Red Hat Enterprise Linux 8

システムの状態とパフォーマンスの監視と管理

システムのスループット、レイテンシー、および電力消費の最適化

Last Updated: 2024-06-26

Red Hat Enterprise Linux 8 システムの状態とパフォーマンスの監視と管理

システムのスループット、レイテンシー、および電力消費の最適化

法律上の通知

概要

さまざまなシナリオで Red Hat Enterprise Linux 8 のスループット、レイテンシー、消費電力を監視して最適化します。

目次

RED HAT ドキュメントへのフィードバック (英語のみ) 8

第1章 パフォーマンス監視オプションの概要 9

第2章 TUNED を使い始める 11
2.1. TUNED の目的 11
2.2. TUNED プロファイル 11
2.3. デフォルトの TUNED プロファイル 12
2.4. マージされた TUNED プロファイル 12
2.5. TUNED プロファイルの場所 13
2.6. RHEL とともに配布される TUNED プロファイル 13
2.7. TUNED CPU-PARTITIONING プロファイル 15
2.8. 低レイテンシーチューニングへの TUNED の CPU-PARTITIONING プロファイルの使用 17
2.9. CPU-PARTITIONING TUNED プロファイルのカスタマイズ 17
2.10. RHEL とともに配布されるリアルタイムの TUNED プロファイル 18
2.11. TUNED の静的および動的チューニング 19
2.12. TUNED の NO-DAEMON モード 19
2.13. TUNED のインストールと有効化 20
2.14. 利用可能な TUNED プロファイルのリスト表示 21
2.15. TUNED プロファイルの設定 21
2.16. TUNED D-BUS インターフェイスの使用 22
2.17. TUNED の無効化 24

第3章 TUNED プロファイルのカスタマイズ 25
3.1. TUNED プロファイル 25
3.2. デフォルトの TUNED プロファイル 25
3.3. マージされた TUNED プロファイル 26
3.4. TUNED プロファイルの場所 26
3.5. TUNED プロファイル間の継承 27
3.6. TUNED の静的および動的チューニング 27
3.7. TUNED プラグイン 28
3.8. 利用可能な TUNED プラグイン 30
3.9. SCHEDULER TUNED プラグインの機能 35
3.10. TUNED プロファイルの変数 40
3.11. TUNED プロファイルの組み込み関数 41
3.12. TUNED プロファイルで利用可能な組み込み関数 42
3.13. 新しい TUNED プロファイルの作成 43
3.14. 既存の TUNED プロファイルの変更 44
3.15. TUNED を使用したディスクスケジューラーの設定 45

第4章 TUNA インターフェイスを使用したシステムの確認 48
4.1. TUNA ツールのインストール 48
4.2. TUNA ツールを使用したシステムステータスの表示 48
4.3. TUNA ツールを使用した CPU の調整 49
4.4. TUNA ツールを使用した IRQ のチューニング 51

第5章 RHEL システムロールを使用したパフォーマンスの監視 53
5.1. RHEL システムロールを使用するためのコントロールノードと管理対象ノードの準備 53
5.2. METRICS RHEL システムロールの概要 58
5.3. METRICS RHEL システムロールを使用して視覚的にローカルシステムを監視する 58
5.4. METRICS RHEL システムロールを使用して自己監視するようにシステム群を設定する 59
5.5. METRICS RHEL システムロールを使用して、ローカルマシンからマシン群を集中的に監視する 60

5.6. METRICS RHEL システムロールを使用してシステムを監視しながら認証を設定する 61
5.7. METRICS RHEL システムロールを使用して SQL SERVER のメトリクス収集を設定して有効にする 63

第6章 PCP の設定 65
6.1. PCP の概要 65
6.2. PCP のインストールおよび有効化 65
6.3. 最小限の PCP 設定のデプロイメント 66
6.4. PCP で配布されるシステムサービスおよびツール 67
6.5. PCP デプロイメントのアーキテクチャー 70
6.6. 推奨されるデプロイメントアーキテクチャー 74
6.7. サイジングファクター 74
6.8. PCP スケーリングの設定オプション 75
6.9. 例: 集中ロギングデプロイメントの分析 76
6.10. 例: 統合型セットアップデプロイメントの分析 77
6.11. 高メモリー使用率のトラブルシューティング 77

第7章 PMLOGGER でのパフォーマンスデータのロギング 80
7.1. PMLOGCONF で PMLOGGER 設定ファイルの変更 80
7.2. PMLOGGER の設定ファイルの手動編集 80
7.3. PMLOGGER サービスの有効化 81
7.4. メトリクス収集のためのクライアントシステムの設定 82
7.5. データ収集用の中央サーバーの設定 83
7.6. SYSTEMD ユニットと PMLOGGER 84
7.7. PMREP で PCP ログアーカイブの再生 87

第8章 PERFORMANCE CO-PILOT によるパフォーマンスの監視 89
8.1. PMDA-POSTFIX での POSTFIX の監視 89
8.2. PCP CHARTS アプリケーションで PCP ログアーカイブを視覚的にトレース 90
8.3. PCP を使用した SQL SERVER からのデータの収集 92

第9章 PCP を使用した XFS のパフォーマンス分析 95
9.1. XFS PMDA の手動インストール 95
9.2. PMINFO を使用した XFS パフォーマンスメトリックの検証 96
9.3. PMSTORE を使用した XFS パフォーマンスメトリックのリセット 97
9.4. XFS の PCP メトリックグループ 98
9.5. XFS のデバイスごとの PCP メトリックグループ 100

第10章 PCP メトリックのグラフィカル表示の設定 102
10.1. PCP の PCP-ZEROCONF での設定 102
10.2. GRAFANA-SERVER の設定 102
10.3. GRAFANA WEB UI へのアクセス 103
10.4. PCP REDIS の設定 105
10.5. PCP REDIS データソースでのパネルおよびアラートの作成 106
10.6. アラートの通知チャネルの追加 108
10.7. PCP コンポーネント間の認証の設定 109
10.8. PCP BPFTRACE のインストール 111
10.9. PCP BPFTRACE システム分析ダッシュボードの表示 112
10.10. PCP VECTOR のインストール 113
10.11. PCP VECTOR CHECKLIST の表示 114
10.12. GRAFANA のヒートマップの使用 115
10.13. GRAFANA に関する問題のトラブルシューティング 116

第11章 WEB コンソールを使用したシステムパフォーマンスの最適化 119
11.1. WEB コンソールでのパフォーマンスチューニングオプション 119
11.2. WEB コンソールでのパフォーマンスプロファイルの設定 119

11.3. WEB コンソールを使用したローカルシステムのパフォーマンスの監視 120
11.4. WEB コンソールと GRAFANA を使用して複数のシステムのパフォーマンスを監視する 122

第12章 ディスクスケジューラーの設定 125
12.1. 利用可能なディスクスケジューラー 125
12.2. 各種ユースケースで異なるディスクスケジューラー 126
12.3. デフォルトのディスクスケジューラー 126
12.4. アクティブなディスクスケジューラーの決定 126
12.5. TUNED を使用したディスクスケジューラーの設定 127
12.6. UDEV ルールを使用したディスクスケジューラーの設定 129
12.7. 特定ディスクに任意のスケジューラーを一時的に設定 130

第13章 SAMBA サーバーのパフォーマンスチューニング 131
13.1. SMB プロトコルバージョンの設定 131
13.2. 大量のファイルを含むディレクトリーとの共有の調整 131
13.3. パフォーマンスが低下する可能性のある設定 132

第14章 仮想マシンのパフォーマンスの最適化 133
14.1. 仮想マシンのパフォーマンスに影響を及ぼすもの 133
14.2. TUNED を使用した仮想マシンのパフォーマンスの最適化 134
14.3. 仮想マシンのメモリーの設定 135
14.4. 仮想マシンの I/O パフォーマンスの最適化 138
14.5. 仮想マシンの CPU パフォーマンスの最適化 141
14.6. 仮想マシンのネットワークパフォーマンスの最適化 152
14.7. 仮想マシンのパフォーマンス監視ツール 153
14.8. 関連情報 155

第15章 電源管理の重要性 156
15.1. 電源管理の基本 156
15.2. 監査および分析の概要 157
15.3. 監査用ツール 158

第16章 POWERTOP を使用した電力消費の管理 161
16.1. POWERTOP の目的 161
16.2. POWERTOP の使用 161
16.3. POWERTOP の統計 162
16.4. POWERTOP で FREQUENCY STATS に値が表示されない場合がある理由 164
16.5. HTML 出力の生成 165
16.6. 電力消費の最適化 165

第17章 CPU 周波数を調整してエネルギー消費を最適化 167
17.1. 対応している CPUPOWER ツールコマンド 167
17.2. CPU アイドル状態 168
17.3. CPUFREQ の概要 169

第18章 PERF の使用 174
18.1. PERF の概要 174
18.2. PERF のインストール 174
18.3. 一般的な PERF コマンド 174

第19章 PERF TOP を使用した、リアルタイムでの CPU 使用率のプロファイリング 176
19.1. PERF TOP の目的 176
19.2. PERF TOP を使用した CPU 使用率のプロファイリング 176
19.3. PERF TOP 出力の解釈 177
19.4. PERF が一部の関数名を RAW 関数アドレスとして表示する理由 177

19.5. デバッグおよびソースのリポジトリーの有効化 177
19.6. GDB を使用したアプリケーションまたはライブラリーの DEBUGINFO パッケージの取得 178

第20章 PERF STAT を使用したプロセス実行中のイベントのカウント 180
20.1. PERF STAT の目的 180
20.2. PERF STAT を使用したイベントのカウント 180
20.3. PERF STAT 出力の解釈 181
20.4. 実行中のプロセスに PERF STAT を割り当てる 182

第21章 PERF によるパフォーマンスプロファイルの記録および分析 183
21.1. PERF RECORD の目的 183
21.2. ROOT アクセスなしのパフォーマンスプロファイルの記録 183
21.3. ROOT アクセスによるパフォーマンスプロファイルの記録 183
21.4. CPU ごとのモードでのパフォーマンスプロファイルの記録 184
21.5. PERF レコードで呼び出し先のデータを取得する 184
21.6. PERF レポートを使用した PERF.DATA の分析 185
21.7. PERF REPORT 出力の解釈 186
21.8. 別のデバイスで読み取り可能な PERF.DATA ファイルの生成 186
21.9. 別のデバイスで作成された PERF.DATA ファイルの分析 187
21.10. PERF が一部の関数名を RAW 関数アドレスとして表示する理由 188
21.11. デバッグおよびソースのリポジトリーの有効化 188
21.12. GDB を使用したアプリケーションまたはライブラリーの DEBUGINFO パッケージの取得 189

第22章 PERF を使用したビジーな CPU の調査 191
22.1. PERF STAT でカウントされた CPU イベントの表示 191
22.2. PERF レポートを使用して実行した CPU サンプルの表示 191
22.3. PERF TOP を使用したプロファイリング中の特定の CPU の表示 192
22.4. PERF レコードと PERF レポートを使用した特定 CPU の監視 192

第23章 PERF でアプリケーションパフォーマンスの監視 194
23.1. 実行中のプロセスに PERF レコードを割り当てる 194
23.2. PERF レコードで呼び出し先のデータを取得する 194
23.3. PERF レポートを使用した PERF.DATA の分析 195

第24章 PERF を使用した UPROBE の作成 197
24.1. PERF を使用した関数レベルでのプローブの作成 197
24.2. PERF を使用した関数内の行でのアップローブの作成 197
24.3. UPROBE に記録されたデータのスクリプト出力を実行する 198

第25章 PERF MEM によるメモリーアクセスのプロファイリング 200
25.1. PERF MEM の目的 200
25.2. PERF MEM によるメモリーアクセスのサンプリング 200
25.3. PERF NEN 出力の解釈 202

第26章 偽共有の検出 204
26.1. PERF C2C の目的 204
26.2. PERF C2C でキャッシュライン競合の検出 204
26.3. PERF C2C レコードで記録された PERF.DATA ファイルの可視化 205
26.4. PERF C2C 出力の解釈 207
26.5. PERF C2C を使用した偽共有の検出 208

第27章 FLAMEGRAPHS の使用 211
27.1. FLAMEGRAPHS のインストール 211
27.2. システム全体でのフレームグラフの作成 211
27.3. 特定プロセスにおけるフレームグラフの作成 212

27.4. FLAMEGRAPHS の解釈 213

第28章 PERF CIRCULAR バッファーを使用したパフォーマンスのボトルネックの監視 215
28.1. PERF を使用した循環バッファーおよびイベント固有のスナップショット 215
28.2. PERF 循環バッファーを使用したパフォーマンスのボトルネックを監視するための特定のデータの収集 215

第29章 PERF を停止または再起動せずに、実行中の PERF コレクターからトレースポイントを追加および削除する 217
29.1. PERF を停止または再起動せずに、実行中の PERF コレクターにトレースポイントを追加する 217
29.2. PERF を停止または再起動せずに、実行中の PERF コレクターからトレースポイントを削除する 218

第30章 NUMASTAT を使用したメモリー割り当てのプロファイリング 219
30.1. デフォルトの NUMASTAT 統計 219
30.2. NUMASTAT を使用したメモリー割り当ての表示 219

第31章 CPU 使用率を最適化するためのオペレーティングシステムの設定 221
31.1. プロセッサーの問題を監視および診断するためのツール 221
31.2. システムトポロジーの種類 222
31.3. カーネルティック時間の設定 224
31.4. 割り込み要求の概要 226

第32章 スケジューリングポリシーの調整 229
32.1. スケジューリングポリシーの分類 229
32.2. SCHED_FIFO を使用した静的優先度スケジューリング 229
32.3. SCHED_RR を使用したラウンドロビン優先度スケジューリング 230
32.4. SCHED_OTHER を使用した通常のスケジューリング 230
32.5. スケジューラーポリシーの設定 230
32.6. CHRT コマンドのポリシーオプション 232
32.7. ブートプロセス中のサービス優先度の変更 232
32.8. 優先順位マップ 233
32.9. TUNED CPU-PARTITIONING プロファイル 234
32.10. 低レイテンシーチューニングへの TUNED の CPU-PARTITIONING プロファイルの使用 235
32.11. CPU-PARTITIONING TUNED プロファイルのカスタマイズ 236

第33章 I/O およびファイルシステムパフォーマンスに影響を与える要因 238
33.1. I/O およびファイルシステムの問題を監視および診断するツール 238
33.2. ファイルシステムのフォーマットに利用可能なチューニングオプション 240
33.3. ファイルシステムのマウントに利用可能なチューニングオプション 241
33.4. 未使用ブロックの破棄の種類 242
33.5. ソリッドステートディスク (SSD) の調整に関する考慮事項 242
33.6. 汎用ブロックデバイスのチューニングパラメーター 243

第34章 ネットワークパフォーマンスのチューニング 245
34.1. ネットワークアダプター設定のチューニング 245
34.2. IRQ バランシングのチューニング 249
34.3. ネットワーク遅延の改善 252
34.4. 大量の連続したデータストリームのスループットの向上 256
34.5. 高スループットのための TCP 接続のチューニング 259
34.6. UDP 接続のチューニング 264
34.7. アプリケーション読み取りソケットバッファーのボトルネックの特定 269
34.8. 大量のリクエストを受信するアプリケーションのチューニング 271
34.9. リッスンキューのロック競合の回避 273
34.10. デバイスドライバーと NIC のチューニング 277
34.11. ネットワークアダプターのオフロード設定 280
34.12. 割り込み結合設定のチューニング 282

34.13. TCP タイムスタンプの利点 286
34.14. イーサネットネットワークのフロー制御 287

第35章 メモリーアクセスを最適化するためにオペレーティングシステムの設定 289
35.1. システムメモリーの問題を監視および診断するツール 289
35.2. システムのメモリーの概要 290
35.3. 仮想メモリーパラメーター 290
35.4. ファイルシステムパラメーター 293
35.5. カーネルパラメーター 294
35.6. メモリー関連のカーネルパラメーターの設定 294

第36章 HUGE PAGE の設定 296
36.1. 利用可能な HUGE PAGE の機能 296
36.2. 起動時に HUGETLB ページを確保するためのパラメーター 297
36.3. 起動時の HUGETLB の設定 297
36.4. ランタイム時に HUGETLB ページを確保するためのパラメーター 299
36.5. ランタイム時の HUGETLB の設定 300
36.6. 透過的な HUGEPAGE の有効化 301
36.7. 透過的な HUGE PAGE の無効化 301
36.8. 翻訳されたバッファーサイズのページサイズの影響 302

