수십 기가바이트의 대규모 및 비교적 고정된 데이터 세트에서 작동하는 애플리케이션은 4KB 페이지를 사용할 때 성능 문제가 발생할 수 있습니다. 이러한 데이터 세트에는 많은 양의 4KB 페이지가 필요할 수 있으므로 운영 체제 및 CPU의 오버헤드가 발생할 수 있습니다.

이 섹션에서는 RHEL 9에서 사용할 수 있는 대규모 페이지 및 구성 방법에 대한 정보를 제공합니다.

36.1. 사용 가능한 대규모 페이지 기능

Red Hat Enterprise Linux 9를 사용하면 빅 데이터 세트에서 작업하는 애플리케이션에 대규모 페이지를 사용하고 이러한 애플리케이션의 성능을 향상시킬 수 있습니다.

다음은 RHEL 9에서 지원되는 대규모 페이지 방법입니다.

HugeTLB 페이지

HugeTLB 페이지는 정적 대규모 페이지라고도 합니다. HugeTLB 페이지를 예약하는 방법에는 두 가지가 있습니다.

- 부팅 시: 메모리가 아직 크게 조각화되지 않았기 때문에 성공할 가능성이 높아집니다. 그러나 NUMA 시스템에서는 NUMA 노드 간에 페이지 수가 자동으로 분할됩니다. 부팅 시 HugeTLB 페이지 동작에 영향을 주는 매개변수에 대한 자세한 내용은 부팅 시 HugeTLB 페이지 예약 매개 변수 및 부팅 시 HugeTLB 페이지를 구성하는 방법을 참조하십시오.

- 런타임에는 NUMA 노드당 대규모 페이지를 예약할 수 있습니다. 부팅 프로세스에서 가능한 빨리 런타임 예약을 수행하면 메모리 조각화 가능성이 낮아집니다. 런타임에 HugeTLB 페이지 동작에 영향을 주는 매개변수에 대한 자세한 내용은 런타임에 HugeTLB 페이지 예약 매개 변수 및 이러한 매개변수를 사용하여 런타임에 HugeTLB 페이지를 구성하는 방법을 참조하십시오.

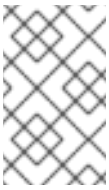
Transparent HugePages (THP)

THP를 사용하면 커널은 프로세스에 대규모 페이지를 자동으로 할당하므로 정적 대규모 페이지를

수동으로 예약할 필요가 없습니다. 다음은 THP에서의 두 가지 작동 모드입니다.

- System-wide: 여기에서 커널은 대규모 페이지를 할당할 수 있을 때마다 프로세스에 대규모 페이지를 할당하고 프로세스가 대규모 연속 가상 메모리 영역을 사용하고 있습니다.

- 여기서 커널은 madvise() 시스템 호출을 사용하여 지정할 수 있는 개별 프로세스의 메모리 영역에 대규모 페이지만 할당합니다.



참고

THP 기능은 2MB 페이지만 지원합니다.

부팅 시 HugeTLB 페이지 동작에 영향을 주는 매개변수에 대한 자세한 내용은 투명한 hugepages 및 투명한 hugepages 비활성화를 참조하십시오.

36.2. 부팅 시 HUGETLB 페이지 예약 매개변수

다음 매개변수를 사용하여 부팅 시 HugeTLB 페이지 동작에 영향을 미칩니다.

부팅 시 HugeTLB 페이지를 구성하는 데 이러한 매개변수를 사용하는 방법에 대한 자세한 내용은 부팅 시 HugeTLB 구성을 참조하십시오.

표 36.1. 부팅 시 HugeTLB 페이지를 구성하는 데 사용되는 매개변수

매개변수	설명	기본값
hugepages	부팅 시 커널에 구성된 영구 대규모 페이지 수를 정의합니다. NUMA 시스템에서 이 매개 변수가 정의된 대규모 페이지는 노드 간에 동일하게 나뉩니다. **/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages** 파일의 노드 값을 변경하여 런타임 시 특정 노드에 대규모 페이지를 할당할 수 있습니다.	기본값은 **0**입니다. 부팅 시 이 값을 업데이트하려면 **/proc/sys/vm/nr_hugepages** 파일에서 이 매개변수 값을 변경합니다.

매개변수	설명	기본값
hugepagesz	부팅 시 커널에 구성된 영구 대규모 페이지의 크기를 정의합니다.	유효한 값은 **2MB** 및 **1GB** 입니다. 기본값은 **2MB** 입니다.
default_hugepagesz	부팅 시 커널에 구성된 영구 대규모 페이지의 기본 크기를 정의합니다.	유효한 값은 **2MB** 및 **1GB** 입니다. 기본값은 **2MB** 입니다.

36.3. 부팅 시 HUGETLB 구성

HugeTLB 하위 시스템이 지원하는 페이지 크기는 아키텍처에 따라 다릅니다. x86_64 아키텍처는 2MB 의 대규모 페이지와 1GB 대규모 페이지를 지원합니다.

다음 절차에서는 부팅 시 1GB 페이지를 예약하는 방법을 설명합니다.

절차

1. */etc/default/grub 파일의 커널 명령줄 옵션을 root로 추가하여 1GB 페이지용 HugeTLB 풀을 생성합니다.*

```
default_hugepagesz=1G hugepagesz=1G
```

2. *편집한 기본 파일을 사용하여 GRUB2 설정을 다시 생성합니다.*

   a. *시스템이 BIOS 펌웨어를 사용하는 경우 다음 명령을 실행합니다.*

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

   b. *시스템이 UEFI 프레임워크를 사용하는 경우 다음 명령을 실행합니다.*

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

3. */usr/lib/systemd/system/ 디렉터리에 hugetlb-gigantic-pages.service 라는 새 파일을 생성하고 다음 내용을 추가합니다.*

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
```

```
DefaultDependencies=no
Before=dev-hugepages.mount
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/lib/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

4. */usr/lib/systemd/* 디렉터리에 **hugetlb-reserve-pages.sh** 라는 새 파일을 생성하고 다음 내용을 추가합니다.

다음 콘텐츠를 추가하는 동안 **number_of_pages** 를 예약하려는 **1GB** 페이지 수로, **node** 를 이러한 페이지를 예약할 노드 이름으로 바꿉니다.

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
   echo "ERROR: $nodes_path does not exist"
   exit 1
fi

reserve_pages()
{
   echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages number_of_pages node
```

예를 들어 **node0** 에 **1GB** 페이지와 **node1** 에 **1GB** 페이지를 예약하려면 **number_of_pages** 를 **node0** 의 경우 **2**로, **node1** 의 경우 **1**을 바꿉니다.