第37章 SYSTEMTAP の使用 303
37.1. SYSTEMTAP の目的 303
37.2. SYSTEMTAP のインストール 303
37.3. SYSTEMTAP を実行する特権 304
37.4. SYSTEMTAP スクリプトの実行 305

第38章 SYSTEMTAP のクロスインストルメンテーション 306
38.1. SYSTEMTAP のクロスインストルメンテーション 306
38.2. SYSTEMTAP のクロスインストルメンテーションの初期化 307

第39章 SYSTEMTAP でのネットワークアクティビティーの監視 309
39.1. SYSTEMTAP でのネットワークアクティビティーのプロファイル 309
39.2. SYSTEMTAP でネットワークソケットコードで呼び出される関数のトレース 310
39.3. SYSTEMTAP でのネットワークパケットドロップの監視 310

第40章 SYSTEMTAP でのカーネルアクティビティーのプロファイル 312
40.1. SYSTEMTAP での関数呼び出しのカウント 312
40.2. SYSTEMTAP での関数呼び出しのトレース 313
40.3. SYSTEMTAP でカーネルとユーザー空間で費やす時間の決定 314
40.4. SYSTEMTAP を使用したポーリングアプリケーションの監視 315
40.5. SYSTEMTAP で最も頻繁に使用されるシステムコールの追跡 316
40.6. SYSTEMTAP を使用したプロセスごとのシステムコールボリュームの追跡 316

第41章 SYSTEMTAP でのディスクおよび I/O アクティビティーの監視 318
41.1. SYSTEMTAP でのディスクの読み取り/書き込みトラフィックの概要 318
41.2. SYSTEMTAP での各ファイルの読み取りまたは書き込みの I/O 時間の追跡 319
41.3. SYSTEMTAP で累積 I/O の追跡 320
41.4. SYSTEMTAP を使用した、特定デバイスでの I/O アクティビティーの監視 320
41.5. SYSTEMTAP を使用したファイルの読み取りと書き込みの監視 321

第42章 BPF コンパイラーコレクションでシステムパフォーマンスの分析 323
42.1. BCC-TOOLS パッケージのインストール 323
42.2. BCC-TOOLS でパフォーマンスの分析 323

RED HAT ドキュメントへのフィードバック (英語のみ)

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

Jira からのフィードバック送信 (アカウントが必要)

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

第1章 パフォーマンス監視オプションの概要

以下は、Red Hat Enterprise Linux 8 で利用可能なパフォーマンス監視および設定ツールの一部です。

- Performance Co-Pilot (**pcp**) は、システムレベルのパフォーマンス測定の監視、視覚化、保存、および分析に使用されます。これにより、リアルタイムデータの監視および管理、および履歴データのログと取得が可能になります。

- Red Hat Enterprise Linux 8 は、ランレベル **5** 以外のシステムを監視するためにコマンドラインから使用できる複数のツールを提供します。以下は、ビルトインのコマンドラインツールです。

  - **top** は、**procps-ng** パッケージで提供されます。これにより、実行中のシステムのプロセスの動的ビューが提供されます。システムの概要や Linux カーネルが現在管理しているタスクのリストなど、さまざまな情報が表示されます。

  - **ps** は **procps-ng** パッケージで提供されます。これは、アクティブなプロセスの選択したグループのスナップショットをキャプチャーします。デフォルトでは、検査されたグループは、現在のユーザーが所有し、**ps** コマンドが実行される端末に関連付けられているプロセスに制限されます。

  - 仮想メモリーの統計 (**vmstat**) は、**procps-ng** パッケージで提供されます。システムのプロセス、メモリー、ページング、ブロックの入出力、割り込み、および CPU アクティビティーの即時レポートを提供します。

  - System activity reporter (**sar**) は **sysstat** パッケージで提供されます。過去に発生したシステムアクティビティーに関する情報を収集し、報告します。

- **perf** は、ハードウェアパフォーマンスカウンターとカーネルトレースポイントを使用して、システム上の他のコマンドやアプリケーションの影響を追跡します。

- **bcc-tools** は BPF コンパイラーコレクション (BCC) に使用され ます。これは、カーネルアクティビティーを監視する 100 を超える **eBPF** スクリプトを提供します。各ツールの詳細は、ツールの使用方法と、ツールが実行する機能について説明する man ページを参照してください。

- **turbostat** は **kernel-tools** パッケージで提供されます。Intel 64 プロセッサーのプロセッサートポロジー、周波数、アイドル時の電力状態の統計、温度、および電力使用量について報告します。

- **iostat** は **sysstat** パッケージで提供されます。管理者が物理ディスク間で IO 負荷のバランスを取る方法を決定できるように、システム IO デバイスのロードを監視および報告します。

- **irqbalance** は、システムパフォーマンスを改善するために、複数のプロセッサーにハードウェア割り込みを分散します。

- **ss** はソケットに関する統計情報を出力するため、管理者は時間とともにデバイスのパフォーマンスを評価することができます。Red Hat は、Red Hat Enterprise Linux 8 で **ss** over **netstat** を使用することを推奨します。

- **numastat** は **numactl** パッケージで提供されます。デフォルトでは、**numastat** は、カーネルメモリーアロケーターからノードごとの NUMA ヒットしたシステム統計を表示します。最適なパフォーマンスは、高い **numa_hit** 値および低い **numa_miss** 値によって示されます。

- **numad** は NUMA アフィニティーの自動管理デーモンです。NUMA リソースの割り当て、管理、システムのパフォーマンスを動的に改善するシステム内の NUMA トポロジーとリソースの使用状況を監視します。

- **SystemTap** は、特にカーネルアクティビティーなど、オペレーティングシステムのアクティビティーを監視および分析します。
- **valgrind** は、アプリケーションを合成 CPU で実行し、実行中の既存のアプリケーションコードをインストルメント化してアプリケーションを分析します。次に、アプリケーション実行に関連する各プロセスをユーザー指定のファイル、ファイル記述子、またはネットワークソケットに明確に識別するコメントを出力します。また、メモリーリークを見つける場合にも便利です。
- **pqos** は **intel-cmt-cat** パッケージで提供されます。最新の Intel プロセッサーで CPU キャッシュとメモリー帯域幅を監視および制御します。

関連情報

- **pcp**、**top**、**ps**、**vmstat**、**sar**、**perf**、**iostat**、**irqbalance**、**ss**、**numastat**、**numad**、**valgrind**、および **pqos** の man ページ
- **/usr/share/doc/** ディレクトリー
- What exactly is the meaning of value "await" reported by iostat?(Red Hat ナレッジベースのアーティクル記事)
- Performance Co-Pilot によるパフォーマンスの監視

第2章 TUNED を使い始める

システム管理者は、TuneD アプリケーションを使用して、さまざまなユースケースに合わせてシステムのパフォーマンスプロファイルを最適化できます。

2.1. TUNED の目的

TuneD は、システムを監視し、特定のワークロードでパフォーマンスを最適化するサービスです。TuneD の中核となるのは、さまざまなユースケースに合わせてシステムをチューニングする **プロファイル** です。

TuneD には、以下のようなユースケース用に定義されたプロファイルが多数同梱されています。

- 高スループット
- 低レイテンシー
- 節電

各プロファイル向けに定義されたルールを変更し、特定のデバイスのチューニング方法をカスタマイズできます。別のプロファイルに切り替えたり、TuneD を非アクティブにすると、以前のプロファイルによるシステム設定への変更はすべて、元の状態に戻ります。

また、TuneD を設定してデバイスの使用状況の変化に対応し、設定を調整して、アクティブなデバイスのパフォーマンスを向上させ、非アクティブなデバイスの消費電力を削減することもできます。

2.2. TUNED プロファイル

システムを詳細に分析することは、非常に時間のかかる作業です。TuneD では、一般的なユースケースに合わせて定義済みのプロファイルを多数提供しています。プロファイルを作成、変更、および削除することも可能です。

TuneD で提供されるプロファイルは、以下のカテゴリーに分類されます。

- 省電力プロファイル
- パフォーマンス重視プロファイル

performance-boosting プロファイルの場合は、次の側面に焦点が置かれます。

- ストレージおよびネットワークに対して少ないレイテンシー
- ストレージおよびネットワークの高い処理能力
- 仮想マシンのパフォーマンス
- 仮想化ホストのパフォーマンス

プロファイル設定の構文
tuned.conf ファイルは、1つの **[main]** セクションとプラグインインスタンスを設定するためのその他のセクションが含まれます。ただし、すべてのセクションはオプションです。

ハッシュ記号 (**#**) で始まる行はコメントです。

関連情報

- **tuned.conf(5)** の man ページ

2.3. デフォルトの TUNED プロファイル

インストール時に、システムの最適なプロファイルが自動的に選択されます。現時点では、以下のカスタマイズ可能なルールに従ってデフォルトのプロファイルが選択されます。

環境	デフォルトプロファイル	目的
コンピュートノード	**throughput-performance**	最適なスループットパフォーマンス
仮想マシン	**virtual-guest**	ベストパフォーマンスベストパフォーマンスが重要でない場合は、**balanced** プロファイルまたは **powersave** プロファイルに変更できます。
その他のケース	**balanced**	パフォーマンスと電力消費の調和

関連情報

- **tuned.conf(5)** の man ページ

2.4. マージされた TUNED プロファイル

試験目的で提供された機能として、複数のプロファイルを一度に選択することができます。TuneD は、読み込み中にマージを試みます。

競合が発生した場合は、最後に指定されたプロファイルの設定が優先されます。

> **例2.1 仮想ゲストの低消費電力**
>
> 以下の例では、仮想マシンでの実行でパフォーマンスを最大化するようにシステムが最適化され、同時に、(低消費電力が最優先である場合は) 低消費電力を実現するようにシステムがチューニングされます。
>
> ```
> # tuned-adm profile virtual-guest powersave
> ```

警告

マージは自動的に行われ、使用されるパラメーターの組み合わせが適切であるかどうかはチェックされません。結果として、この機能は一部のパラメーターを逆に調整する可能性があります。これは逆効果になる可能性があります。たとえば、**throughput-performance** プロファイルで高スループットにディスクを設定し、同時に、**spindown-disk** プロファイルでディスクスピンダウンを低い値に設定します。

関連情報

__tuned-adm__ man ページ __tuned.conf(5)__ の man ページ

2.5. TUNED プロファイルの場所

TuneD は、次のディレクトリーにプロファイルを保存します。

/usr/lib/tuned/

ディストリビューション固有のプロファイルは、このディレクトリーに保存されます。各プロファイルには独自のディレクトリーがあります。プロファイルは **tuned.conf** という名前の主要設定ファイルと、ヘルパースクリプトなどの他の任意のファイルから設定されます。

/etc/tuned/

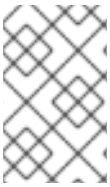
プロファイルをカスタマイズする必要がある場合は、プロファイルのカスタマイズに使用されるディレクトリーにプロファイルディレクトリーをコピーします。同じ名前のプロファイルが 2 つある場合、カスタムのプロファイルは、**/etc/tuned/** に置かれています。

関連情報

- **tuned.conf(5)** の man ページ

2.6. RHEL とともに配布される TUNED プロファイル

以下は、Red Hat Enterprise Linux に **TuneD** とともにインストールされるプロファイルのリストです。



注記

利用可能な製品固有またはサードパーティーの **TuneD** プロファイルが複数存在する可能性があります。このようなプロファイルは通常、個別の RPM パッケージで提供されます。

balanced

デフォルトの省電力プロファイル。パフォーマンスと電力消費のバランスを取ることが目的です。可能な限り、自動スケーリングと自動チューニングを使用します。唯一の欠点はレイテンシーが増加することです。今回の **TuneD** リリースでは、CPU、ディスク、オーディオ、およびビデオプラグインを有効にし、**conservative** CPU ガバナーを有効にします。**radeon_powersave** オプションは、**dpm-balanced** 値に対応している場合はその値を使用し、それ以外の場合は **auto** に設定されます。
energy_performance_preference 属性を **normal** の電力設定に変更します。また、**scaling_governor** ポリシー属性を **conservative** または **powersave** CPU ガバナーのいずれかに変更します。

powersave

省電力パフォーマンスを最大化するプロファイル。実際の電力消費を最小化するためにパフォーマンスを調整できます。今回の **TuneD** リリースでは、SATA ホストアダプターの USB 自動サスペンド、WiFi 省電力、および Aggressive Link Power Management (ALPM) の省電力を有効にします。また、ウェイクアップ率が低いシステムのマルチコア省電力がスケジュールされ、**ondemand** ガバナーがアクティブ化されます。さらに、AC97 音声省電力と、システムに応じて HDA-Intel 省電力 (10 秒のタイムアウト) が有効になります。KMS が有効なサポート対象の Radeon グラフィックカードがシステムに搭載されている場合、プロファイルは自動省電力に設定されます。ASUS Eee PC では、動的な Super Hybrid Engine が有効になります。

energy_performance_preference 属性を **powersave** または **power** 電力設定に変更します。また、**scaling_governor** ポリシー属性を **ondemand** または **powersave** CPU ガバナーのいずれかに変更します。

注記

場合によっては、**balanced** プロファイルの方が、**powersave** プロファイルよりも効率的です。

定義された量の作業を行う場合 (たとえば、動画ファイルをトランスコードする必要がある場合) を考えてください。トランスコードがフルパワーで実行される場合に、マシンの電力消費が少なくなることがあります。これは、タスクがすぐに完了し、マシンがアイドル状態になり、非常に効率的な省電力モードに自動的に切り替わることがあるためです。その一方で、調整されたマシンでファイルをトランスコードすると、マシンはトランスコード中に少ない電力を消費しますが、処理に時間がかかり、全体的な消費電力は高くなることがあります。

このため、一般的に **balanced** プロファイルが優れたオプションになる場合があります。

throughput-performance

高スループットに最適化されたサーバープロファイル。これにより、節電メカニズムが無効になり、**sysctl** が有効になるため、ディスクおよびネットワーク IO のスループットパフォーマンスが向上します。CPU ガバナーは **performance** に設定されます。
energy_performance_preference および **scaling_governor** 属性を **performance** プロファイルに変更します。

accelerator-performance

accelerator-performance プロファイルには、**throughput-performance** プロファイルと同じチューニングが含まれます。さらに、CPU を低い C 状態にロックし、レイテンシーが 100us 未満になるようにします。これにより、GPU などの特定のアクセラレーターのパフォーマンスが向上します。

latency-performance

低レイテンシーに最適化されたサーバープロファイル。省電力メカニズムが無効になり、レイテンシーを向上させる **sysctl** 設定が有効になります。CPU ガバナーは **performance** に設定され、CPU は低い C 状態にロックされます (PM QoS を使用)。
energy_performance_preference および **scaling_governor** 属性を **performance** プロファイルに変更します。

network-latency

低レイテンシーネットワークチューニング向けプロファイル。**latency-performance** プロファイルに基づきます。さらに、透過的な huge page と NUMA 分散を無効にし、他のいくつかのネットワーク関連の **sysctl** パラメーターの調整を行います。
latency-performance プロファイルを継承します。また、**energy_performance_preference** および **scaling_governor** 属性を **performance** プロファイルに変更します。

hpc-compute

高パフォーマンスコンピューティング向けに最適化されたプロファイル。**latency-performance** プロファイルに基づきます。

network-throughput

スループットネットワークチューニング向けプロファイル。**throughput-performance** プロファイルに基づきます。さらに、カーネルネットワークバッファーを増やします。
latency-performance または **throughput-performance** プロファイルのいずれかを継承します。また、**energy_performance_preference** および **scaling_governor** 属性を **performance** プロファイルに変更します。

virtual-guest

throughput-performance プロファイルに基づく Red Hat Enterprise 8 仮想マシンおよび VMWare ゲスト向けプロファイル。仮想メモリーのスワップの減少や、ディスクの readahead 値の増加などが行われます。ディスクバリアは無効になりません。
throughput-performance プロファイルを継承します。また、**energy_performance_preference** および **scaling_governor** 属性を **performance** プロファイルに変更します。

virtual-host

throughput-performance プロファイルに基づいて仮想ホスト用に設計されたプロファイル。他のタスクの中でも特に、仮想メモリーのスワップを減らし、ディスクの先読み値を増やし、ダーティーページの書き戻しというより積極的な値を可能にします。
throughput-performance プロファイルを継承します。また、**energy_performance_preference** および **scaling_governor** 属性を **performance** プロファイルに変更します。

oracle

Oracle データベース向けに最適化されたプロファイルは、**throughput-performance** プロファイルに基づいて読み込まれます。これにより Transparent Huge Page が無効になり、その他のパフォーマンス関連カーネルパラメーターが変更されます。このプロファイルは、**tuned-profiles-oracle** パッケージで利用できます。

desktop

balanced プロファイルに基づく、デスクトップに最適化されたプロファイル。対話型アプリケーションの応答を向上させるスケジューラーオートグループが有効になります。

optimize-serial-console

printk 値を減らすことで、シリアルコンソールへの I/O アクティビティーを調整するプロファイル。これにより、シリアルコンソールの応答性が向上します。このプロファイルは、他のプロファイルのオーバーレイとして使用することが意図されています。以下に例を示します。