```
reserve_pages 2 node0
reserve_pages 1 node1
```

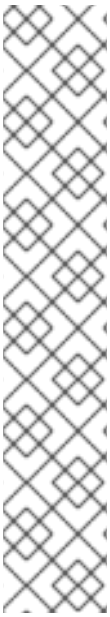
5. 실행 가능한 스크립트를 생성합니다.

```
# chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
```

6. 초기 부팅 예약을 활성화합니다.

-

```
# systemctl enable hugetlb-gigantic-pages
```



참고

- 언제든지 **nr_hugepages** 에 작성하여 런타임 시 **1GB** 페이지를 예약하려고 할 수 있습니다. 그러나 이러한 예약은 메모리 조각화로 인해 실패할 수 있습니다. **1GB** 페이지를 예약하는 가장 안정적인 방법은 부팅 중에 초기에 실행되는 **hugetlb-reserve-pages.sh** 스크립트를 사용하는 것입니다.

- 정적 대규모 페이지를 예약하면 시스템에서 사용할 수 있는 메모리 양을 효과적으로 줄이고 전체 메모리 용량을 올바르게 사용하지 못하게 할 수 있습니다. 예약된 대규모 페이지의 적절히 크기 조정된 대규모 페이지 풀은 이를 사용하는 애플리케이션에 유용할 수 있지만, 예약된 대규모 페이지의 과도하게 또는 사용되지 않은 풀은 결국 전체 시스템 성능에 부담을 줄 것입니다. 예약된 대규모 페이지 풀을 설정할 때 시스템이 전체 메모리 용량을 올바르게 활용할 수 있는지 확인합니다.

추가 리소스

- **systemd.service(5) man page**

- **/usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt file**

36.4. 런타임 시 HUGETLB 페이지 예약 매개변수

다음 매개변수를 사용하여 런타임에 **HugeTLB** 페이지 동작에 영향을 미칩니다.

런타임에 이러한 매개변수를 사용하여 **HugeTLB** 페이지를 구성하는 방법에 대한 자세한 내용은 런타임에 **HugeTLB** 구성을 참조하십시오.

표 **36.2.** 런타임 시 **HugeTLB** 페이지를 구성하는 데 사용되는 매개변수

매개변수	설명	파일 이름
nr_hugepages	지정된 NUMA 노드에 할당된 지정된 크기의 대규모 페이지 수를 정의합니다.	**/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages**

매개변수	설명	파일 이름
nr_overcommit_hugepages	메모리 과다 할당을 통해 시스템에서 생성하고 사용할 수 있는 최대 대규모 페이지 수를 정의합니다. 이 파일에 0이 아닌 값을 기록하면 영구 대규모 페이지 풀이 소진된 경우 시스템이 커널의 일반 페이지 풀에서 대규모 페이지 수를 얻을 수 있음을 나타냅니다. 이러한 대규모 페이지가 사용되지 않으므로 여유가 해제되고 커널의 일반 페이지 풀로 반환됩니다.	**/proc/sys/vm/nr_overcommit_hugepages**

36.5. 런타임에 HUGETLB 구성

*이 절차에서는 **node2** 에 **20 2048 kB** 대규모 페이지를 추가하는 방법을 설명합니다.*

요구 사항에 따라 페이지를 예약하려면 다음을 교체하십시오.

- ***20** 예약하려는 대규모 페이지 수입니다.*
- ***Huge Page**의 크기가 있는 **2048KB***
- *페이지를 예약하려는 노드가 있는 **node2** 입니다.*

절차

1. *메모리 통계를 표시합니다.*

```
# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1 Node 2 Node 3  Total add
AnonHugePages         0      2      0      8     10
HugePages_Total       0      0      0      0      0
HugePages_Free        0      0      0      0      0
HugePages_Surp        0      0      0      0      0
```

2. 지정된 크기의 대규모 페이지 수를 노드에 추가합니다.

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

검증 단계

- 대규모 페이지 수가 추가되었는지 확인합니다.

```
# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1 Node 2 Node 3  Total
AnonHugePages         0      2      0      8     10
HugePages_Total       0      0     40      0     40
HugePages_Free        0      0     40      0     40
HugePages_Surp        0      0      0      0      0
```

추가 리소스

- **numastat(8)** 도움말 페이지

36.6. 투명 HUGEPAGES 활성화

THP는 **Red Hat Enterprise Linux 9**에서 기본적으로 활성화되어 있습니다. 그러나 **THP**를 활성화하거나 비활성화할 수 있습니다.

이 절차에서는 **THP**를 활성화하는 방법을 설명합니다.

절차

1. **THP**의 현재 상태를 확인합니다.

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. **THP**를 활성화합니다.

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

3. 애플리케이션이 필요 이상으로 많은 메모리 리소스를 할당하지 않도록 하려면 시스템 전체 투명한 대규모 페이지를 비활성화하고 **madvise** 를 통해 명시적으로 요청하는 애플리케이션에 대

해서만 해당 페이지를 활성화합니다.

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```

참고

단기 할당에 짧은 할당에 짧은 대기 시간을 제공하면 수명이 긴 할당으로 최상의 성능을 즉시 달성하는 것보다 우선 순위가 높습니다. 이러한 경우 **THP**를 사용하도록 설정한 상태에서 직접 압축을 비활성화할 수 있습니다.

직접 압축은 대규모 페이지 할당 중에 동기식 메모리 압축입니다. 직접 압축을 비활성화하면 메모리를 절약할 수 없지만 페이지 폴트 중에 대기 시간이 길어지는 위험을 줄일 수 있습니다. **THP**를 통해 워크로드의 이점이 크게 감소하는 경우 성능이 저하됩니다. 직접 압축을 비활성화하십시오.

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
```

추가 리소스

- **madvise(2)** 매뉴얼 페이지
- 투명한 **hugepages** 비활성화.

36.7. 투명 HUGEPAGES 비활성화

THP는 **Red Hat Enterprise Linux 9**에서 기본적으로 활성화되어 있습니다. 그러나 **THP**를 활성화하거나 비활성화할 수 있습니다.

이 절차에서는 **THP**를 비활성화하는 방법을 설명합니다.

절차

1. **THP**의 현재 상태를 확인합니다.

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. *THP를 비활성화합니다.*

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

36.8. 번역 버퍼 크기에 대한 페이지 크기의 영향

페이지 테이블에서 주소 매핑을 읽는 데 시간이 오래 걸리는 경우 리소스 확장이므로 CPU는 최근에 사용한 주소 캐시(TLB)로 빌드됩니다. 그러나 기본 TLB는 특정 수의 주소 매핑만 캐시할 수 있습니다.

요청된 주소 매핑이 TLB라는 TLB에 없는 경우에도 시스템은 여전히 페이지 테이블을 읽고 가상 주소 매핑에 물리적인 주소를 결정해야 합니다. 애플리케이션 메모리 요구 사항과 주소 매핑에 사용되는 페이지 크기 간의 관계로 인해 메모리 요구 사항이 큰 애플리케이션이 최소 메모리 요구 사항이 있는 애플리케이션보다 성능이 저하될 가능성이 높습니다. 따라서 가능한 경우 TLB 누락을 방지하는 것이 중요합니다.

HugeTLB 및 Transparent Huge Page 기능을 사용하면 애플리케이션이 4KB 보다 큰 페이지를 사용할 수 있습니다. 이를 통해 TLB에 저장된 주소가 더 많은 메모리를 참조할 수 있으므로 TLB 누락을 줄이고 애플리케이션 성능이 향상됩니다.

37장. SYSTEMTAP 시작하기

시스템 관리자는 SystemTap을 사용하여 실행 중인 Linux 시스템에서 버그 또는 성능 문제의 근본 원인을 확인할 수 있습니다.

애플리케이션 개발자는 SystemTap을 사용하여 애플리케이션이 Linux 시스템 내에서 어떻게 작동하는지 정확하게 모니터링할 수 있습니다.

37.1. SYSTEMTAP의 목적

SystemTap은 운영 체제(특히 커널)의 활동을 조사하고 모니터링하는 데 사용할 수 있는 추적 및 프로빙 툴입니다. SystemTap은 netstat,ps,top, iostat 와 같은 툴 출력과 유사한 정보를 제공합니다. 그러나 SystemTap은 수집된 정보에 대해 더 많은 필터링 및 분석 옵션을 제공합니다. SystemTap 스크립트에서 SystemTap이 수집하는 정보를 지정합니다.

SystemTap은 사용자에게 커널 활동을 추적하고 이 기능을 두 가지 속성과 결합하는 인프라를 제공하여 기존 Linux 모니터링 툴 세트를 보완하는 것을 목표로 합니다.

유연성

SystemTap 프레임워크를 사용하면 다양한 커널 기능, 시스템 호출 및 커널 공간에 발생하는 기타 이벤트를 조사하고 모니터링하는 간단한 스크립트를 개발할 수 있습니다. 이를 통해 SystemTap은 자체 커널별 감지 및 모니터링 툴을 개발할 수 있는 시스템이기 때문에 많은 도구가 아닙니다.

ease-of-Use

SystemTap을 사용하면 커널을 다시 컴파일하거나 시스템을 재부팅하지 않고도 커널 활동을 모니터링할 수 있습니다.

37.2. SYSTEMTAP 설치

SystemTap 사용을 시작하려면 필수 패키지를 설치합니다. 시스템에 여러 커널이 설치된 커널에서 SystemTap을 사용하려면 각 커널 버전에 해당하는 필수 커널 패키지를 설치하십시오.

사전 요구 사항

- 디버그 및 소스 리포지토리 활성화에 설명된 대로 디버그 리포지토리를 활성화했습니다.

절차

1. *필요한 **SystemTap** 패키지를 설치합니다.*

```
# dnf install systemtap
```

2. *필요한 커널 패키지를 설치합니다.*

   a. ***stap-prep** 사용:*

```
# stap-prep
```

   b. ***stap-prep** 이 작동하지 않는 경우 필요한 커널 패키지를 수동으로 설치합니다.*

```
# dnf install kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -i)-
$(uname -r) kernel-devel-$(uname -r)
```

   ***$(uname -i)** 는 시스템의 하드웨어 플랫폼으로 자동 교체되며 **$(uname -r)** 는 실행 중인 커널 버전으로 자동 교체됩니다.*

검증 단계

- ***SystemTap**으로 확인할 커널을 현재 사용 중인 경우 설치에 성공했는지 테스트합니다.*

```
# stap -v -e 'probe kernel.function("vfs_read") {printf("read performed\n"); exit()}'
```

***SystemTap**을 성공적으로 배포하면 다음과 유사한 출력이 생성됩니다.*

```
Pass 1: parsed user script and 45 library script(s) in 340usr/0sys/358real ms.
Pass 2: analyzed script: 1 probe(s), 1 function(s), 0 embed(s), 0 global(s) in
290usr/260sys/568real ms.
Pass 3: translated to C into
"/tmp/stapiArgLX/stap_e5886fa50499994e6a87aacdc43cd392_399.c" in
490usr/430sys/938real ms.
Pass 4: compiled C into "stap_e5886fa50499994e6a87aacdc43cd392_399.ko" in
3310usr/430sys/3714real ms.
Pass 5: starting run. 1
read performed 2
Pass 5: run completed in 10usr/40sys/73real ms. 3
```

출력의 마지막 세 줄(**Pass 5**사용 시작)은 다음을 나타냅니다.

1

SystemTap이 커널을 검색하고 계측을 실행한 계측을 성공적으로 생성했습니다.

2

SystemTap이 지정된 이벤트(이 경우 **ARWX** 읽기)를 감지했습니다.

3

SystemTap이 유효한 처리기를 실행했습니다(텍스트를 인쇄한 다음 오류 없이 종료).

37.3. SYSTEMTAP을 실행하는 권한

SystemTap 스크립트를 실행하려면 상승된 시스템 권한이 필요하지만, 경우에 따라 권한이 없는 사용자는 시스템에서 **SystemTap** 계측을 실행해야 할 수 있습니다.

사용자가 **root** 액세스없이 **SystemTap**을 실행할 수 있도록 하려면 다음 사용자 그룹 모두에 사용자를 추가합니다.

stapdev

이 그룹의 멤버는 **stap** 를 사용하여 **SystemTap** 스크립트를 실행하거나 **staprun** 을 사용하여 **SystemTap** 계측 모듈을 실행할 수 있습니다.

stap 를 실행하려면 **SystemTap** 스크립트를 커널 모듈로 컴파일하고 커널에 로드해야 합니다. 여기에는 **stapdev** 멤버에 부여된 시스템에 대한 승격된 권한이 필요합니다. 유감스럽게도 이러한 권한은 **stapdev** 멤버에 대한 효과적인 **root** 액세스 권한을 부여합니다. 따라서 **root** 액세스 권한으로 신뢰할 수 있는 사용자에게 **stapdev** 그룹 멤버십만 부여합니다.

stapusr

이 그룹의 멤버는 **staprun** 을 사용하여 **SystemTap** 계측 모듈을 실행할 수 있습니다. 또한 **/lib/modules/kernel_version/systemtap/** 디렉터리에서만 해당 모듈을 실행할 수 있습니다. 이 디렉터리는 **root** 사용자만 소유해야 하며 **root** 사용자만 쓸 수 있어야 합니다.

37.4. SYSTEMTAP 스크립트 실행

표준 입력 또는 파일에서 **SystemTap** 스크립트를 실행할 수 있습니다.