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

mssql

Microsoft SQL Server に提供されるプロファイル。**throughput-performance** プロファイルに基づきます。

intel-sst

ユーザー定義の Intel Speed Select Technology 設定で最適化されたプロファイル。このプロファイルは、他のプロファイルのオーバーレイとして使用することが意図されています。以下に例を示します。

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

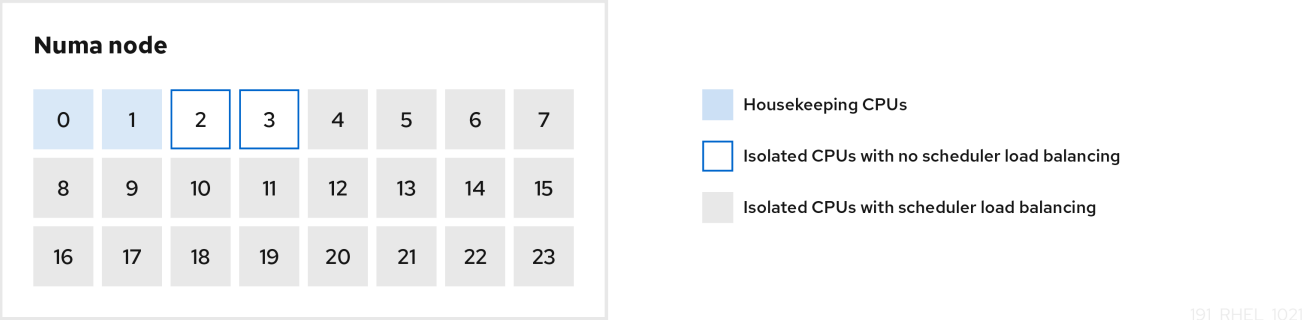
2.7. TUNED CPU-PARTITIONING プロファイル

レイテンシーに敏感なワークロード用に Red Hat Enterprise Linux 8 を調整する場合は、**cpu-partitioning** TuneD プロファイルを使用することが推奨されます。

Red Hat Enterprise Linux 8 以前では、低レイテンシーの Red Hat ドキュメントで、低レイテンシーのチューニングを実現するために必要な低レベルの手順が数多く説明されていました。Red Hat Enterprise Linux 8 では、**cpu-partitioning** TuneD プロファイルを使用することで、低レイテンシーのチューニングをより効率的に実行できます。このプロファイルは、個々の低レイテンシーアプリケーションの要件に従って簡単にカスタマイズできます。

以下の図は、**cpu-partitioning** プロファイルの使用方法を示す例になります。この例では、CPU とノードのレイアウトを使用します。

図2.1 cpu-partitioning の図



/etc/tuned/cpu-partitioning-variables.conf ファイルで cpu-partitioning プロファイルを設定するには、以下の設定オプションを使用します。

負荷分散機能のある分離された CPU

cpu-partitioning の図では、4 から 23 までの番号が付けられたブロックが、デフォルトの分離された CPU です。カーネルスケジューラーのプロセスの負荷分散は、この CPU で有効になります。これは、カーネルスケジューラーの負荷分散を必要とする複数のスレッドを使用した低レイテンシープロセス用に設計されています。

isolated_cores=cpu-list オプションを使用して、**/etc/tuned/cpu-partitioning-variables.conf** ファイルで cpu-partitioning プロファイルを設定できます。このオプションは、カーネルスケジューラーの負荷分散を使用する分離する CPU をリスト表示します。

分離された CPU のリストはコンマ区切りで表示するか、**3-5** のようにハイフンを使用して範囲を指定できます。このオプションは必須です。このリストにない CPU は、自動的にハウスキーピング CPU と見なされます。

負荷分散を行わずに分離した CPU

cpu-partitioning の図では、2 と 3 の番号が付けられたブロックは、追加のカーネルスケジューラープロセスの負荷分散を提供しない分離された CPU です。

/etc/tuned/cpu-partitioning-variables.conf ファイルで cpu-partitioning プロファイルを設定するには、**no_balance_cores=cpu-list** オプションを使用します。このオプションは、カーネルスケジューラーの負荷分散を使用しない CPU を分離するようにリスト表示します。

no_balance_cores オプションの指定は任意ですが、このリストの CPU は、**isolated_cores** リストに記載されている CPU のサブセットである必要があります。

このような CPU を使用するアプリケーションスレッドは、各 CPU に個別にピン留めする必要があります。

ハウスキーピング CPU

cpu-partitioning-variables.conf ファイル内で分離されていない CPU は、自動的にハウスキーピング CPU と見なされます。ハウスキーピング CPU では、すべてのサービス、デーモン、ユーザープロセス、移動可能なカーネルスレッド、割り込みハンドラー、およびカーネルタイマーの実行が許

可されます。

関連情報

- **tuned-profiles-cpu-partitioning(7)** man ページ

2.8. 低レイテンシーチューニングへの TUNED の CPU-PARTITIONING プロファイルの使用

この手順では、TuneD の **cpu-partitioning** プロファイルを使用して、低レイテンシーになるようにシステムをチューニングする方法を説明します。これは、cpu-partitioning の図で説明されているように、**cpu-partitioning** と CPU レイアウトを使用できる低レイテンシーのアプリケーションの例を使用します。

この場合のアプリケーションでは、以下を使用します。

- ネットワークからデータを読み込む 1 つの専用リーダースレッドが、CPU 2 に固定されます。
- このネットワークデータを処理する多数のスレッドは、CPU 4-23 に固定されます。
- 処理されたデータをネットワークに書き込む専用のライタースレッドは、CPU 3 に固定されます。

前提条件

- **yum install tuned-profiles-cpu-partitioning** コマンドを root で使用して、 **cpu-partitioning** TuneD プロファイルをインストールしている。

手順

1. **/etc/tuned/cpu-partitioning-variables.conf** ファイルを編集し、以下の内容を追加します。

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

2. **cpu-partitioning** TuneD プロファイルを設定します。

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

3. 再起動
   再起動後、システムは、cpu-partitioning の図の分離に従って、低レイテンシーにチューニングされます。このアプリケーションでは、タスクセットを使用して、リーダーおよびライターのスレッドを CPU 2 および 3 に固定し、残りのアプリケーションスレッドを CPU 4-23 に固定できます。

関連情報

- **tuned-profiles-cpu-partitioning(7)** man ページ

2.9. CPU-PARTITIONING TUNED プロファイルのカスタマイズ

TuneD プロファイルを拡張して、追加のチューニング変更を行うことができます。

たとえば、**cpu-partitioning** プロファイルは、**cstate=1** を使用する CPU を設定します。**cpu-partitioning** プロファイルを使用しながら、cstate1 から cstate0 に CPU の cstate を変更するために、以下の手順では **my_profile** という名前の新しい TuneD プロファイルを説明しています。このプロファイルは、**cpu-partitioning** プロファイルを継承した後、C state-0 を設定します。

手順

1. **/etc/tuned/my_profile** ディレクトリーを作成します。

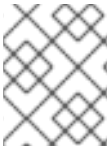
```
# mkdir /etc/tuned/my_profile
```

2. このディレクトリーに **tuned.conf** ファイルを作成し、次の内容を追加します。

```
# 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. 新しいプロファイルを使用します。

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



注記

この共有例では、再起動は必要ありません。ただし、**my_profile** プロファイルの変更を有効にするために再起動が必要な場合は、マシンを再起動します。

関連情報

- **tuned-profiles-cpu-partitioning(7)** man ページ

2.10. RHEL とともに配布されるリアルタイムの TUNED プロファイル

リアルタイムプロファイルは、リアルタイムカーネルを実行するシステムを対象としています。特殊なカーネルビルドなしでは、システムはリアルタイムになりません。RHEL では、このプロファイルは追加のリポジトリーから利用できます。

利用できるリアルタイムプロファイルは以下の通りです。

リアルタイム

ベアメタルのリアルタイムシステムで使用します。
tuned-profiles-realtime パッケージにより提供されます。これは、RT リポジトリーまたは NFV リポジトリーから入手できます。

realtime-virtual-host

リアルタイムに設定された仮想ホストで使用します。
NFV リポジトリーから利用できる **tuned-profiles-nfv-host** パッケージにより提供されます。

realtime-virtual-guest

リアルタイムに設定された仮想化ゲストで使用します。
NFV リポジトリーから利用できる **tuned-profiles-nfv-guest** パッケージにより提供されます。

2.11. TUNED の静的および動的チューニング

TuneD が適用するシステムチューニングの 2 つのカテゴリー (**static** と **dynamic**) の違いを理解することは、特定の状況や目的にどちらを使用するかを決定する際に重要です。

静的なチューニング

主に、事前定義された **sysctl** 設定および **sysfs** 設定の適用と、**ethtool** などの複数の設定ツールのワンショットアクティベーションから設定されます。

動的チューニング

システムのアップタイム中に、さまざまなシステムコンポーネントがどのように使用されているかを監視します。**TuneD** は、その監視情報に基づいてシステム設定を動的に調整します。
たとえば、ハードドライブは起動時およびログイン時に頻繁に使用されますが、Web ブラウザーや電子メールクライアントなどのアプリケーションをユーザーが主に使用する場合はほとんど使用されません。同様に、CPU とネットワークデバイスは、異なるタイミングで使用されます。**TuneD** は、このようなコンポーネントのアクティビティーを監視し、その使用の変化に反応します。

デフォルトでは、動的チューニングは無効になっています。これを有効にするには、**/etc/tuned/tuned-main.conf** ファイルを編集して、**dynamic_tuning** オプションを **1** に変更します。**TuneD** は、システムの統計を定期的に分析してから、その統計を使用してシステムのチューニング設定を更新します。これらの更新間の時間間隔を秒単位で設定するには、**update_interval** オプションを使用します。

現在実装されている動的チューニングアルゴリズムは、パフォーマンスと省電力のバランスを取ろうとし、パフォーマンスプロファイルで無効になります。各プラグインのダイナミックチューニングは、**TuneD** プロファイルで有効または無効にできます。

例2.2 ワークステーションでの静的および動的のチューニング

一般的なオフィスワークステーションでは、イーサネットネットワークインターフェイスは常に非アクティブの状態です。少数の電子メールのみが出入りするか、一部の Web ページが読み込まれている可能性があります。

このような負荷の場合、ネットワークインターフェイスはデフォルト設定のように常に最高速度で動作する必要はありません。**TuneD** には、ネットワークデバイスを監視してチューニングを行うプラグインがあり、これによりこの低いアクティビティーを検出して、自動的にそのインターフェイスの速度を下げることができるため、通常は消費電力が少なくなります。

DVD イメージをダウンロードしているとき、または大きな添付ファイル付きのメールが開いているときなど、インターフェイスのアクティビティーが長期間にわたって増加した場合は、**TuneD** がこれを検出し、アクティビティーレベルが高い間にインターフェイスの速度を最大に設定します。

この原則は、CPU およびディスクの他のプラグインにも使用されます。

2.12. TUNED の NO-DAEMON モード

TuneD は、常駐メモリーを必要としない **no-daemon** モードで実行できます。このモードでは、**TuneD** が設定を適用して終了します。

デフォルトでは、このモードには、以下のように多くの **TuneD** 機能がないため、**no-daemon** モードが無効になっています。

- D-Bus サポート
- ホットプラグサポート
- 設定のロールバックサポート

no-daemon モードを有効にするには、**/etc/tuned/tuned-main.conf** ファイルに以下の行を含めます。

```
daemon = 0
```

2.13. TUNED のインストールと有効化

この手順では、**TuneD** アプリケーションをインストールして有効にし、**TuneD** プロファイルをインストールして、システムにデフォルトの **TuneD** プロファイルをあらかじめ設定します。

手順

1. **Tuned** パッケージをインストールします。

```
# yum install tuned
```

2. **Tuned** サービスを有効にして開始します。

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

3. 必要に応じて、リアルタイムシステムの**TuneD** プロファイルをインストールします。
リアルタイムシステムの **Tuned** プロファイルの場合は、**rhel-8** リポジトリーを有効にします。

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

インストールします。

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

4. **TuneD**プロファイルが有効であり、適用されていることを確認します。

```
$ tuned-adm active

Current active profile: throughput-performance
```

注記

TuneD が自動的にプリセットするアクティブなプロファイルは、マシンのタイプとシステム設定によって異なります。

```
$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
```

> See tuned log file ('/var/log/tuned/tuned.log') for details.

2.14. 利用可能な TUNED プロファイルのリスト表示

この手順では、使用しているシステムで現在利用可能なTuneDプロファイルのリストを表示します。

手順

- システムで使用可能なすべてのTuneDプロファイルをリスト表示するには、次を使用します。

```
$ 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
```

- 現在アクティブなプロファイルのみを表示する場合は、次のコマンドを使用します。

```
$ tuned-adm active

Current active profile: throughput-performance
```

関連情報

- **tuned-adm(8)** の man ページ

2.15. TUNED プロファイルの設定

この手順では、選択した **TuneD** プロファイルを有効にします。

前提条件

- **TuneD** サービスが実行されています。詳細は、TuneD のインストールと有効化 を参照してください。

手順

1. 必要に応じて、**TuneD** がシステムに最も適したプロファイルを推奨できます。

```
# tuned-adm recommend

throughput-performance
```

2. プロファイルをアクティブ化します。

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

または、複数のプロファイルの組み合わせをアクティベートできます。

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

例2.3 低消費電力向けに最適化された仮想マシン

以下の例では、仮想マシンでの実行でパフォーマンスを最大化するようにシステムが最適化され、同時に、(低消費電力が最優先である場合は) 低消費電力を実現するようにシステムがチューニングされます。

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

3. お使いのシステムで現在アクティブな **TuneD** プロファイルを表示します。

```
# tuned-adm active

Current active profile: selected-profile
```

4. システムを再起動します。

```
# reboot
```

検証手順

- **TuneD** プロファイルが有効であり、適用されていることを確認します。

```
$ tuned-adm verify

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

関連情報

- **tuned-adm(8)** の man ページ

2.16. TUNED D-BUS インターフェイスの使用

TuneD D-Bus インターフェイスを介してランタイム時に TuneD と直接通信し、さまざまな TuneD サービスを制御できます。

D-Bus API にアクセスするには、**busctl** または **dbus-send** コマンドを使用できます。

注記

busctl コマンドまたは **dbus-send** コマンドを使用できますが、**busctl** コマンドは **systemd** の一部であるため、ほとんどのホストにすでに存在しています。

2.16.1. TuneD D-Bus インターフェイスを使用した利用可能な TuneD D-Bus API メソッドの表示

TuneD D-Bus インターフェイスを使用すると、TuneD で使用できる D-Bus API メソッドを確認できます。

前提条件

- TuneD サービスが実行されている。詳細は、TuneD のインストールと有効化 を参照してください。

手順

- 利用可能な TuneD API メソッドを確認するには、次のコマンドを実行します。

```
$ busctl introspect com.redhat.tuned /Tuned com.redhat.tuned.control
```

この出力は、以下のようになります。

```
NAME                                TYPE      SIGNATURE RESULT/VALUE FLAGS
.active_profile                     method    -         s            -
.auto_profile                       method    -         (bs)         -
.disable                            method    -         b            -
.get_all_plugins                    method    -         a{sa{ss}}    -
.get_plugin_documentation           method    s         s            -
.get_plugin_hints                   method    s         a{ss}        -
.instance_acquire_devices           method    ss        (bs)         -
.is_running                         method    -         b            -
.log_capture_finish                 method    s         s            -
.log_capture_start                  method    ii        s            -
.post_loaded_profile                method    -         s            -
.profile_info                       method    s         (bsss)       -
.profile_mode                       method    -         (ss)         -
.profiles                           method    -         as           -
.profiles2                          method    -         a(ss)        -
.recommend_profile                  method    -         s            -
.register_socket_signal_path        method    s         b            -
.reload                             method    -         b            -
.start                              method    -         b            -
.stop                               method    -         b            -
.switch_profile                     method    s         (bs)         -
.verify_profile                     method    -         b            -
.verify_profile_ignore_missing      method    -         b            -
.profile_changed                    signal    sbs       -            -
```

利用可能なさまざまなメソッドの説明は、TuneD のアップストリームリポジトリー に記載されています。

2.16.2. TuneD D-Bus インターフェイスを使用したアクティブな TuneD プロファイルの変更

TuneD D-Bus インターフェイスを使用して、アクティブな TuneD プロファイルを必要な TuneD プロファイルに置き換えることができます。

前提条件

- TuneD サービスが実行されている。詳細は、TuneD のインストールと有効化 を参照してください。

手順

- アクティブな TuneD プロファイルを変更するには、次のコマンドを実行します。

```
$ busctl call com.redhat.tuned /Tuned com.redhat.tuned.control switch_profile s profile
(bs) true "OK"
```

  profile は、必要なプロファイルの名前に置き換えます。

検証

- 現在アクティブな TuneD プロファイルを表示するには、次のコマンドを実行します。

```
$ busctl call com.redhat.tuned /Tuned com.redhat.tuned.control active_profile
s "profile"
```

2.17. TUNED の無効化

この手順では、**TuneD** を無効にし、影響を受けるすべてのシステム設定を **TuneD** が変更する前の元の状態にリセットします。

手順

- すべてのチューニングを一時的に無効にするには、次のコマンドを実行します。

```
# tuned-adm off
```

  チューニングは、**TuneD** サービスの再起動後に再度適用されます。

- または、**TuneD** サービスを完全に停止して無効にするには、次のようにします。

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

関連情報

- **tuned-adm(8)** の man ページ

第3章 TUNED プロファイルのカスタマイズ

TuneDプロファイルを作成または変更して、ユースケースに合わせてシステムパフォーマンスを最適化できます。

前提条件

- TuneD のインストールと有効化 に詳述されているように、**TuneD** をインストールおよび有効化します。

3.1. TUNED プロファイル

システムを詳細に分析することは、非常に時間のかかる作業です。**TuneD** では、一般的なユースケースに合わせて定義済みのプロファイルを多数提供しています。プロファイルを作成、変更、および削除することも可能です。

TuneD で提供されるプロファイルは、以下のカテゴリーに分類されます。

- 省電力プロファイル
- パフォーマンス重視プロファイル

performance-boosting プロファイルの場合は、次の側面に焦点が置かれます。

- ストレージおよびネットワークに対して少ないレイテンシー
- ストレージおよびネットワークの高い処理能力
- 仮想マシンのパフォーマンス
- 仮想化ホストのパフォーマンス

プロファイル設定の構文
tuned.conf ファイルは、1つの **[main]** セクションとプラグインインスタンスを設定するためのその他のセクションが含まれます。ただし、すべてのセクションはオプションです。

ハッシュ記号 (**#**) で始まる行はコメントです。

関連情報

- **tuned.conf(5)** の man ページ

3.2. デフォルトの TUNED プロファイル

インストール時に、システムの最適なプロファイルが自動的に選択されます。現時点では、以下のカスタマイズ可能なルールに従ってデフォルトのプロファイルが選択されます。

環境	デフォルトプロファイル	目的
コンピュートノード	**throughput-performance**	最適なスループットパフォーマンス

環境	デフォルトプロファイル	目的
仮想マシン	**virtual-guest**	ベストパフォーマンスベストパフォーマンスが重要でない場合は、**balanced** プロファイルまたは **powersave** プロファイルに変更できます。
その他のケース	**balanced**	パフォーマンスと電力消費の調和

関連情報

- **tuned.conf(5)** の man ページ

3.3. マージされた TUNED プロファイル

試験目的で提供された機能として、複数のプロファイルを一度に選択することができます。TuneD は、読み込み中にマージを試みます。

競合が発生した場合は、最後に指定されたプロファイルの設定が優先されます。

> **例3.1 仮想ゲストの低消費電力**
>
> 以下の例では、仮想マシンでの実行でパフォーマンスを最大化するようにシステムが最適化され、同時に、(低消費電力が最優先である場合は) 低消費電力を実現するようにシステムがチューニングされます。
>
> ```
> # tuned-adm profile virtual-guest powersave
> ```

警告

マージは自動的に行われ、使用されるパラメーターの組み合わせが適切であるかどうかはチェックされません。結果として、この機能は一部のパラメーターを逆に調整する可能性があります。これは逆効果になる可能性があります。たとえば、**throughput-performance** プロファイルで高スループットにディスクを設定し、同時に、**spindown-disk** プロファイルでディスクスピンダウンを低い値に設定します。

関連情報

__tuned-adm__ man ページ __tuned.conf(5)__ の man ページ

3.4. TUNED プロファイルの場所

TuneD は、次のディレクトリーにプロファイルを保存します。

/usr/lib/tuned/

ディストリビューション固有のプロファイルは、このディレクトリーに保存されます。各プロファイルには独自のディレクトリーがあります。プロファイルは **tuned.conf** という名前の主要設定ファイルと、ヘルパースクリプトなどの他の任意のファイルから設定されます。

/etc/tuned/

プロファイルをカスタマイズする必要がある場合は、プロファイルのカスタマイズに使用されるディレクトリーにプロファイルディレクトリーをコピーします。同じ名前のプロファイルが 2 つある場合、カスタムのプロファイルは、**/etc/tuned/** に置かれています。

関連情報

- **tuned.conf(5)** の man ページ

3.5. TUNED プロファイル間の継承

TuneDプロファイルは、他のプロファイルを基にして、親プロファイルの特定の側面のみを変更できます。

TuneD プロファイルの **[main]** セクションは、**include** オプションを認識します。

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

親 プロファイルの設定はすべて、この **子** プロファイルに読み込まれます。以下のセクションでは、**子** プロファイルは、**親** プロファイルから継承された特定の設定をオーバーライドするか、**親** プロファイルに表示されない新しい設定を追加します。

/usr/lib/tuned/ にあらかじめインストールしておいたプロファイルでパラメーターをいくつか調整するだけで、**/etc/tuned/** に独自の **子** プロファイルを作成できます。

TuneD のアップグレード後などに、**親** プロファイルが更新されると、この変更は **子** プロファイルに反映されます。

例3.2 バランスの取れた省電力プロファイル

以下は、**balanced** プロファイルを拡張し、すべてのデバイスの Aggressive Link Power Management (ALPM) を最大省電力に設定するカスタムプロファイルの例です。

```
[main]
include=balanced

[scsi_host]
alpm=min_power
```

関連情報

- **tuned.conf(5)** の man ページ

3.6. TUNED の静的および動的チューニング

TuneD が適用するシステムチューニングの 2 つのカテゴリー (**static** と **dynamic**) の違いを理解することは、特定の状況や目的にどちらを使用するかを決定する際に重要です。

静的なチューニング

主に、事前定義された **sysctl** 設定および **sysfs** 設定の適用と、**ethtool** などの複数の設定ツールのワンショットアクティベーションから設定されます。

動的チューニング

システムのアップタイム中に、さまざまなシステムコンポーネントがどのように使用されているかを監視します。**TuneD** は、その監視情報に基づいてシステム設定を動的に調整します。
たとえば、ハードドライブは起動時およびログイン時に頻繁に使用されますが、Web ブラウザーや電子メールクライアントなどのアプリケーションをユーザーが主に使用する場合はほとんど使用されません。同様に、CPU とネットワークデバイスは、異なるタイミングで使用されます。**TuneD** は、このようなコンポーネントのアクティビティーを監視し、その使用の変化に反応します。

デフォルトでは、動的チューニングは無効になっています。これを有効にするには、**/etc/tuned/tuned-main.conf** ファイルを編集して、**dynamic_tuning** オプションを **1** に変更します。**TuneD** は、システムの統計を定期的に分析してから、その統計を使用してシステムのチューニング設定を更新します。これらの更新間の時間間隔を秒単位で設定するには、**update_interval** オプションを使用します。

現在実装されている動的チューニングアルゴリズムは、パフォーマンスと省電力のバランスを取ろうとし、パフォーマンスプロファイルで無効になります。各プラグインのダイナミックチューニングは、**TuneD** プロファイルで有効または無効にできます。

例3.3 ワークステーションでの静的および動的のチューニング

一般的なオフィスワークステーションでは、イーサネットネットワークインターフェイスは常に非アクティブの状態です。少数の電子メールのみが出入りするか、一部の Web ページが読み込まれている可能性があります。

このような負荷の場合、ネットワークインターフェイスはデフォルト設定のように常に最高速度で動作する必要はありません。**TuneD** には、ネットワークデバイスを監視してチューニングを行うプラグインがあり、これによりこの低いアクティビティーを検出して、自動的にそのインターフェイスの速度を下げることができるため、通常は消費電力が少なくなります。

DVD イメージをダウンロードしているとき、または大きな添付ファイル付きのメールが開いているときなど、インターフェイスのアクティビティーが長期間にわたって増加した場合は、**TuneD** がこれを検出し、アクティビティーレベルが高い間にインターフェイスの速度を最大に設定します。

この原則は、CPU およびディスクの他のプラグインにも使用されます。

3.7. TUNED プラグイン

プラグインは、**TuneD** がシステムのさまざまなデバイスを監視または最適化するために使用する **TuneD** プロファイルのモジュールです。

TuneD では、以下の 2 つのタイプのプラグインを使用します。

プラグインの監視

モニタリングプラグインは、稼働中のシステムから情報を取得するために使用されます。監視プラグインの出力は、動的チューニング向けチューニングプラグインで使用できます。

監視プラグインは、有効ないずれかのチューニングプラグインでメトリックが必要な場合に必ず自動的にインスタンス化されます。2 つのチューニングプラグインで同じデータが必要な場合は、監視プラグインのインスタンスが 1 つだけ作成され、データが共有されます。

プラグインのチューニング

各チューニングプラグインは、個々のサブシステムをチューニングし、TuneD プロファイルから設定されたいくつかのパラメーターを取得します。各サブシステムには、チューニングプラグインの個別インスタンスで処理される複数のデバイス (複数の CPU やネットワークカードなど) を含めることができます。また、個別デバイスの特定の設定もサポートされます。

TuneD プロファイルのプラグインの構文

プラグインインスタンスが記述されるセクションは、以下のように書式化されます。

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

NAME

ログで使用されるプラグインインスタンスの名前です。これは、任意の文字列です。

TYPE

チューニングプラグインのタイプです。

DEVICES

このプラグインインスタンスが処理するデバイスのリストです。
device の行には、リスト、ワイルドカード (*)、否定 (**!**) が含まれます。**device** の行がないと、**TYPE** のシステムに現在または後で接続されるすべてのデバイスは、プラグインインスタンスにより処理されます。**devices=*** オプションを使用する場合と同じです。

例3.4 ブロックデバイスとプラグインのマッチング

次の例では、**sda**、**sdb** など **sd** で始まるすべてのブロックデバイスに一致し、それらに対する境界は無効にしない例になります。

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

次の例は、**sda1** および **sda2** を除くすべてのブロックデバイスと一致します。

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

プラグインのインスタンスを指定しないと、そのプラグインは有効になりません。

このプラグインがより多くのオプションに対応していると、プラグインセクションでも指定できます。このオプションが指定されておらず、含まれているプラグインでこれまで指定しなかった場合は、デフォルト値が使用されます。

短いプラグイン構文

プラグインインスタンスにカスタム名を付ける必要がなく、設定ファイルにインスタンスの定義が 1 つしかない場合、**TuneD** は以下の簡単な構文に対応します。

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

この場合は、**type** の行を省略することができます。タイプと同様に、インスタンスは名前で参照されます。上記の例は、以下のように書き換えることができます。

例3.5 短い構文を使用したブロックデバイスのマッチング

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

プロファイルで競合するプラグインの定義

include オプションを使用して同じセクションを複数回指定した場合は、設定がマージされます。設定をマージできない場合は、競合がある以前の設定よりも、競合がある最後の定義が優先されます。以前に定義されたものが分からない場合は、**replace** ブール式オプションを使用して、それを **true** に設定します。これにより、同じ名前の以前の定義がすべて上書きされ、マージは行われません。

また、**enabled=false** オプションを指定してプラグインを無効にすることもできます。これは、インスタンスが定義されない場合と同じ効果になります。**include** オプションから以前の定義を再定義し、カスタムプロファイルでプラグインをアクティブにしない場合には、プラグインを無効にすると便利です。

注記

TuneD には、チューニングプロファイルの有効化または無効化の一環として、シェルコマンドを実行する機能が含まれます。これにより、TuneD に統合されていない機能で、**TuneD** プロファイルを拡張できます。
任意のシェルコマンドは、**script** プラグインを使用して指定できます。

関連情報

- **tuned.conf(5)** の man ページ

3.8. 利用可能な TUNED プラグイン

プラグインの監視

現在、以下の監視プラグインが実装されています。

disk

デバイスおよび測定間隔ごとのディスク負荷 (IO 操作の数) を取得します。

net

ネットワークカードおよび測定間隔ごとのネットワーク負荷 (転送済みパケットの数) を取得します。

load

CPU および測定間隔ごとの CPU 負荷を取得します。

プラグインのチューニング

現在、以下のチューニングプラグインが実装されています。動的チューニングを実装するのは、これらのプラグインの一部のみです。プラグインで対応しているオプションもリスト表示されます。

cpu

CPU ガバナーを、**governor** オプションで指定された値に設定し、CPU 負荷に応じて、電源管理サービス品質 (PM QoS) CPU ダイレクトメモリーアクセス (DMA) のレイテンシーを動的に変更します。
CPU 負荷が **load_threshold** オプションで指定された値よりも小さい場合、レイテンシーは **latency_high** オプションで指定した値に設定されます。それ以外では、**latency_low** で指定した値に設定されます。

レイテンシーを特定の値に強制し、さらに動的に変更しないようにすることもできます。これを行うには、**force_latency** オプションを、必要なレイテンシーの値に設定します。

eeepc_she

CPU の負荷に応じて、フロントサイドバス (FSB) の速度を動的に設定します。
この機能は一部のネットブックで利用でき、ASUS Super buf Engine (SHE) としても知られています。

CPU 負荷が **load_threshold_powersave** オプションで指定した値と同じかそれ未満の場合、プラグインは、FSB 速度を、**she_powersave** オプションで指定した値に設定します。CPU 負荷が **load_threshold_normal** オプションで指定した値と同じかそれより上になる場合は、FSB 速度が、**she_normal** オプションで指定された値に設定されます。

この機能のハードウェアサポートを **TuneD** が検出しない場合、静的チューニングには対応せず、プラグインも透過的に無効になります。

net

Wake on LAN 機能を、**wake_on_lan** オプションで指定した値に設定します。**ethtool** ユーティリティーと同じ構文を使用します。また、インターフェイスの使用状況に応じてインターフェイス速度が動的に変更します。

sysctl

プラグインオプションで指定したさまざまな **sysctl** 設定を設定します。
この構文は、**name=value** です。name は、**sysctl** ユーティリティーが指定した名前と同じです。

TuneDで利用可能な別のプラグインで対応していない設定を変更する必要がある場合は、**sysctl** プラグインを使用します。他の特定プラグインが、この設定に対応している場合は、そのプラグインを使用することが推奨されます。

usb

USB デバイスの autosuspend タイムアウトを、**autosuspend** パラメーターで指定した値に設定します。
値が **0** の場合は、autosuspend が無効になります。

vm

transparent_hugepages オプションの値に合わせて、Transparent Huge Page を有効または無効にします。
transparent_hugepages オプションの有効な値は次のとおりです。

- "always"
- "never"

- "madvise"

audio

音声コーデックの autosuspend タイムアウトを、**timeout** オプションで指定した値に設定します。現在、**snd_hda_intel** コーデックおよび **snd_ac97_codec** コーデックに対応しています。値が **0** の場合は、autosuspend が無効になります。また、ブール値オプション **reset_controller** を **true** に設定することにより、コントローラーを強制的にリセットすることもできます。

disk

elevator オプションで指定された値にディスクエレベーターを設定します。
また、以下も設定します。

- **apm** オプションで指定された値への APM
- **scheduler_quantum** オプションで指定された値へのスケジューラーの量子
- **spindown** オプションで指定された値へのディスクスピンダウンタイムアウト
- **readahead** パラメーターで指定した値までディスク先読み
- 現在のディスクが、**readahead_multiply** オプションで指定した定数を掛けた値に先読みされます。

さらに、このプラグインにより、現在のドライブ使用状況に応じて、ドライブの高度な電力管理設定および spindown タイムアウト設定が動的に変更します。動的チューニングは、ブール値オプション **dynamic** により制御でき、デフォルトで有効になります。

scsi_host

SCSI ホストのオプションをチューニングします。
Aggressive Link Power Management (ALPM) を、**alpm** オプションで指定した値に設定します。

mounts

disable_barriers オプションのブール値に応じて、マウントのバリアを有効または無効にします。

script

プロファイルの読み込み時またはアンロード時に、外部スクリプトまたはバイナリーを実行します。任意の実行可能ファイルを選択できます。

重要

script プラグインは、以前のリリースとの互換性を維持するために提供されています。必要な機能をカバーする場合は、他の**TuneD** プラグインを使用することが推奨されます。