SystemTap 설치와 함께 배포된 샘플 스크립트는 **/usr/share/systemtap/examples** 디렉터리에서 확인할 수 있습니다.

사전 요구 사항

1. **SystemTap** 및 관련 필수 커널 패키지는 **Systemtap 설치에 설명된** 대로 설치됩니다.

2. **SystemTap** 스크립트를 일반 사용자로 실행하려면 사용자를 **SystemTap** 그룹에 추가합니다.

```
# usermod --append --groups
stapdev,stapusr user-name
```

절차

- **SystemTap** 스크립트를 실행합니다.
  - 표준 입력에서 다음을 수행합니다.

```
# echo "probe timer.s(1) {exit()}" | stap -
```

이 명령은 **echo** 가 표준 입력에 전달하는 스크립트를 실행하도록 **stap** 에 지시합니다. **stap** 옵션을 추가하려면 **-** 문자 앞에 삽입합니다. 예를 들어 이 명령의 결과를 보다 자세한 표시로 만들려면 명령은 다음과 같습니다.

```
# echo "probe timer.s(1) {exit()}" | stap -v -
```

  - 파일에서 다음을 수행합니다.

```
# stap file_name
```

38장. SYSTEMTAP의 교차 계측

SystemTap의 교차 계측은 SystemTap을 완전히 배포하지 않은 다른 시스템에서 사용할 SystemTap 스크립트에서 SystemTap 계측 모듈을 생성하고 있습니다.

38.1. SYSTEMTAP 교차 계측

SystemTap 스크립트를 실행하면 해당 스크립트에서 커널 모듈이 빌드됩니다. 그런 다음 SystemTap 이 모듈을 커널에 로드합니다.

일반적으로 SystemTap 스크립트는 SystemTap이 배포된 시스템에서만 실행할 수 있습니다. 10개의 시스템에서 SystemTap을 실행하려면 SystemTap을 이러한 모든 시스템에 배포해야 합니다. 어떤 경우에는 이것이 가능하지 않거나 바람직하지 않을 수 있습니다. 예를 들어 기업 정책은 컴파일러를 제공하는 패키지를 설치하거나 특정 시스템에 대한 정보를 디버그하지 못하도록 SystemTap 배포를 방지할 수 있습니다.

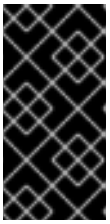
이 문제를 해결하려면 교차 계측을 사용하십시오. 교차 계측은 다른 시스템에서 사용할 SystemTap 스크립트에서 SystemTap 계측 모듈을 생성하는 프로세스입니다. 이 프로세스는 다음과 같은 이점을 제공합니다.

- 다양한 시스템의 커널 정보 패키지를 단일 호스트 시스템에 설치할 수 있습니다.

중요

커널 패키징 버그로 인해 설치를 방지할 수 있습니다. 이러한 경우 호스트 시스템의 kernel-debuginfo 및 kernel-devel 패키지가 일치해야 합니다. 버그가 발생하면 https://bugzilla.redhat.com/ 에서 버그를 보고합니다.

- 생성된 SystemTap 계측 모듈인 systemtap-runtime 을 사용하려면 각 대상 시스템을 한 개만 설치해야 합니다.



중요

호스트 시스템이 동일한 아키텍처여야 하며 기본 조정 모듈이 작동하려면 대상 시스템과 동일한 Linux 배포를 실행해야 합니다.

용어

계측 모듈

SystemTap 스크립트로 빌드된 커널 모듈. **SystemTap** 모듈은 호스트 시스템에 빌드되고 대상 시스템의 대상 커널에 로드됩니다.

호스트 시스템

계측 모듈(**SystemTap** 스크립트)이 컴파일되는 시스템입니다. 대상 시스템에 로드됩니다.

대상 시스템

계측 모듈이 빌드되는 시스템입니다(**SystemTap** 스크립트에서).

대상 커널

대상 시스템의 커널입니다. 이는 계측 모듈을 로드하고 실행하는 커널입니다.

38.2. SYSTEMTAP의 교차 계측 초기화

SystemTap의 교차 계측을 초기화하여 한 시스템에서 **SystemTap** 계측 모듈을 빌드하고 **SystemTap**을 완전히 배포하지 않은 다른 시스템에서 사용합니다.

사전 요구 사항

- **SystemTap**은 Systemtap 설치에 설명된 대로 호스트 시스템에 설치됩니다.
- **systemtap-runtime** 패키지는 각 대상 시스템에 설치됩니다.

```
# dnf install systemtap-runtime
```

- 호스트 시스템과 대상 시스템은 모두 동일한 아키텍처입니다.
- 호스트 시스템과 대상 시스템 모두 동일한 **Red Hat Enterprise Linux** 버전(예: **Red Hat Enterprise Linux 9**)을 실행하고 있습니다.

중요

커널 패키징 버그는 여러 **kernel-debuginfo** 및 **kernel-devel** 패키지가 하나의 시스템에 설치되지 않도록 할 수 있습니다. 이러한 경우 호스트 시스템과 대상 시스템의 마이너 버전이 일치해야 합니다. 버그가 발생하면 **https://bugzilla.redhat.com/** 에서 보고합니다.

절차

1. 각 대상 시스템에서 실행 중인 커널을 확인합니다.

```
$ uname -r
```

각 대상 시스템에 대해 이 단계를 반복합니다.

2. 호스트 시스템에서 **Systemtap** 설치에 설명된 방법으로 각 대상 시스템의 대상 커널 및 관련 패키지를 설치합니다.

3. 호스트 시스템에 계측 모듈을 빌드하고 이 모듈을 에 복사하고 대상 시스템의 에서 이 모듈을 실행합니다.

   a. 원격 구현 사용:

```
# stap --remote target_system script
```

   이 명령은 대상 시스템에서 지정된 스크립트를 원격으로 구현합니다. 호스트 시스템에서 대상 시스템에 **SSH** 연결을 수행할 수 있는지 확인해야 합니다.

   b. 수동:

      i. 호스트 시스템에서 계측 모듈을 빌드합니다.

```
# stap -r kernel_version script -m module_name -p 4
```

      여기서 **kernel_version** 은 **1**단계에서 결정된 대상 커널 버전을 나타냅니다. 스크립트 는 계측 모듈로 변환되는 스크립트를 나타내며 **module _name** 은 계측 모듈 의 원

*하는 이름입니다. **p4** 옵션은 **SystemTap**에 컴파일된 모듈을 로드하고 실행하지 않도록 지시합니다.*

ii. *계측 모듈이 컴파일되면 대상 시스템에 복사하여 다음 명령을 사용하여 로드합니다.*

```
# staprun module_name.ko
```

39장. SYSTEMTAP으로 네트워크 활동 모니터링

systemtap-testsuite 패키지를 설치할 때 /usr/share/systemtap/testsuite/systemtap.examples/ 디렉터리에서 사용할 수 있는 SystemTap 스크립트 예를 사용하여 시스템의 네트워크 활동을 모니터링하고 조사할 수 있습니다.