TuneD は、以下のいずれかの引数で実行ファイルを呼び出します。

- プロファイルの読み込み時に **start**
- プロファイルのアンロード時に **stop**

実行可能ファイルに **stop** アクションを適切に実装し、**start** アクション中に変更したすべての設定を元に戻す必要があります。この手順を行わないと、**TuneD** プロファイルを変更した後のロールバック手順が機能しません。

bash スクリプトは、Bash ライブラリー **/usr/lib/tuned/functions** をインポートし、そこで定義されている関数を使用できます。これらの関数は、**TuneD** がネイティブに提供していない機能にのみ使用してください。関数名が **_wifi_set_power_level** などのアンダースコアで始まる場合は、将来変更される可能性があるため、関数をプライベートにし、スクリプトでは使用しないでください。

プラグイン構造の **script** パラメーターを使用して、実行ファイルへのパスを指定します。

例3.6 プロファイルからの Bash スクリプトの実行

プロファイルディレクトリーに置かれた **script.sh** という名前の Bash スクリプトを実行するには、次のコマンドを実行します。

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

sysfs

プラグインオプションで指定したさまざまな **sysfs** 設定を設定します。
構文は **name=value** となります。name は、使用する **sysfs** パスです。

このプラグインは、他のプラグインで対応していない一部の設定を変更する必要がある場合に使用します。特定のプラグインが必要な設定に対応する場合は、そのプラグインを優先します。

video

ビデオカードのさまざまな省電力レベルを設定します。現在、Radeon カードにのみ対応しています。
省電力レベルは、**radeon_powersave** オプションを使用して指定できます。対応している値は次のとおりです。

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

詳細は www.x.org を参照してください。このプラグインは実験的なものであるため、今後のリリースでオプションが変更する可能性があることに注意してください。

bootloader

カーネルコマンドラインにオプションを追加します。このプラグインは、GRUB 2 ブートローダーのみに対応しています。

grub2_cfg_file オプションを使用すると、GRUB 2 設定ファイルの場所を、標準以外のカスタマイズされた場所に指定できます。

そのカーネルオプションは、現在の GRUB 設定とそのテンプレートに追加されます。カーネルオプションを有効にするには、システムを再起動する必要があります。

別のプロファイルに切り替えるか、**TuneD** サービスを手動で停止すると、追加のオプションが削除されます。システムをシャットダウンまたは再起動しても、カーネルオプションは **grub.cfg** ファイルに残ります。

カーネルオプションは、以下の構文で指定できます。

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

例3.7 カーネルコマンドラインの変更

たとえば、**quiet** カーネルオプションを TuneD プロファイルに追加するには、**tuned.conf** ファイルに次の行を含めます。

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

以下に、**isolcpus=2** オプションをカーネルコマンドラインに追加するカスタムプロファイルの例を示します。

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

service

プラグインオプションで指定されたさまざまな **sysvinit**、**sysv-rc**、**openrc**、および **systemd** サービスを処理します。
構文は **service.service_name=command[,file:file]** です。

サポートされているサービス処理コマンドは次のとおりです。

- **start**
- **stop**
- **enable**
- **disable**

コンマ (**,**) またはセミコロン (**;**) を使用して、複数のコマンドを区切ります。ディレクティブの競合の場合、**service** プラグインは最後にリストされたものを使用します。

オプションの **file:file** ディレクティブを使用して、**systemd** 専用のオーバーレイ設定ファイル **file** をインストールします。他の init システムは、このディレクティブを無視します。**service** プラグインは、オーバーレイ設定ファイルを **/etc/systemd/system/service_name.service.d/** ディレクトリーにコピーします。プロファイルがアンロードされると、**service** プラグインは、これらのディレクトリーが空の場合は削除します。

注記

service プラグインは、**systemd** init システム以外の現在のランレベルでのみ動作します。

例3.8 オーバーレイファイルを使用した sendmail サービスの開始および有効化

```
[service]
service.sendmail=start,enable,file:${i:PROFILE_DIR}/tuned-sendmail.conf
```

内部変数 **${i:PROFILE_DIR}** は、プラグインがプロファイルをロードするディレクトリーを指します。

scheduler

スケジューリングの優先度、CPU コア分離、プロセスアフィニティー、スレッドアフィニティー、および IRQ アフィニティーを調整するためのさまざまなオプションを提供します。

利用可能なさまざまなオプションの詳細は、Functionalities of the **scheduler** TuneD plug-in を参照してください。

3.9. SCHEDULER TUNED プラグインの機能

scheduler TuneD プラグインを使用して、スケジューリングの優先度、CPU コアの分離、プロセスアフィニティー、スレッドアフィニティー、および IRQ アフィニティーを制御および調整します。

CPU の分離

プロセス、スレッド、および IRQ が特定の CPU を使用しないようにするには、**isolated_cores** オプションを使用します。これは、プロセスおよびスレッドアフィニティー、IRQ アフィニティーを変更し、IRQ の **default_smp_affinity** パラメーターを設定します。

CPU アフィニティーマスクは、**sched_setaffinity()** システムコールの成功を条件として、**ps_whitelist** オプションに一致するすべてのプロセスとスレッドに対して調整されます。**ps_whitelist** 正規表現のデフォルト設定は、すべてのプロセスおよびスレッド名に一致する **.*** です。特定のプロセスおよびスレッドを除外するには、**ps_blacklist** オプションを使用します。このオプションの値も正規表現として解釈されます。プロセス名とスレッド名は、その正規表現と照合されます。プロファイルロールバックにより、一致するすべてのプロセスとスレッドがすべての CPU で実行され、プロファイルアプリケーションの前に IRQ 設定が復元されます。

ps_whitelist オプションおよび **ps_blacklist** オプションで、**;** で区切った複数の正規表現がサポートされます。エスケープされたセミコロン **\;** はそのまま使用されます。

例3.9 CPUs 2-4 の分離

以下の設定は CPU 2-4 を分離します。**ps_blacklist** 正規表現に一致するプロセスおよびスレッドは、分離に関係なく任意の CPU を使用できます。

```
[scheduler]
isolated_cores=2-4
ps_blacklist=.*pmd.*;.*PMD.*;^DPDK;.*qemu-kvm.*
```

IRQ SMP アフィニティー

/proc/irq/default_smp_affinity ファイルには、すべての非アクティブな割り込み要求 (IRQ) ソース用のシステム上のデフォルトのターゲット CPU コアを表すビットマスクが含まれます。IRQ がアクティブまたは割り当てられると、**/proc/irq/default_smp_affinity** ファイルの値は IRQ のアフィニティービットマスクを決定します。

default_irq_smp_affinity パラメーターは、TuneD が **/proc/irq/default_smp_affinity** ファイルに書き込むものを制御します。**default_irq_smp_affinity** パラメーターは、以下の値と動作をサポートします。

calc

isolated_cores パラメーターから **/proc/irq/default_smp_affinity** ファイルの内容を計算します。**isolated_cores** パラメーターの反転は、分離していないコアを計算します。
次に、分離されていないコアの交差部分と、**/proc/irq/default_smp_affinity** ファイルの以前の内容が **/proc/irq/default_smp_affinity** ファイルに書き込まれます。

これは、**default_irq_smp_affinity** パラメーターが省略された場合のデフォルトの動作です。

ignore

TuneD は、**/proc/irq/default_smp_affinity** ファイルを変更しません。

CPU リスト

1 などの単一の数値、**1,3** などのコンマ区切りのリスト、または **3-5** などの範囲の形式を取ります。CPU リストを展開し、これを **/proc/irq/default_smp_affinity** ファイルに直接書き込みます。

例3.10 明示的な CPU リストを使用したデフォルトの IRQ smp アフィニティーの設定

以下の例では、明示的な CPU リストを使用して、デフォルトの IRQ SMP アフィニティーを CPU 0 および 2 に設定します。

```
[scheduler]
isolated_cores=1,3
default_irq_smp_affinity=0,2
```

スケジューリングポリシー

プロセスまたはスレッドのグループのスケジューリングポリシー、優先度、およびアフィニティーを調整するには、以下の構文を使用します。

```
group.groupname=rule_prio:sched:prio:affinity:regex
```

ここで **rule_prio** は、ルールの内部 TuneD 優先度を定義します。ルールは優先度に基づいてソートされます。これは、継承が以前に定義されたルールを並べ替えることができるようにするために必要です。同等の **rule_prio** ルールは、定義された順序で処理される必要があります。ただし、これは Python インタープリターに依存します。**groupname** の継承されたルールを無効にするには、以下を使用します。

```
group.groupname=
```

sched は以下のいずれかである必要があります。

f

先入れ先出し (FIFO)

b

バッチ

r

ラウンドロビン

o

その他

*

変更対象外

affinity は 16 進数での CPU アフィニティーです。変更しない場合は * を使用します。

prio はスケジューリングの優先度です (**chrt -m** を参照)。

regex は Python の正規表現です。これは、**ps -eo cmd** コマンドの出力と照合されます。

指定したプロセス名は、複数のグループに一致させることができます。このような場合、最後に一致する **regex** により、優先順位とスケジューリングポリシーが決まります。

例3.11 スケジューリングポリシーおよび優先度の設定

以下の例では、スケジューリングポリシーと優先度をカーネルスレッドおよびウォッチドッグに設定します。

```
[scheduler]
group.kthreads=0:*:1:*:\[.*\]$
group.watchdog=0:f:99:*:\[watchdog.*\]
```

scheduler プラグインは、**perf** イベントループを使用して、新しく作成されたプロセスを識別します。デフォルトでは、**perf.RECORD_COMM** および **perf.RECORD_EXIT** のイベントをリッスンします。

perf_process_fork パラメーターを **true** に設定すると、プラグインに対して **perf.RECORD_FORK** イベントもリッスンするように指示します。つまり、**fork()** システムコールによって作成された子プロセスが処理されます。

注記

perf イベントの処理には大量の CPU オーバーヘッドが発生する可能性があります。

スケジューラープラグインの CPU オーバーヘッドは、スケジューラー **runtime** オプションを使用して **0** に設定することで軽減できます。これにより、動的スケジューラー機能が完全に無効になり、perf イベントは監視されず、処理されません。これによるデメリットは、プロセスとスレッドの調整がプロファイルアプリケーションでのみ実行されることです。

例3.12 動的スケジューラー機能の無効化

以下の例では、CPU 1 と 3 を分離しながら、動的スケジューラー機能を無効にします。

```
[scheduler]
runtime=0
isolated_cores=1,3
```

mmapped バッファーは **perf** イベントに使用されます。負荷が大きい場合、このバッファーがオーバーフローする可能性があり、プラグインが欠落しているイベントを開始し、新しく作成されたプロセスを処理しない可能性があります。このような場合は、**perf_mmap_pages** パラメーターを使用してバッファーサイズを増やします。**perf_mmap_pages** パラメーターの値は 2 の累乗である必要があります。**perf_mmap_pages** パラメーターが手動で設定されていない場合は、デフォルト値の 128 が使用されます。

cgroupsを使用した制限

scheduler プラグインは、**cgroups** v1 を使用したプロセスおよびスレッド制限をサポートします。

cgroup_mount_point オプションは、cgroup ファイルシステムをマウントするパス、または、**TuneD** のマウントが想定される場所を指定します。設定されていない場合、**/sys/fs/cgroup/cpuset** が想定されます。

cgroup_groups_init オプションが **1** に設定されている場合、**TuneD** は、**cgroup*** オプションで定義されたすべての **cgroups** を作成および削除します。これがデフォルトの動作です。**cgroup_mount_point** オプションが **0** に設定されている場合、**cgroups** は他の方法で事前設定する必要があります。

cgroup_mount_point_init オプションが **1** に設定されている場合、**TuneD** は cgroup マウントポイントを作成し、削除します。これは **cgroup_groups_init = 1** を意味します。**cgroup_mount_point_init** オプションが **0** に設定されている場合は、他の方法で **cgroups** マウントポイントを事前設定する必要があります。これがデフォルトの動作です。

cgroup_for_isolated_cores オプションは、**isolated_cores** オプション機能の **cgroup** 名です。たとえば、システムに 4 つの CPU がある場合、**isolated_cores=1** は、**Tuned** がすべてのプロセスとスレッドを CPU 0、2、および 3 に移動することを意味します。**scheduler** プラグインは、計算された CPU アフィニティーを指定された cgroup の **cpuset.cpus** コントロールファイルに書き込み、一致するすべてのプロセスおよびスレッドをこのグループに移動することで、指定されたコアを分離します。このオプションが設定されていない場合、**sched_setaffinity()** を使用する従来の cpuset アフィニティーが CPU アフィニティーを設定します。

cgroup.cgroup_name オプションは、任意の **cgroups** のアフィニティーを定義します。階層的な cgroups を使用することもできますが、階層を正しい順序で指定する必要があります。**TuneD** は、**cgroup_mount_point** オプションで指定された場所に **cgroup** を強制的に配置する点を除き、ここでは健全性チェックを行いません。

group. で始まるスケジューラーオプションの構文が拡張され、16 進数の **affinity** ではなく、**cgroup.cgroup_name** が使用されるようになりました。一致するプロセスは **cgroup cgroup_name** に移動されます。上記のように、**cgroup.** オプションで定義されていない cgroup を使用することもできます。たとえば、**TuneD** によって管理されない **cgroups** などがあります。

すべての **cgroup** 名は、ピリオド (**.**) をスラッシュ (/) に置き換えてサニタイズされます。これにより、プラグインが **cgroup_mount_point** オプションで指定された場所の外部に書き込むことを防ぎます。

例3.13 scheduler プラグインでの cgroups v1 の使用

以下の例では、2 つの **cgroups**、**group1**、および **group2** を作成します。cgroup **group1** アフィニティーを CPU 2 に設定し、**cgroup group2** を CPU 0 および 2 に設定します。4 つの CPU 設定を指定すると、**isolated_cores=1** オプションはすべてのプロセスとスレッドを CPU コア 0、2、および 3 に移動します。**ps_blacklist** 正規表現で指定されたプロセスおよびスレッドは移動されません。

```
[scheduler]
cgroup_mount_point=/sys/fs/cgroup/cpuset
cgroup_mount_point_init=1
cgroup_groups_init=1
cgroup_for_isolated_cores=group
cgroup.group1=2
cgroup.group2=0,2

group.ksoftirqd=0:f:2:cgroup.group1:ksoftirqd.*
ps_blacklist=ksoftirqd.*;rcuc.*;rcub.*;ktimersoftd.*
isolated_cores=1
```

cgroup_ps_blacklist オプションは、指定された **cgroups** に属するプロセスを除外します。このオプションで指定された正規表現は、**/proc/PID/cgroups** の **cgroup** 階層と照合されます。コンマ (**,**) は、正規表現の一致前に **cgroups** v1 階層を **/proc/PID/cgroups** から分離します。以下は、正規表現が照合される内容の例です。

```
10:hugetlb:/,9:perf_event:/,8:blkio:/
```

複数の正規表現はセミコロン (**;**) で区切ることができます。セミコロンは論理 or 演算子を表します。

例3.14 cgroups を使用したスケジューラーからのプロセスの除外

以下の例では、**scheduler** プラグインは、cgroup **/daemons** に属するプロセスを除いて、すべてのプロセスをコア 1 から移動します。**\b** 文字列は、単語境界に一致する正規表現のメタ文字です。

```
[scheduler]
isolated_cores=1
cgroup_ps_blacklist=:/daemons\b
```

以下の例では、**scheduler** プラグインは、階層 ID が 8 で、controller-list **blkio** を持つ cgroup に属するすべてのプロセスを除外します。

```
[scheduler]
isolated_cores=1
cgroup_ps_blacklist=\b8:blkio:
```

最近のカーネルは、一部の **sched_** および **numa_balancing_** カーネルランタイムパラメーターを **sysctl** ユーティリティーが管理する **/proc/sys/kernel** ディレクトリーから、通常は **/sys/kernel/debug** ディレクトリーにマウントされる **debugfs** に移動しました。**TuneD** は、**scheduler** プラグインを介して以下のパラメーターの抽象化メカニズムを提供します。このメカニズムでは、**TuneD** は、使用されるカーネルに基づいて、指定された値を正しい場所に書き込みます。

- **sched_min_granularity_ns**
- **sched_latency_ns**
- **sched_wakeup_granularity_ns**
- **sched_tunable_scaling**
- **sched_migration_cost_ns**