39.1. SYSTEMTAP을 사용한 네트워크 활동 프로파일링

nettop.stp 예제 SystemTap 스크립트를 사용하여 네트워크 활동을 프로파일링할 수 있습니다. 스크립트는 시스템에서 네트워크 트래픽을 생성하는 프로세스를 추적하고 각 프로세스에 대한 다음 정보를 제공합니다.

PID

나열된 프로세스의 ID입니다.

UID

사용자 ID. 사용자 ID 0은 root 사용자를 나타냅니다.

DEV

데이터 전송 또는 수신에 사용되는 프로세스가 ethernet 장치(예: eth0, eth1)입니다.

XMIT_PK

프로세스에서 전송하는 패킷 수입니다.

RECV_PK

프로세스에서 수신한 패킷 수입니다.

XMIT_KB

프로세스에서 보낸 데이터 양(KB)입니다.

RECV_KB

서비스에서 수신한 데이터의 양(KB)입니다.

사전 요구 사항

- SystemTap 설치에 설명된 대로 SystemTap 을 설치했습니다.

절차

- **nettop.stp** *스크립트를 실행합니다.*

```
# stap  --example nettop.stp
```

nettop.stp *스크립트는* **5***초마다 네트워크 프로필 샘플링을 제공합니다.*

nettop.stp *스크립트의 출력은 다음과 유사합니다.*

```
[...]
  PID   UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
    0     0 eth0         0       5       0       0 swapper
11178     0 eth0         2       0       0       0 synergyc
  PID   UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
 2886     4 eth0        79       0       5       0 cups-polld
11362     0 eth0         0      61       0       5 firefox
    0     0 eth0         3      32       0       3 swapper
 2886     4 lo           4       4       0       0 cups-polld
11178     0 eth0         3       0       0       0 synergyc
  PID   UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
    0     0 eth0         0       6       0       0 swapper
 2886     4 lo           2       2       0       0 cups-polld
11178     0 eth0         3       0       0       0 synergyc
 3611     0 eth0         0       1       0       0 Xorg
  PID   UID DEV    XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
    0     0 eth0         3      42       0       2 swapper
11178     0 eth0        43       1       3       0 synergyc
11362     0 eth0         0       7       0       0 firefox
 3897     0 eth0         0       1       0       0 multiload-apple
```

39.2. SYSTEMTAP을 사용하여 네트워크 소켓 코드에서 호출되는 함수 추적

socket-trace.stp *예제* **SystemTap** *스크립트를 사용하여 커널의* **net/socket.c** *파일에서 라는 함수를 추적할 수 있습니다. 이렇게 하면 각 프로세스가 커널 수준에서 네트워크와 상호 작용하는 방식을 자세히 식별할 수 있습니다.*

사전 요구 사항

- **SystemTap** *설치에 설명된 대로* **SystemTap** *을 설치했습니다.*

절차

- **socket-trace.stp** 스크립트를 실행합니다.

```
# stap  --example socket-trace.stp
```

socket-trace.stp 스크립트 출력의 **3**초 발췌 내용은 다음과 유사합니다.

```
[...]
0 Xorg(3611): -> sock_poll
3 Xorg(3611): <- sock_poll
0 Xorg(3611): -> sock_poll
3 Xorg(3611): <- sock_poll
0 gnome-terminal(11106): -> sock_poll
5 gnome-terminal(11106): <- sock_poll
0 scim-bridge(3883): -> sock_poll
3 scim-bridge(3883): <- sock_poll
0 scim-bridge(3883): -> sys_socketcall
4 scim-bridge(3883):  -> sys_recv
8 scim-bridge(3883):   -> sys_recvfrom
12 scim-bridge(3883):-> sock_from_file
16 scim-bridge(3883):<- sock_from_file
20 scim-bridge(3883):-> sock_recvmsg
24 scim-bridge(3883):<- sock_recvmsg
28 scim-bridge(3883):   <- sys_recvfrom
31 scim-bridge(3883):  <- sys_recv
35 scim-bridge(3883): <- sys_socketcall
[...]
```

39.3. SYSTEMTAP으로 네트워크 패킷 삭제 모니터링

Linux의 네트워크 스택은 다양한 이유로 패킷을 삭제할 수 있습니다. 일부 **Linux** 커널에는 패킷이 삭제되는 추적 지점인 **kernel.trace("kfree_skb")** 가 포함되어 있습니다.

dropwatch.stp SystemTap 스크립트는 **kernel.trace("kfree_skb")** 를 사용하여 패킷을 삭제합니다. 이 스크립트는 **5**초 간격으로 패킷을 삭제하는 위치를 요약합니다.

사전 요구 사항

- **SystemTap** 설치에 설명된 대로 **SystemTap** 을 설치했습니다.

절차

- **dropwatch.stp** 스크립트를 실행합니다.

-

```
# stap  --example dropwatch.stp
```

15초 동안 dropwatch.stp 스크립트를 실행하면 다음과 유사한 출력이 표시됩니다.

```
Monitoring for dropped packets
51 packets dropped at location 0xffffffff8024cd0f
2 packets dropped at location 0xffffffff8044b472
51 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
97 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
Stopping dropped packet monitor
```

참고

패킷 위치를 보다 의미 있게 떨어지도록 하려면 **/boot/System.map-$(uname -r)** 파일을 참조하십시오. 이 파일에는 각 함수의 시작 주소가 나열되므로 **dropwatch.stp** 스크립트 출력의 주소를 특정 함수 이름에 매핑할 수 있습니다. **/boot/System.map-$(uname -r)** 파일의 스니펫을 고려할 때 **0xffffffffff8024cd0f** 주소는 **unix_stream_recvmsg** 에 매핑되고 주소 **0xffffffffffffffffff8044b472** 는 함수 **arp_rcv:**

```
[...]
ffffffff8024c5cd T unlock_new_inode
ffffffff8024c5da t unix_stream_sendmsg
ffffffff8024c920 t unix_stream_recvmsg
ffffffff8024cea1 t udp_v4_lookup_longway
[...]
ffffffff8044addc t arp_process
ffffffff8044b360 t arp_rcv
ffffffff8044b487 t parp_redo
ffffffff8044b48c t arp_solicit
[...]
```

40장. SYSTEMTAP을 사용하여 커널 활동 프로파일링

다음 섹션에서는 함수 호출을 모니터링하여 커널 활동을 프로파일링하는 스크립트를 보여줍니다.