- **sched_nr_migrate**
- **numa_balancing_scan_delay_ms**
- **numa_balancing_scan_period_min_ms**
- **numa_balancing_scan_period_max_ms**
- **numa_balancing_scan_size_mb**

例3.15 移行を決定するためにタスクの "cache hot" 値を設定します。

古いカーネルで以下のパラメーターを設定すると、**sysctl** は **500000** の値を **/proc/sys/kernel/sched_migration_cost_ns** ファイルに書き込むことを意味します。

```
[sysctl]
kernel.sched_migration_cost_ns=500000
```

これは、最近のカーネルでは、**scheduler** プラグインを介して次のパラメーターを設定するのと同じです。

```
[scheduler]
sched_migration_cost_ns=500000
```

つまり、TuneD は **500000** の値を **/sys/kernel/debug/sched/migration_cost_ns** ファイルに書き込みます。

3.10. TUNED プロファイルの変数

TuneD プロファイルがアクティブになると、変数は実行時にデプロイメントします。

TuneD変数を使用すると、TuneDプロファイルで必要な入力を減らすことができます。

TuneDプロファイルには事前定義された変数はありません。プロファイルに **[variables]** セクションを作成し、以下の構文を使用すると、独自の変数を定義できます。

```
[variables]

variable_name=value
```

プロファイル内の変数の値をデプロイメントするには、以下の構文を使用します。

```
${variable_name}
```

例3.16 変数を使用した CPU コアの分離

以下の例では、**${isolated_cores}** 変数が **1,2** にデプロイメントされるため、カーネルは **isolcpus=1,2** オプションで起動します。

```
[variables]
isolated_cores=1,2
```

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

変数は個別のファイルで指定できます。たとえば、次の行を **tuned.conf** に追加できます。

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

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

isolated_cores=1,2 オプションを **/etc/tuned/my-variables.conf** ファイルに追加すると、カーネルが **isolcpus=1,2** オプションで起動します。

関連情報

- **tuned.conf(5)** の man ページ

3.11. TUNED プロファイルの組み込み関数

組み込み関数は、**TuneD** プロファイルがアクティブになると、実行時に拡張します。

これにより、以下が可能になります。

- さまざまな組み込み関数と、**TuneD**変数の使用
- Python でカスタム関数を作成し、プラグインの形式で**TuneD** に追加します。

関数を呼び出すには、以下の構文を使用します。

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

プロファイルと **tuned.conf** ファイルが置かれたディレクトリーパスをデプロイメントするには、特殊な構文が必要な **PROFILE_DIR** 関数を使用します、

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

例3.17 変数と組み込み関数を使用した CPU コア分離

次の例では、**${non_isolated_cores}** 変数は **0,3-5** にデプロイメントされ、**cpulist_invert** 組み込み関数が **0,3-5** 引数で呼び出されます。

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

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

cpulist_invert 関数は、CPU のリストを反転します。6 CPU のマシンでは、反転が **1,2** になり、カーネルは **isolcpus=1,2** コマンドラインオプションで起動します。

関連情報

- **tuned.conf(5)** の man ページ

3.12. TUNED プロファイルで利用可能な組み込み関数

すべての TuneD プロファイルで、以下の組み込み関数を使用できます。

PROFILE_DIR

プロファイルと **tuned.conf** ファイルが置かれているディレクトリーパスを返します。

exec

プロセスを実行し、その出力を返します。

assertion

2 つの引数を比較します。**一致しない** 場合、関数は最初の引数からテキストをログに記録し、プロファイルの読み込みを中止します。

assertion_non_equal

2 つの引数を比較します。2 つの引数が **一致する** 場合、関数は最初の引数からテキストをログに記録し、プロファイルの読み込みを中止します。

kb2s

キロバイトをディスクセクターに変換します。

s2kb

ディスクセクターをキロバイトに変換します。

strip

渡されたすべての引数から文字列を作成し、最初と最後の空白の両方を削除します。

virt_check

TuneD が仮想マシン (VM) またはベアメタルのどちらで実行しているかを確認します。

- 仮想マシン内では、この関数が最初の引数を返します。
- ベアメタルでは、この関数は、エラーが発生した場合でも 2 番目の引数を返します。

cpulist_invert

補完するために CPU のリストを反転します。たとえば、0 から 3 までの番号が付けられた 4 つの CPU を持つシステムでは、リスト **0,2,3** の反転は **1** です。

cpulist2hex

CPU リストを 16 進数の CPU マスクに変換します。

cpulist2hex_invert

CPU リストを 16 進数の CPU マスクに変換し、反転します。

hex2cpulist

16 進数の CPU マスクを CPU リストに変換します。

cpulist_online

リストからの CPU がオンラインかどうかをチェックします。オンライン CPU のみを含むリストを返します。

cpulist_present

リストに CPU が存在するかどうかを確認します。存在する CPU のみを含むリストを返します。

cpulist_unpack

1-3,4 形式の CPU リストを、**1,2,3,4** にデプロイメントします。

cpulist_pack

CPU リストを、**1,2,3,5** の形式で **1-3,5** に圧縮します。

3.13. 新しい TUNED プロファイルの作成

この手順では、カスタムパフォーマンスルールを使用して新しい**TuneD**プロファイルを作成します。

前提条件

- **TuneD** サービスが実行されています。詳細は、TuneD のインストールと有効化 を参照してください。

手順

1. **/etc/tuned/** ディレクトリーで、作成するプロファイルと同じ名前の新しいディレクトリー作成します。

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

2. 新しいディレクトリーに、ファイル **tuned.conf** を作成します。必要に応じて、**[main]** セクションとプラグイン定義を追加します。
たとえば、**balanced** プロファイルの設定を表示します。

```
[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. プロファイルをアクティベートするには、次のコマンドを実行します。

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

4. **TuneD** プロファイルが有効であり、システム設定が適用されていることを確認します。

```
$ 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.
```

関連情報

- **tuned.conf(5)** の man ページ

3.14. 既存の TUNED プロファイルの変更

この手順では、既存の**TuneD** プロファイルに基づいて変更した子プロファイルを作成します。

前提条件

- **TuneD** サービスが実行されています。詳細は、TuneD のインストールと有効化 を参照してください。

手順

1. **/etc/tuned/** ディレクトリーで、作成するプロファイルと同じ名前の新しいディレクトリー作成します。

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

2. 新しいディレクトリーに、ファイル **tuned.conf** を作成し、以下のように **[main]** セクションを設定します。

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

parent-profile を、変更しているプロファイルの名前に置き換えます。

3. プロファイルの変更を含めます。

例3.18 throughput-performance プロファイルでスワップを低減

throughput-perfromance プロファイルの設定を使用し、**vm.swappiness** の値を、デフォルトの 10 ではなく 5 に変更するには、以下を使用します。

```
[main]
include=throughput-performance

[sysctl]
vm.swappiness=5
```

4. プロファイルをアクティベートするには、次のコマンドを実行します。

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

5. **TuneD** プロファイルが有効であり、システム設定が適用されていることを確認します。

```
$ 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.
```

関連情報

- **tuned.conf(5)** の man ページ

3.15. TUNED を使用したディスクスケジューラーの設定

この手順では、選択したブロックデバイスに特定のディスクスケジューラーを設定するTuneD プロファイルを作成して有効にします。この設定は、システムを再起動しても持続します。

以下のコマンドと設定で、以下の内容を置き換えます。

- device をブロックデバイスの名前に置き換えます (例: **sdf**)。
- selected-scheduler を、デバイスに設定するディスクスケジューラーに置き換えます (例: **bfq**)。

前提条件

- **Tuned** サービスがインストールされ、有効になっている。詳細は、TuneD のインストールと有効化 を参照してください。

手順

1. 必要に応じて、プロファイルのベースとなる既存のTuneDプロファイルを選択します。利用可能なプロファイルのリストは、RHEL とともに配布される TuneD プロファイル を参照してください。
   現在アクティブなプロファイルを確認するには、次のコマンドを実行します。

   ```
   $ tuned-adm active
   ```

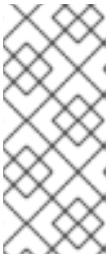
2. TuneD プロファイルを保持する新しいディレクトリーを作成します。

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

3. 選択したブロックデバイスのシステム固有の識別子を見つけます。

   ```
   $ 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
   ```



注記

この例のコマンドは、指定したブロックデバイスに関連付けられた World Wide Name (WWN) またはシリアル番号として識別されるすべての値を返します。WWN を使用することが推奨されますが、WWN は特定のデバイスで常に利用できる訳ではなく、コマンド例で返される値は、**デバイスのシステム固有の ID** として使用することが許容されます。

4. **/etc/tuned/my-profile/tuned.conf** 設定ファイルを作成します。このファイルで、以下のオプションを設定します。

   a. 必要に応じて、既存のプロファイルを追加します。

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

   b. WWN 識別子に一致するデバイスに対して選択したディスクスケジューラーを設定します。

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

   ここでは、以下のようになります。

   - **IDNAME** を、使用されている識別子名に置き換えます (例:**ID_WWN**)。
   - **device system unique id** を、選択した識別子の値に置き換えます (例:**0x5002538d00000000**)。
     devices_udev_regex オプションで複数のデバイスに一致させるには、識別子を括弧で囲み、垂直バーで区切ります。

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

5. プロファイルを有効にします。

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

検証手順

1. TuneD プロファイルがアクティブで、適用されていることを確認します。

   ```
   $ 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** ファイルの内容を読み取ります。

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

ファイル名の **device** を、**sdc** などのブロックデバイス名に置き換えます。

アクティブなスケジューラーは、角括弧 (**[]**) にリスト表示されます。

関連情報

- TuneD プロファイルのカスタマイズ

第4章 TUNA インターフェイスを使用したシステムの確認

tuna ツールを使用してスケジューラーの調整可能パラメーターの調整、スレッド優先度の調整、IRQ ハンドラー、CPU コアおよびソケットの分離を行います。tuna は、チューニングタスクを実行する際の複雑性を軽減します。

tuna ツールは、以下の操作を実行します。

- システム上の CPU の表示
- システム上で現在実行中の割り込み要求 (IRQ) の表示
- スレッドに関するポリシーおよび優先度の情報の変更
- システムの現在のポリシーと優先度 の表示

4.1. TUNA ツールのインストール

tuna ツールは、稼働中のシステムで使用されるように設計されています。これにより、アプリケーション固有の測定ツールで、変更の直後にシステムパフォーマンスを確認および分析できます。

手順

- **tuna** ツールをインストールします。

```
# yum install tuna
```

検証手順

- 利用可能な **tuna** CLI オプションを表示します。

```
# tuna -h
```

関連情報

- **tuna(8)** の man ページ

4.2. TUNA ツールを使用したシステムステータスの表示

この手順では、**tuna** コマンドラインインターフェイス (CLI) ツールを使用してシステムの状態を表示する方法を説明します。

前提条件

- tuna ツールがインストールされている。詳細は、tuna ツールのインストール を参照してください。

手順

- 現在のポリシーおよび優先度を表示するには、以下を実行します。

```
# 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 に対応する特定のスレッドまたはコマンド名と一致する場合は、次のコマンドを実行します。

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

  pid_or_cmd_list 引数は、コンマ区切りの PID またはコマンド名パターンのリストです。

- **tuna** CLI を使用して CPU をチューニングするには、tuna ツールを使用した CPU のチューニング を参照してください。

- **tuna** ツールを使用して IRQ をチューニングするには、tuna ツールを使用した IRQ のチューニング を参照してください。

- 変更した設定を保存するには、以下を実行します。

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

  このコマンドは、現在実行中のカーネルスレッドのみを保存します。実行していないプロセスは保存されません。

関連情報

- **tuna(8)** の man ページ

4.3. TUNA ツールを使用した CPU の調整

tuna ツールコマンドは、個別の CPU をターゲットとして指定できます。

tuna ツールを使用すると、以下が可能になります。

CPU の分離
: 指定した CPU で実行しているすべてのタスクが、次に利用可能な CPU に移動します。CPU の分離は、すべてのスレッドのアフィニティーマスクから削除することで利用できなくなります。

CPU の追加
: 指定された CPU でタスクを実行できるようにします。

CPU の復元
: 指定した CPU を以前の設定に戻します。

この手順では、**tuna** CLI を使用して CPU を調整する方法を説明します。

前提条件

- tuna ツールがインストールされている。詳細は、tuna ツールのインストール を参照してください。

手順

- コマンドの影響を受ける CPU のリストを指定するには、次のコマンドを実行します。

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

cpu_list 引数は、コンマ区切りの CPU 番号のリストです。例: **--cpus=0,2**.CPU リストは、**--cpus="1-3"** の範囲でも指定でき、CPU 1、2、および 3 を選択します。

現在の **cpu_list** に特定の CPU を追加するには、たとえば **--cpus=+0** を使用します。

[command] を、**--isolate** に置き換えます。

- CPU を分離するには、以下を実行します。

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

- CPU を指定するには、以下を実行します。

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

- 4 つ以上のプロセッサーを持つシステムを使用するには、すべての ssh スレッドを CPU **0** および **1** で実行し、CPU **2** および **3** のすべての **http** スレッドを実行する方法を表示します。

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

このコマンドは、以下の操作を順次実行します。

1. CPU **0** および **1** を選択します。
2. **ssh** で開始するスレッドをすべて選択します。
3. 選択したスレッドを選択した CPU に移動します。tuna は、**ssh** で始まるスレッドのアフィニティーマスクを適切な CPU に設定します。CPU は、数字で **0** および **1** で表すことができ、16 進マスクでは 0x3 で、またはバイナリーでは 11 として表現できます。
4. CPU リストを **2** および **3** にリセットします。
5. **http** で始まるすべてのスレッドを選択します。
6. 選択したスレッドを指定された CPU に移動します。tuna は、**http** で始まるスレッドのアフィニティーマスクを指定された CPU に設定します。CPU は、16 進マスクで 0xC または 1100 のバイナリーで **2** および **3** で表すこともできます。

検証手順

- 現在の設定を表示し、変更が想定どおりに実行されたことを確認します。

```
# 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
```

このコマンドは、以下の操作を順次実行します。

1. **gnome-sc** スレッドで始まるすべてのスレッドを選択します。
2. 選択したスレッドを表示して、ユーザーがアフィニティーマスクと RT の優先度を検証できるようにします。
3. CPU 0 を選択します。
4. **gnome-sc** スレッドを指定の CPU 0 に移動します。
5. 移動の結果を表示します。
6. CPU リストを CPU 1 にリセットします。
7. **gnome-sc** スレッドを指定した CPU (CPU 1) に移動します。
8. 移動の結果を表示します。
9. CPU リストに CPU 0 を追加します。
10. **gnome-sc** スレッドを、指定した CPU、CPU 0、および 1 に移動します。
11. 移動の結果を表示します。

関連情報

- **/proc/cpuinfo** ファイル
- **tuna(8)** の man ページ

4.4. TUNA ツールを使用した IRQ のチューニング

/proc/interrupts ファイルには、IRQ ごとの割り込みの数、割り込みのタイプ、およびその IRQ にあるデバイスの名前が記録されます。

この手順では、**tuna** ツールを使用して IRQ を調整する方法を説明します。

前提条件

- tuna ツールがインストールされている。詳細は、tuna ツールのインストール を参照してください。

手順

- 現在の IRQ とそれらのアフィニティーを表示するには、以下を実行します。

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

- コマンドの影響を受ける IRQ のリストを指定するには、次のコマンドを実行します。

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

irq_list 引数は、コンマ区切りの IRQ 番号またはユーザー名パターンのリストです。

[**コマンド**] を、たとえば **--spred** に置き換えます。

- 指定した CPU に割り込みを移動するには、以下を実行します。

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

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

128 を irq_list 引数に置き換え、 3 を cpu_list 引数に置き換えます。

cpu_list 引数は、**--cpus=0,2** などのコンマ区切り CPU 番号のリストです。詳細は、 tuna ツールを使用した CPU の調整 を参照してください。

検証手順

- 選択した IRQ の状態を、割り込みを指定の CPU に移動してから比較します。

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

関連情報

- **/procs/interrupts** ファイル
- **tuna(8)** の man ページ

第5章 RHEL システムロールを使用したパフォーマンスの監視

システム管理者は、Ansible Automation Platform コントロールノードで **metrics** RHEL システムロールを使用して、システムのパフォーマンスを監視できます。

5.1. RHEL システムロールを使用するためのコントロールノードと管理対象ノードの準備

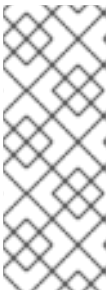
個々の RHEL システムロールを使用してサービスと設定を管理するには、その前に、コントロールノードと管理対象ノードを準備する必要があります。

5.1.1. RHEL 8 でのコントロールノードの準備

RHEL システムロールを使用する前に、コントロールノードを設定する必要があります。次に、このシステムは、Playbook に従ってインベントリーから管理対象ホストを設定します。

前提条件

- RHEL 8.6 以降がインストールされている。RHEL のインストールの詳細は、標準 RHEL 8 インストールの実行 を参照してください。



注記

RHEL 8.5 以前のバージョンでは、Ansible パッケージは Ansible Core ではなく Ansible Engine を通じて提供され、さまざまなサポートレベルが提供されていました。パッケージは RHEL 8.6 以降の Ansible Automation コンテンツと互換性がない可能性があるため、Ansible Engine は使用しないでください。詳細は、Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories を参照してください。

- システムはカスタマーポータルに登録されます。
- **Red Hat Enterprise Linux Server** サブスクリプションがシステムにアタッチされている。
- オプション: **Ansible Automation Platform** サブスクリプションがシステムにアタッチされている。

手順

1. Playbook を管理および実行するための **ansible** という名前のユーザーを作成します。