40.1. SYSTEMTAP으로 함수 호출 계산

functioncallcount.stp SystemTap 스크립트를 사용하여 특정 커널 함수 호출을 계산할 수 있습니다. 이 스크립트를 사용하여 여러 커널 기능을 대상으로 지정할 수도 있습니다.

사전 요구 사항

- **Systemtap 설치**에 설명된 대로 **SystemTap**을 설치했습니다.

절차

- **functioncallcount.stp** 스크립트를 실행합니다.

```
# stap --example functioncallcount.stp 'argument'
```

이 스크립트는 대상 커널 함수를 인수로 사용합니다. 인수 와일드카드를 사용하여 여러 커널 기능을 특정 범위까지 대상으로 할 수 있습니다.

스크립트 출력의 알파벳순으로는 호출된 함수의 이름과 샘플 시간 동안 호출된 횟수가 포함됩니다.

다음 예제를 고려하십시오.

```
# stap -w -v --example functioncallcount.stp "*@mm*.c" -c /bin/true
```

다음과 같습니다.

- **-w:** 경고 경고를 표시합니다.
- **-V :** 시작 커널의 출력을 볼 수 있도록 합니다.

- ***-c command : Tells SystemTap** 명령을 실행하는 동안 함수 호출을 계산하도록 **SystemTap**. 이 예제에서는 **/bin/true** 입니다.*

  출력은 다음과 유사합니다.

```
[...]
__vma_link 97
__vma_link_file 66
__vma_link_list 97
__vma_link_rb 97
__xchg 103
add_page_to_active_list 102
add_page_to_inactive_list 19
add_to_page_cache 19
add_to_page_cache_lru 7
all_vm_events 6
alloc_pages_node 4630
alloc_slabmgmt 67
anon_vma_alloc 62
anon_vma_free 62
anon_vma_lock 66
anon_vma_prepare 98
anon_vma_unlink 97
anon_vma_unlock 66
arch_get_unmapped_area_topdown 94
arch_get_unmapped_exec_area 3
arch_unmap_area_topdown 97
atomic_add 2
atomic_add_negative 97
atomic_dec_and_test 5153
atomic_inc 470
atomic_inc_and_test 1
[...]
```

40.2. SYSTEMTAP을 사용하여 함수 호출 추적

***para-callgraph.stp SystemTap** 스크립트를 사용하여 함수 호출 및 함수 반환을 추적할 수 있습니다.*

사전 요구 사항

- ***Systemtap** 설치에 설명된 대로 **SystemTap**을 설치했습니다.*

절차

- ***para-callgraph.stp** 스크립트를 실행합니다.*

```
# stap --example para-callgraph.stp 'argument1' 'argument2'
```

script para-callgraph.stp 에는 두 개의 명령줄 인수가 사용됩니다.

1. 추적하려는 항목/시험의 이름입니다.

2. 스레드별로 추적을 활성화하거나 비활성화하는 선택적 트리거 함수입니다. 트리거 함수가 아직 종료되지 않은 한 각 스레드에서 추적은 계속됩니다.

다음 예제를 고려하십시오.

```
# stap -wv --example para-callgraph.stp 'kernel.function("*@fs/proc.c*")' 'kernel.function("vfs_read")' -c "cat /proc/sys/vm/* || true"
```

다음과 같습니다.

- **-w:** 경고 경고를 표시합니다.

- **-V :** 시작 커널의 출력을 볼 수 있도록 합니다.

- **-c command : Tells SystemTap** 명령을 실행하는 동안 함수 호출을 계산하도록 **SystemTap.** 이 예제에서는 **/bin/true** 입니다.

출력은 다음과 유사합니다.

```
[...]
   267 gnome-terminal(2921): <-do_sync_read return=0xfffffffffffffff5
   269 gnome-terminal(2921):<-vfs_read return=0xfffffffffffffff5
     0 gnome-terminal(2921):->fput file=0xffff880111eebbc0
     2 gnome-terminal(2921):<-fput
     0 gnome-terminal(2921):->fget_light fd=0x3 fput_needed=0xffff88010544df54
     3 gnome-terminal(2921):<-fget_light return=0xffff8801116ce980
     0 gnome-terminal(2921):->vfs_read file=0xffff8801116ce980 buf=0xc86504 count=0x1000 pos=0xffff88010544df48
     4 gnome-terminal(2921): ->rw_verify_area read_write=0x0 file=0xffff8801116ce980 ppos=0xffff88010544df48 count=0x1000
     7 gnome-terminal(2921): <-rw_verify_area return=0x1000
```

```
    12 gnome-terminal(2921): ->do_sync_read filp=0xffff8801116ce980 buf=0xc86504 len=0x1000
ppos=0xffff88010544df48
    15 gnome-terminal(2921): <-do_sync_read return=0xfffffffffffffff5
    18 gnome-terminal(2921):<-vfs_read return=0xfffffffffffffff5
     0 gnome-terminal(2921):->fput file=0xffff8801116ce980
```

40.3. SYSTEMTAP을 사용하여 커널 및 사용자 공간에 소요되는 시간 확인

thread-times.stp SystemTap 스크립트를 사용하여 지정된 스레드가 커널 또는 사용자 공간에 소비되는 시간을 확인할 수 있습니다.

사전 요구 사항

- **Systemtap** 설치에 설명된 대로 **SystemTap**을 설치했습니다.

절차

- **thread-times.stp** 스크립트를 실행합니다.

```
# stap --example thread-times.stp
```

이 스크립트는 **5**초 동안 **CPU** 시간을 소비하는 상위 **20**개의 프로세스와 샘플 중에 수행된 총 **CPU** 눈금 수를 표시합니다. 이 스크립트의 출력은 각 프로세스가 사용된 **CPU** 시간과 해당 시간이 커널 공간 또는 사용자 공간에 소비되었는지 여부도 기록합니다.

```
tid   %user %kernel (of 20002 ticks)
  0   0.00%  87.88%
32169   5.24%   0.03%
9815   3.33%   0.36%
9859   0.95%   0.00%
3611   0.56%   0.12%
9861   0.62%   0.01%
11106   0.37%   0.02%
32167   0.08%   0.08%
3897   0.01%   0.08%
3800   0.03%   0.00%
2886   0.02%   0.00%
3243   0.00%   0.01%
3862   0.01%   0.00%
3782   0.00%   0.00%
21767   0.00%   0.00%
2522   0.00%   0.00%
3883   0.00%   0.00%
3775   0.00%   0.00%
3943   0.00%   0.00%
3873   0.00%   0.00%
```

40.4. SYSTEMTAP으로 폴링 애플리케이션 모니터링

timeout.stp SystemTap 스크립트를 사용하여 폴링 중인 애플리케이션을 식별하고 모니터링할 수 있습니다. 이를 통해 불필요한 또는 과도한 폴링을 추적할 수 있으므로 **CPU** 사용량 및 전력 절감 측면에서 개선할 수 있는 영역을 파악하는 데 도움이 됩니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- **timeout.stp** 스크립트를 실행합니다.

```
# stap --example timeout.stp
```

이 스크립트는 각 애플리케이션에서 시간이 지남에 따라 다음 시스템 호출을 사용하는 횟수를 추적합니다.

- **poll**
- 선택 사항
- **epoll**
- **itimer**
- **futex**
- **nanosleep**
- **signal**

이 예제 출력에서 어떤 프로세스가 어떤 시스템 호출을 사용했는지와 횟수를 확인할 수 있습니다.

```
uid |   poll  select   epoll  itimer   futex nanosle  signal| process
28937 | 148793      0      0    4727   37288      0       0| firefox
22945 |      0   56949      0       1       0      0       0| scim-bridge
    0 |      0       0      0   36414       0      0       0| swapper
4275 |  23140       0      0       1       0      0       0| mixer_applet2
4191 |      0   14405      0       0       0      0       0| scim-launcher
22941 |   7908       1      0      62       0      0       0| gnome-terminal
4261 |      0       0      0       2       0   7622       0| escd
3695 |      0       0      0       0       0   7622       0| gdm-binary
3483 |      0    7206      0       0       0      0       0| dhcdbd
4189 |   6916       0      0       2       0      0       0| scim-panel-gtk
1863 |   5767       0      0       0       0      0       0| iscsid
```

40.5. SYSTEMTAP으로 가장 자주 사용되는 시스템 호출 추적

topsys.stp SystemTap 스크립트를 사용하여 **5**초 간격으로 시스템에서 사용하는 상위 **20**개의 시스템 호출을 나열할 수 있습니다. 또한 해당 기간 동안 각 시스템 호출이 사용된 횟수를 나열합니다.

사전 요구 사항

- **Systemtap** 설치에 설명된 대로 **SystemTap**을 설치했습니다.

절차

- **topsys.stp** 스크립트를 실행합니다.

```
# stap --example topsys.stp
```

다음 예제를 고려하십시오.

```
# stap -v --example topsys.stp
```

여기서 **-v**를 사용하면 시작 커널의 출력이 표시됩니다.

출력은 다음과 유사합니다.

```
----------------------------------------------------------------
              SYSCALL      COUNT
```

```
        gettimeofday       1857
                read       1821
               ioctl       1568
                poll       1033
               close        638
                open        503
              select        455
               write        391
              writev        335
               futex        303
             recvmsg        251
              socket        137
       clock_gettime        124
      rt_sigprocmask        121
              sendto        120
           setitimer        106
                stat        90
                time        81
           sigreturn        72
               fstat        66
------------------------------------------------------------
```

40.6. SYSTEMTAP으로 프로세스당 시스템 호출 볼륨 추적

syscalls_by_proc.stp SystemTap 스크립트를 사용하여 가장 많은 시스템 호출 볼륨을 수행하는 프로세스를 확인할 수 있습니다. 대부분의 시스템 호출을 수행하는 **20**개의 프로세스를 표시합니다.

사전 요구 사항

- **Systemtap** 설치에 설명된 대로 **SystemTap**을 설치했습니다.

절차

- **syscalls_by_proc.stp** 스크립트를 실행합니다.

```
# stap --example syscalls_by_proc.stp
```

syscalls_by_proc.stp 스크립트의 출력은 다음과 유사합니다.

```
Collecting data... Type Ctrl-C to exit and display results
#SysCalls  Process Name
1577       multiload-apple
692        synergyc
408        pcscd
376        mixer_applet2
299        gnome-terminal
```

```
293       Xorg
206       scim-panel-gtk
95        gnome-power-man
90        artsd
85        dhcdbd
84        scim-bridge
78        gnome-screensav
66        scim-launcher
[...]
```

41장. SYSTEMTAP을 사용하여 디스크 및 I/O 활동 모니터링

다음 섹션에서는 디스크 및 I/O 활동을 모니터링하는 스크립트를 보여줍니다.

41.1. SYSTEMTAP으로 디스크 읽기/쓰기 트래픽 요약

disktop.stp SystemTap 스크립트를 사용하여 시스템에서 가장 많은 디스크 읽기 및 쓰기를 수행하는 프로세스를 식별할 수 있습니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- **disktop.stp** 스크립트를 실행합니다.

```
# stap --example disktop.stp
```

이 스크립트는 가장 큰 읽기 또는 디스크에 대한 쓰기를 담당하는 상위 10개의 프로세스를 표시합니다.

출력에는 나열된 프로세스별로 다음 데이터가 포함됩니다.

UID

사용자 ID. 사용자 ID 0 은 root 사용자를 나타냅니다.

PID

나열된 프로세스의 ID입니다.

PPID

나열된 프로세스 상위 프로세스의 프로세스 ID입니다.

CMD

나열된 프로세스의 이름입니다.

장치

나열된 프로세스가 읽고 있는 스토리지 장치는 무엇입니까.

T

*나열된 프로세스에서 수행한 작업 유형입니다. 여기서 **W** 는 쓰기를 나타내고 **R** 은 읽기를 나타냅니다.*

바이트

디스크에 읽거나 쓰는 데이터의 양입니다.

***disktop.stp** 스크립트의 출력은 다음과 유사합니다.*

```
[...]
Mon Sep 29 03:38:28 2008 , Average:  19Kb/sec, Read: 7Kb, Write: 89Kb
UID     PID    PPID                CMD   DEVICE    T    BYTES
0    26319   26294             firefox     sda5    W       90229
0     2758    2757      pam_timestamp_c     sda5    R        8064
0     2885       1             cupsd      sda5    W        1678
Mon Sep 29 03:38:38 2008 , Average:   1Kb/sec, Read: 7Kb, Write: 1Kb
UID     PID    PPID                CMD   DEVICE    T    BYTES
0     2758    2757      pam_timestamp_c     sda5    R        8064
0     2885       1             cupsd      sda5    W        1678
```

41.2. SYSTEMTAP으로 읽기 또는 쓰기 각 파일의 I/O 시간 추적

***iotime.stp SystemTap** 스크립트를 사용하여 각 프로세스에서 모든 파일에서 읽거나 쓰는 데 걸리는 시간을 모니터링할 수 있습니다. 이렇게 하면 시스템에서 로드할 속도가 느린 파일을 결정하는 데 도움이 됩니다.*

사전 요구 사항

- ***Systemtap** 설치에 설명된 대로 **SystemTap**을 설치했습니다.*

절차

- ***iotime.stp** 스크립트를 실행합니다.*

```
# stap --example iotime.stp
```

이 스크립트는 시스템 호출이 열리고, 닫히고, 읽고, 파일에 쓸 때마다 추적합니다. 파일마다 시스템 호출 액세스마다 모든 읽기 또는 쓰기가 완료되는 데 걸리는 마이크로초 수를 계산하고 바이트 단위로 읽기 또는 파일에 기록되는 데이터 양을 추적합니다.