```
[root@control-node]# useradd ansible
```

2. 新しく作成した **ansible** ユーザーに切り替えます。

```
[root@control-node]# su - ansible
```

このユーザーとして残りの手順を実行します。

3. SSH の公開鍵と秘密鍵を作成します。

```
[ansible@control-node]$ ssh-keygen
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/home/ansible/.ssh/id_rsa):
Enter passphrase (empty for no passphrase): <password>
Enter same passphrase again: <password>
...
```

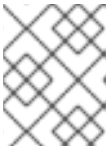
キーファイルの推奨されるデフォルトの場所を使用します。

4. オプション: 接続を確立するたびに Ansible が SSH キーのパスワードを要求しないように、SSH エージェントを設定します。

5. ~/**.ansible.cfg** ファイルを次の内容で作成します。

```
[defaults]
inventory = /home/ansible/inventory
remote_user = ansible

[privilege_escalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = True
```



注記

~/**.ansible.cfg** ファイルの設定は優先度が高く、グローバルな /**etc**/**ansible**/**ansible.cfg** ファイルの設定をオーバーライドします。

これらの設定を使用して、Ansible は次のアクションを実行します。

- 指定されたインベントリーファイルでホストを管理します。
- 管理対象ノードへの SSH 接続を確立するときに、**remote_user** パラメーターで設定されたアカウントを使用します。
- **sudo** ユーティリティーを使用して、**root** ユーザーとして管理対象ノードでタスクを実行します。
- Playbook を適用するたびに、リモートユーザーの root パスワードの入力を求められます。これは、セキュリティー上の理由から推奨されます。

6. 管理対象ホストのホスト名をリストする ~/**inventory** ファイルを INI または YAML 形式で作成します。インベントリーファイルでホストのグループを定義することもできます。たとえば、以下は、3 つのホストと **US** という名前の 1 つのホストグループを含む INI 形式のインベントリーファイルです。

```
managed-node-01.example.com

[US]
managed-node-02.example.com ansible_host=192.0.2.100
managed-node-03.example.com
```

コントロールノードはホスト名を解決できる必要があることに注意してください。DNS サーバーが特定のホスト名を解決できない場合は、ホストエントリーの横に **ansible_host** パラメーターを追加して、その IP アドレスを指定します。

7. RHEL システムロールをインストールします。

   - Ansible Automation Platform のない RHEL ホストに、**rhel-system-roles** パッケージをインストールします。

     ```
     [root@control-node]# yum install rhel-system-roles
     ```

     このコマンドは、**/usr/share/ansible/collections/ansible_collections/redhat/rhel_system_roles/** ディレクトリーにコレクションをインストールし、依存関係として **ansible-core** パッケージをインストールします。

   - Ansible Automation Platform で、**ansible** ユーザーとして次の手順を実行します。

     i. **~/.ansible.cfg** ファイルで コンテンツのプライマリーソースとして Red Hat Automation Hub を定義します。

     ii. Red Hat Automation Hub から **redhat.rhel_system_roles** コレクションをインストールします。

     ```
     [ansible@control-node]$ ansible-galaxy collection install
     redhat.rhel_system_roles
     ```

     このコマンドは、コレクションを **~/.ansible/collections/ansible_collections/redhat/rhel_system_roles/** ディレクトリーにインストールします。

次のステップ

- 管理対象ノードを準備します。詳細は、管理対象ノードの準備 を参照してください。

関連情報

- RHEL 9 および RHEL 8.6 以降の AppStream リポジトリーに含まれる Ansible Core パッケージのサポート範囲
- subscription-manager を使用して Red Hat カスタマーポータルにシステムを登録してサブスクライブする
- **ssh-keygen(1)** man ページ
- ssh-agent を使用して SSH キーでリモートマシンに接続する手順
- Ansible 構成設定
- インベントリーの構築方法
- Updates to using Ansible in RHEL 8.6 and 9.0

5.1.2. 管理対象ノードの準備

管理対象ノードはインベントリーにリストされているシステムであり、Playbook に従ってコントロールノードによって設定されます。管理対象ホストに Ansible をインストールする必要はありません。

前提条件

- コントロールノードを準備している。詳細は、RHEL 8 でのコントロールノードの準備 を参照してください。
- コントロールノードから SSH アクセスできる。

重要

root ユーザーとしての直接 SSH アクセスはセキュリティーリスクを引き起こします。このリスクを軽減するには、管理対象ノードを準備するときに、このノード上にローカルユーザーを作成し、**sudo** ポリシーを設定します。続いて、コントロールノードの Ansible は、ローカルユーザーアカウントを使用して管理対象ノードにログインし、**root** などの別のユーザーとして Playbook を実行できます。

手順

1. **ansible** という名前のユーザーを作成します。

```
[root@managed-node-01]# useradd ansible
```

コントロールノードは後でこのユーザーを使用して、このホストへの SSH 接続を確立します。

2. **ansible** ユーザーのパスワードを設定します。

```
[root@managed-node-01]# passwd ansible
Changing password for user ansible.
New password: <password>
Retype new password: <password>
passwd: all authentication tokens updated successfully.
```

Ansible が **sudo** を使用して **root** ユーザーとしてタスクを実行する場合は、このパスワードを入力する必要があります。

3. **ansible** ユーザーの SSH 公開鍵を管理対象ノードにインストールします。
   a. **ansible** ユーザーとしてコントロールノードにログインし、SSH 公開鍵を管理対象ノードにコピーします。

```
[ansible@control-node]$ ssh-copy-id managed-node-01.example.com
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/home/ansible/.ssh/id_rsa.pub"
The authenticity of host 'managed-node-01.example.com (192.0.2.100)' can't be
established.
ECDSA key fingerprint is
SHA256:9bZ33GJNODK3zbNhybokN/6Mq7hu3vpBXDrCxe7NAvo.
```

   b. プロンプトが表示されたら、**yes** と入力して接続します。

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys
```

   c. プロンプトが表示されたら、パスワードを入力します。

```
ansible@managed-node-01.example.com's password: <password>

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'managed-node-01.example.com'"
and check to make sure that only the key(s) you wanted were added.
```

d. コントロールノードでコマンドをリモートで実行して、SSH 接続を確認します。

```
[ansible@control-node]$ ssh managed-node-01.example.com whoami
ansible
```

4. **ansible** ユーザーの **sudo** 設定を作成します。

a. **visudo** コマンドを使用して、**/etc/sudoers.d/ansible** ファイルを作成および編集します。

```
[root@managed-node-01]# visudo /etc/sudoers.d/ansible
```

通常のエディターではなく **visudo** を使用する利点は、このユーティリティーがファイルをインストールする前に解析エラーなどの基本的なチェックを提供することです。

b. **/etc/sudoers.d/ansible** ファイルで、要件に応じた **sudoers** ポリシーを設定します。次に例を示します。

- **ansible** ユーザーのパスワードを入力した後、このホスト上で任意のユーザーおよびグループとしてすべてのコマンドを実行する権限を **ansible** ユーザーに付与するには、以下を使用します。

```
ansible   ALL=(ALL) ALL
```

- **ansible** ユーザーのパスワードを入力せずに、このホスト上で任意のユーザーおよびグループとしてすべてのコマンドを実行する権限を **ansible** ユーザーに付与するには、以下を使用します。

```
ansible   ALL=(ALL) NOPASSWD: ALL
```

または、セキュリティー要件に合わせてより細かいポリシーを設定します。**sudoers** ポリシーの詳細は、**sudoers(5)** man ページを参照してください。

検証

1. すべての管理対象ノード上のコントロールノードからコマンドを実行できることを確認します。

```
[ansible@control-node]$ ansible all -m ping
BECOME password: <password>
managed-node-01.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
...
```

ハードコーディングされたすべてのホストグループには、インベントリーファイルにリストされているすべてのホストが動的に含まれます。

2. Ansible **command** モジュールを使用して管理対象ホスト上で **whoami** ユーティリティーを実行し、権限昇格が正しく機能することを確認します。

```
[ansible@control-node]$ ansible managed-node-01.example.com -m command -a
whoami
BECOME password: <password>
managed-node-01.example.com | CHANGED | rc=0 >>
root
```

コマンドが root を返した場合、管理対象ノード上で **sudo** が正しく設定されています。

関連情報

- RHEL 8 でのコントロールノードの準備
- **sudoers(5)** man ページ

5.2. METRICS RHEL システムロールの概要

RHEL システムロールは、複数の RHEL システムをリモートで管理するための一貫した設定インターフェイスを提供する Ansible ロールおよびモジュールのコレクションです。**metrics** システムロールは、ローカルシステムのパフォーマンス分析サービスを設定します。必要に応じて、ローカルシステムによって監視される一連のリモートシステムも分析の対象とします。**metrics** システムロールを使用すると、**pcp** のセットアップとデプロイが Playbook によって処理されるため、**pcp** を別途設定しなくても、**pcp** を使用してシステムのパフォーマンスを監視できます。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

5.3. METRICS RHEL システムロールを使用して視覚的にローカルシステムを監視する

この手順では、**metrics** RHEL システムロールを使用してローカルシステムを監視しながら、同時に **Grafana** によるデータの視覚化をプロビジョニングする方法について説明します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **localhost** がコントロールノードのインベントリーファイルで設定されている。

```
localhost ansible_connection=local
```

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Manage metrics
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_graph_service: yes
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

metrics_graph_service のブール値が **value="yes"** に設定されているため、**Grafana** は自動的にインストールされ、データソースとして追加された **pcp** でプロビジョニングされます。**metrics_manage_firewall** と **metrics_manage_selinux** は両方とも **true** に設定されているため、メトリクスロールは **firewall** および **selinux** システムロールを使用して、メトリクスロールが使用するポートを管理します。

2. Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

検証

- マシンで収集されるメトリクスを視覚化するには、Grafana Web UI へのアクセス で説明されているように **grafana** Web インターフェイスにアクセスします。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

5.4. METRICS RHEL システムロールを使用して自己監視するようにシステム群を設定する

この手順では、**metrics** システムロールを使用して、マシン群が自己監視するように設定する方法を説明します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。

- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure a fleet of machines to monitor themselves
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_retention_days: 0
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

   metrics_manage_firewall と **metrics_manage_selinux** は両方とも **true** に設定されているため、メトリクスロールは **firewall** ロールと **selinux** ロールを使用して、**metrics** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

5.5. METRICS RHEL システムロールを使用して、ローカルマシンからマシン群を集中的に監視する

この手順では、**メトリクス** システムロールを使用してローカルマシンを設定し、マシン群を一元的に監視するとともに、**Grafana** によるデータの視覚化をプロビジョニングし、**Redis** によりデータをクエリーする方法について説明します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- **localhost** がコントロールノードのインベントリーファイルで設定されている。

```
localhost ansible_connection=local
```

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
- name: Set up your local machine to centrally monitor a fleet of machines
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_graph_service: yes
    metrics_query_service: yes
    metrics_retention_days: 10
    metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
    metrics_manage_firewall: yes
    metrics_manage_selinux: yes
```

metrics_graph_service および **metrics_query_service** のブール値は **value="yes"** に設定されているため、**grafana** は、**redis** にインデックス化された **pcp** データの記録のあるデータソースとして追加された **pcp** で自動的にインストールおよびプロビジョニングされます。これにより、**pcp** クエリー言語をデータの複雑なクエリーに使用できます。**metrics_manage_firewall** と **metrics_manage_selinux** は両方とも **true** に設定されているため、**metrics** ロールは **firewall** ロールと **selinux** ロールを使用して、**metrics** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

検証

- マシンによって一元的に収集されるメトリクスのグラフィック表示とデータのクエリーを行うには、Grafana Web UI へのアクセス で説明されているように **grafana** Web インターフェイスにアクセスします。

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

5.6. METRICS RHEL システムロールを使用してシステムを監視しながら認証を設定する

PCP は、Simple Authentication Security Layer (SASL) フレームワークを介して **scram-sha-256** 認証メ

カニズムに対応します。**metrics** RHEL システムロールは、 **scram-sha-256** 認証メカニズムを使用して認証を設定する手順を自動化します。この手順では、**metrics** RHEL システムロールを使用して認証を設定する方法について説明します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 既存の Playbook ファイル (例: ~/**playbook.yml)** を編集し、認証関連の変数を追加します。

```
---
- name: Set up authentication by using the scram-sha-256 authentication mechanism
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_retention_days: 0
    metrics_manage_firewall: true
    metrics_manage_selinux: true
    metrics_username: <username>
    metrics_password: <password>
```

2. Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

検証

- **sasl** 設定を確認します。

```
# pminfo -f -h "pcp://managed-node-01.example.com?username=<username>"
disk.dev.read
Password: <password>
disk.dev.read
inst [0 or "sda"] value 19540
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル

- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー

5.7. METRICS RHEL システムロールを使用して SQL SERVER のメトリクス収集を設定して有効にする

この手順では、**metrics** RHEL システムロールを使用して、ローカルシステムでの **pcp** を使用した Microsoft SQL Server のメトリクス収集の設定と有効化を自動化する方法について説明します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- Red Hat Enterprise Linux 用の Microsoft SQL Server をインストールし、SQL サーバーへの信頼できる接続が確立されている。
- Red Hat Enterprise Linux 用の SQL Server の Microsoft ODBC ドライバーがインストールされている。
- **localhost** がコントロールノードのインベントリーファイルで設定されている。

```
localhost ansible_connection=local
```

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- name: Configure and enable metrics collection for Microsoft SQL Server
  hosts: localhost
  roles:
    - rhel-system-roles.metrics
  vars:
    metrics_from_mssql: true
    metrics_manage_firewall: true
    metrics_manage_selinux: true
```

metrics_manage_firewall と **metrics_manage_selinux** は両方とも **true** に設定されているため、**metrics** ロールは **firewall** ロールと **selinux** ロールを使用して、**metrics** ロールが使用するポートを管理します。

2. Playbook の構文を検証します。

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

```
$ ansible-playbook ~/playbook.yml
```

検証

- **pcp** コマンドを使用して、SQL Server PMDA エージェント (mssql) が読み込まれ、実行されていることを確認します。

```
# pcp
platform: Linux sqlserver.example.com 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/sqlserver.example.com/20200326.16.31
     pmie: primary engine: /var/log/pcp/pmie/sqlserver.example.com/pmie.log
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.metrics/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/metrics/** ディレクトリー
- RHEL 8.2 を使用した Microsoft SQL Server の Performance Co-Pilot に関する ブログ投稿

第6章 PCP の設定

Performance Co-Pilot (PCP) は、システムレベルのパフォーマンス測定を監視、視覚化、保存、および分析するためのツール、サービス、およびライブラリーのスイートです。

6.1. PCP の概要

Python、Perl、C++、および C のインターフェイスを使用したパフォーマンスメトリックを追加できます。分析ツールは、Python、C++、C のクライアント API を直接使用でき、豊富な Web アプリケーションは、JSON インターフェイスを使用して利用可能なすべてのパフォーマンスデータを調べることができます。

ライブ結果とアーカイブされたデータを比較して、データパターンを解析できます。

PCP の機能:

- 軽量の分散アーキテクチャー。複雑なシステムの集中分析に役に立ちます。
- これにより、リアルタイムデータの監視および管理が可能になります。
- これにより、履歴データのログおよび取得が可能になります。