출력에는 다음이 포함됩니다.

- *타임스탬프(마이크로초)*
- *프로세스 **ID** 및 프로세스 이름*
- *액세스 또는 **iotime** 플래그*
- *액세스한 파일*

*프로세스가 데이터를 읽거나 쓸 수 있는 경우 한 쌍의 액세스 및 **iotime** 줄이 함께 표시되어야 합니다. 액세스 줄은 지정된 프로세스가 파일에 액세스하기 시작한 시간을 나타냅니다. 액세스 줄의 끝에는 읽기 또는 작성된 데이터의 양이 표시됩니다. **iotime** 행에는 읽기 또는 쓰기 작업을 위해 프로세스가 걸리는 시간(마이크로초)이 표시됩니다.*

***iotime.stp** 스크립트의 출력은 다음과 유사합니다.*

```
[...]
825946 3364 (NetworkManager) access /sys/class/net/eth0/carrier read: 8190 write: 0
825955 3364 (NetworkManager) iotime /sys/class/net/eth0/carrier time: 9
[...]
117061 2460 (pcscd) access /dev/bus/usb/003/001 read: 43 write: 0
117065 2460 (pcscd) iotime /dev/bus/usb/003/001 time: 7
[...]
3973737 2886 (sendmail) access /proc/loadavg read: 4096 write: 0
3973744 2886 (sendmail) iotime /proc/loadavg time: 11
[...]
```

41.3. SYSTEMTAP을 사용하여 누적 I/O 추적

***traceio.stp SystemTap** 스크립트를 사용하여 시스템에 대한 누적 **I/O** 양을 추적할 수 있습니다.*

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- **traceio.stp** 스크립트를 실행합니다.

```
# stap --example traceio.stp
```

스크립트는 시간이 지남에 따라 **I/O** 트래픽을 생성하는 상위 **10**개의 실행 파일을 출력합니다. 또한 누적 **I/O** 읽기 및 실행 파일에서 수행한 쓰기를 추적합니다. 이 정보는 **1**초 간격으로 추적 및 출력되며 내림차순으로 표시됩니다.

traceio.stp 스크립트의 출력은 다음과 유사합니다.

```
[...]
          Xorg r:   583401 KiB w:        0 KiB
      floaters r:       96 KiB w:     7130 KiB
multiload-apple r:      538 KiB w:      537 KiB
          sshd r:       71 KiB w:       72 KiB
pam_timestamp_c r:      138 KiB w:        0 KiB
       staprun r:       51 KiB w:       51 KiB
        snmpd r:       46 KiB w:        0 KiB
        pcscd r:       28 KiB w:        0 KiB
    irqbalance r:       27 KiB w:        4 KiB
        cupsd r:        4 KiB w:       18 KiB
          Xorg r:   588140 KiB w:        0 KiB
      floaters r:       97 KiB w:     7143 KiB
multiload-apple r:      543 KiB w:      542 KiB
          sshd r:       72 KiB w:       72 KiB
pam_timestamp_c r:      138 KiB w:        0 KiB
       staprun r:       51 KiB w:       51 KiB
        snmpd r:       46 KiB w:        0 KiB
        pcscd r:       28 KiB w:        0 KiB
    irqbalance r:       27 KiB w:        4 KiB
        cupsd r:        4 KiB w:       18 KiB
```

41.4. SYSTEMTAP을 사용하여 특정 장치에서 I/O 활동 모니터링

traceio2.stp SystemTap 스크립트를 사용하여 특정 장치의 **I/O** 활동을 모니터링할 수 있습니다.

사전 요구 사항

- *Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.*

절차

- ***traceio2.stp** 스크립트를 실행합니다.*

```
# stap --example traceio2.stp 'argument'
```

이 스크립트는 전체 장치 번호를 인수로 사용합니다. 이 번호를 찾으려면 다음을 사용할 수 있습니다.

```
# stat -c "0x%D" directory
```

모니터링할 장치에 디렉터리 가 위치합니다.

출력에는 다음이 포함됩니다.

- *읽기 또는 쓰기를 수행하는 프로세스의 이름 및 **ID***
- *수행 중인 기능(**vfs_read** 또는 **vfs_write**)*
- *커널 장치 번호*

***# stap traceio2.stp 0x805**의 출력을 고려하십시오.*

```
[...]
synergyc(3722) vfs_read 0x800005
synergyc(3722) vfs_read 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
[...]
```

41.5. SYSTEMTAP으로 파일 읽기 및 쓰기 모니터링

***inodewatch.stp SystemTap** 스크립트를 사용하여 실시간으로 파일의 읽기 및 쓰기를 모니터링할 수*

있습니다.

사전 요구 사항

- Systemtap 설치에 설명된 대로 SystemTap을 설치했습니다.

절차

- inodewatch.stp 스크립트를 실행합니다.

```
# stap --example inodewatch.stp 'argument1' 'argument2' 'argument3'
```

script inodewatch.stp 에는 세 가지 명령줄 인수가 사용됩니다.

1. 파일의 주요 장치 번호입니다.
2. 파일의 마이너 장치 번호입니다.
3. 파일의 inode 번호입니다.

다음 번호를 사용하여 가져올 수 있습니다.

```
# stat -c '%D %i' filename
```

여기서 filename 은 절대 경로입니다.

다음 예를 고려하십시오.

```
# stat -c '%D %i' /etc/crontab
```

출력은 다음과 같아야 합니다.

```
805 1078319
```

다음과 같습니다.

- **805** 는 **base-16 (hexadecimal)** 장치 번호입니다. 마지막 두 자리 숫자는 마이너 장치 번호입니다. 나머지 숫자는 주요 번호입니다.
- **1078319** 는 **inode** 번호입니다.

/etc/crontab 을 모니터링하려면 다음을 실행합니다.

```
# stap inodewatch.stp 0x8 0x05 1078319
```

처음 두 개의 인수에서는 **base-16** 숫자에 **0x** 접두사를 사용해야 합니다.

출력에는 다음이 포함됩니다.

- 읽기 또는 쓰기를 수행하는 프로세스의 이름 및 **ID**
- 수행 중인 기능(**vfs_read** 또는 **vfs_write**)
- 커널 장치 번호

이 예제의 출력은 다음과 같아야 합니다.

```
cat(16437) vfs_read 0x800005/1078319
cat(16437) vfs_read 0x800005/1078319
```