PCP には以下のコンポーネントがあります。

- Performance Metric Collector Daemon (**pmcd**) は、インストールされている Performance Metric Domain Agents (**pmda**) からパフォーマンスデータを収集します。**PMDA** は、システムで個別にロードまたはアンロードでき、同じホストの **PMCD** によって制御されます。
- **pminfo** や **pmstat** などのさまざまなクライアントツールは、同じホストまたはネットワーク上でこのデータを取得、表示、アーカイブ、処理できます。
- **pcp** パッケージは、コマンドラインツールと、基本的な機能を提供します。
- **pcp-gui** パッケージは、グラフィカルアプリケーションを提供します。**yum install pcp-gui** コマンドを実行して、**pcp-gui** パッケージをインストールします。詳細は、Visually tracing PCP log archives with the PCP Charts application を参照してください。

関連情報

- **pcp(1)** の man ページ
- **/usr/share/doc/pcp-doc/** ディレクトリー
- PCP で配布されるシステムサービスおよびツール
- Red Hat カスタマーポータルの PCP (Performance Co-Pilot) に関するナレッジ、チュートリアル、およびホワイトペーパー
- Red Hat ナレッジベース記事 Side-by-side comparison of PCP tools with legacy tools
- PCP アップストリームのドキュメント

6.2. PCP のインストールおよび有効化

PCP の使用を開始するには、必要なパッケージをすべてインストールし、PCP 監視サービスを有効にします。

この手順では、**pcp** パッケージを使用して PCP をインストールする方法を説明します。PCP のインストールを自動化するには、**pcp-zeroconf** パッケージを使用してインストールします。pcp-zeroconf を使用して PCP をインストールする方法の詳細は、PCP の pcp-zeroconf での設定 を参照してください。

手順

1. **pcp** パッケージをインストールします。

```
# yum install pcp
```

2. ホストマシンで **pmcd** サービスを有効にして起動します。

```
# systemctl enable pmcd

# systemctl start pmcd
```

検証手順

- **pmcd** プロセスがホストで実行されているかどうかを確認します。

```
# 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
```

関連情報

- **pmcd(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール

6.3. 最小限の PCP 設定のデプロイメント

最小 PCP 設定は、Red Hat Enterprise Linux でパフォーマンス統計を収集します。この設定は、詳細な分析のためにデータを収集するために必要な、実稼働システムに最低限のパッケージを追加します。

作成された **tar.gz** ファイルおよび **pmlogger** の出力のアーカイブは、さまざまな PCP ツールを使用して解析し、その他のソースのパフォーマンス情報と比較できます。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。

手順

1. **pmlogger** 設定を更新します。

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

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

```
# systemctl start pmcd.service

# systemctl start pmlogger.service
```

3. 必要な操作を実行して、パフォーマンスデータを記録します。

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

```
# systemctl stop pmcd.service

# systemctl stop pmlogger.service
```

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

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

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

このファイルをデプロイメントし、PCP ツールを使用してデータを解析します。

関連情報

- **pmlogconf(1)**、**pmlogger(1)**、および **pmcd(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール

6.4. PCP で配布されるシステムサービスおよびツール

Performance Co-Pilot (PCP) には、パフォーマンスの測定に使用できるさまざまなシステムサービスとツールが含まれます。基本パッケージ **pcp** には、システムサービスと基本ツールが含まれます。追加のツールは、**pcp-system-tools**、**pcp-gui**、および **pcp-devel** パッケージで提供されます。

PCP で配布されるシステムサービスのロール

pmcd
: PMCD (Performance Metric Collector Daemon)

pmie
: Performance Metrics In difference Engine

pmlogger
: パフォーマンスメトリックロガー。

pmproxy

リアルタイムおよびヒストリカルなパフォーマンスメトリックのプロキシー、時系列クエリー、REST API サービス。

基本 PCP パッケージで配布されるツール

pcp

Performance Co-Pilot インストールの現在のステータスを表示します。

pcp-vmstat

システムパフォーマンスの概要を 5 秒ごとに表示します。プロセス、メモリー、ページング、ブロック IO、トラップ、CPU のアクティビティーに関する情報を表示します。

pmconfig

設定パラメーターの値を表示します。

pmdiff

パフォーマンスのリグレッションを検索する際に重要と思われる変更について、指定された時間枠で、1 つまたは 2 つのアーカイブのすべてのメトリックの平均値を比較します。

pmdumplog

Performance Co-Pilot アーカイブファイルの制御、メタデータ、インデックス、および状態に関する情報を表示します。

pmfind

ネットワークで PCP サービスを見つけます。

pmie

一連の演算式、論理式、およびルール式を定期的に評価する推論エンジン。メトリックは、ライブシステムまたは Performance Co-Pilot アーカイブファイルのいずれかから収集されます。

pmieconf

設定可能な **pmie** 変数を表示または設定します。

pmiectl

pmie のプライマリー以外のインスタンスを管理します。

pminfo

パフォーマンスメトリックに関する情報を表示します。メトリックは、ライブシステムまたは Performance Co-Pilot アーカイブファイルのいずれかから収集されます。

pmlc

アクティブな **pmlogger** インスタンスを対話的に設定します。

pmlogcheck

Performance Co-Pilot アーカイブファイルで無効なデータを特定します。

pmlogconf

pmlogger 設定ファイルを作成および変更します。

pmlogctl

pmlogger のプライマリー以外のインスタンスを管理します。

pmloglabel

Performance Co-Pilot アーカイブファイルのラベルを検証、変更、または修復します。

pmlogsummary

Performance Co-Pilot アーカイブファイルに格納されたパフォーマンスメトリックに関する統計情報を計算します。

pmprobe

パフォーマンスメトリックの可用性を決定します。

pmsocks

ファイアウォールを介して Performance Co-Pilot ホストへのアクセスを許可します。

pmstat

システムパフォーマンスの簡単な概要を定期的に表示します。

pmstore

パフォーマンスメトリックの値を変更します。

pmtrace

トレース PMDA のコマンドラインインターフェイスを提供します。

pmval

パフォーマンスメトリックの現在の値を表示します。

別途インストールする pcp-system-tools パッケージで配布されるツール

pcp-atop

パフォーマンスの観点から最も重要なハードウェアリソース (CPU、メモリー、ディスク、およびネットワーク) のシステムレベルの占有を表示します。

pcp-atopsar

さまざまなシステムリソースの使用状況に関するシステムレベルのアクティビティーレポートを生成します。このレポートは、**pmlogger** または **pcp-atop** の **-w** オプションを使用してあらかじめ記録された生のログファイルから生成されます。

pcp-dmcache

設定されたデバイスマッパーキャッシュターゲット (デバイスの IOP、キャッシュデバイスとメタデータデバイスの使用率、各キャッシュデバイスの読み取り/書き込みのヒット率とミス率、比率など) に関する情報を表示します。

pcp-dstat

一度に 1 台のシステムのメトリックを表示します。複数のシステムのメトリックを表示するには、**--host** オプションを使用します。

pcp-free

システム内の空きメモリーと使用済みメモリーを報告します。

pcp-htop

システム上で実行されているすべてのプロセスとそのコマンドライン引数を、**top** コマンドと同様の形式で表示しますが、縦横にスクロールしたり、マウスで操作したりすることができます。また、プロセスをツリー形式で表示したり、複数のプロセスを選択して一度に処理することもできます。

pcp-ipcs

呼び出しプロセスが読み取りアクセスできる inter-process communication (IPC) ファシリティーの情報を表示します。

pcp-mpstat

CPU および割り込み関連の統計情報を報告します。

pcp-numastat

カーネルのメモリーアロケータからの NUMA 割り当て統計を表示します。

pcp-pidstat

システム上で動作している個々のタスクやプロセスに関する情報を表示します (CPU パーセンテージ、メモリーやスタックの使用率、スケジューリング、優先度など)。デフォルトでは、ローカルホストのライブデータを報告します。

pcp-shping

pmdashping Performance Metrics Domain Agent (PMDA) がエクスポートした shell-ping サービスメトリクスをサンプリングして報告します。

pcp-ss

pmdasockets PMDA が収集したソケットの統計情報を表示します。

pcp-tapestat

テープデバイスの I/O 統計情報を報告します。

pcp-uptime

システムの稼働時間、現在ログオンしているユーザー数、過去 1 分、5 分、15 分のシステム負荷の平均値を表示します。

pcp-verify

Performance Co-Pilot コレクターのインストールのさまざまな側面を検査し、特定の動作モードに対して正しく設定されているかを報告します。

pmiostat

SCSI デバイス (デフォルト) またはデバイスマッパーデバイス (**-x** デバイスマッパーオプションを使用) の I/O 統計情報を報告します。

pmrep

選択した、簡単にカスタマイズ可能なパフォーマンスメトリック値に関するレポート。

別途インストールする pcp-gui パッケージで配布されるツール

pmchart

Performance Co-Pilot の機能を介して利用可能なパフォーマンスメトリック値を描画します。

pmdumptext

ライブまたは Performance Co-Pilot アーカイブから収集されたパフォーマンスメトリックの値を出力します。

別途インストールする pcp-devel パッケージで配布されるツール

pmclient

PMAPI (Performance Metrics Application Programming Interface) を使用して、高水準のシステムパフォーマンスメトリックを表示します。

pmdbg

利用可能な Performance Co-Pilot デバッグ制御フラグとその値を表示します。

pmerr

利用可能な Performance Co-Pilot エラーコードと、それに対応するエラーメッセージを表示します。

6.5. PCP デプロイメントのアーキテクチャー

Performance Co-Pilot (PCP) は、PCP デプロイメントの規模に基づいて、複数のデプロイメントアーキテクチャーをサポートし、高度なセットアップを実現するための多くのオプションを提供します。

Red Hat によって設定された推奨デプロイメント、サイジング係数、および設定オプションに基づいた、利用可能なスケーリングデプロイメントセットアップバリアントには、以下が含まれます。

注記

PCP バージョン 5.3.0 は Red Hat Enterprise Linux 8.4 および Red Hat Enterprise Linux 8 の以前のマイナーバージョンでは利用できないため、Red Hat はローカルホストおよび pmlogger のファームアーキテクチャーを推奨します。

PCP 5.3.0 以前のバージョンにおける pmproxy の既知のメモリーリークについては、Memory leaks in pmproxy in PCP を参照してください。

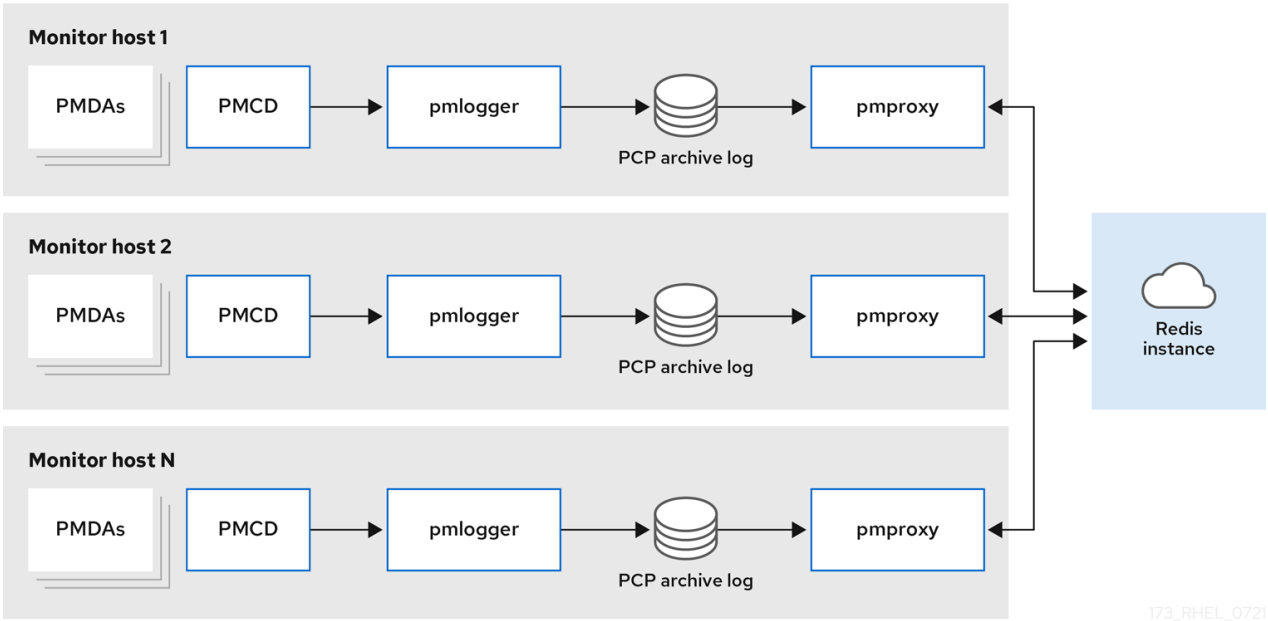
ローカルホスト

各サービスは監視対象のマシン上でローカルに動作します。設定を変更せずにサービスを開始した場合、これがデフォルトのデプロイメントです。この場合、個々のノードを超えたスケーリングはできません。
デフォルトでは、Redis のデプロイメント設定は、スタンドアロン、localhost となっています。しかし、Redis はオプションとして、データを複数のホストで共有する、高可用性と高スケーラビリティを備えたクラスター形態で実行することができます。また、クラウド上に Redis クラスターをデプロイしたり、クラウドベンダーが提供するマネージド Redis クラスターを利用したりすることも可能です。

Decentralized

ローカルホストと分散型のセットアップの唯一の違いは、集中型の Redis サービスです。このモデルでは、ホストは監視対象の各ホスト上で **pmlogger** サービスを実行し、ローカルの **pmcd** インスタンスからメトリックを取得します。そして、ローカルの **pmproxy** サービスは、パフォーマンスメトリックを中央の Redis インスタンスにエクスポートします。

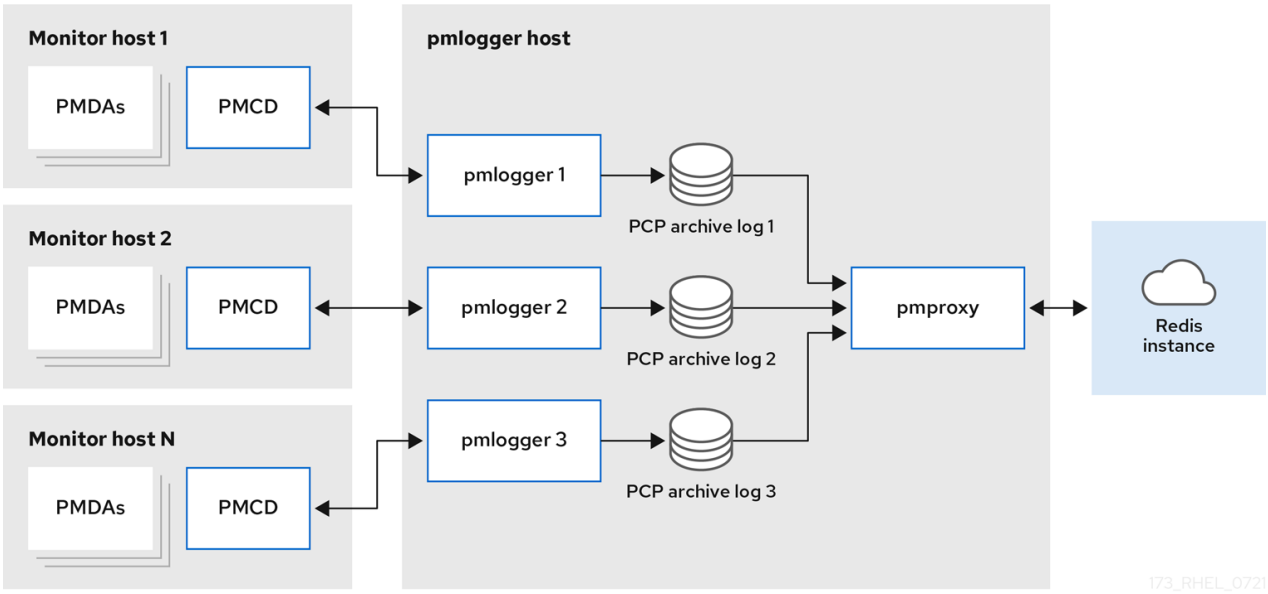
図6.1 分散型ロギング



集中型ロギング - pmlogger ファーム

監視対象ホストのリソース使用量が制限されている場合、**pmlogger** ファームというデプロイメントオプションもあります。これは集中型ロギングとも呼ばれます。この設定では、1 つのロガーホストが複数の **pmlogger** プロセスを実行し、それぞれが異なるリモート **pmcd** ホストからパフォーマンスメトリックを取得するように設定されます。集中ロガーのホストは **pmproxy** サービスを実行するように設定され、このサービスは、結果として生じる PCP アーカイブズのログを検出し、メトリックデータを Redis インスタンスに読み込みます。

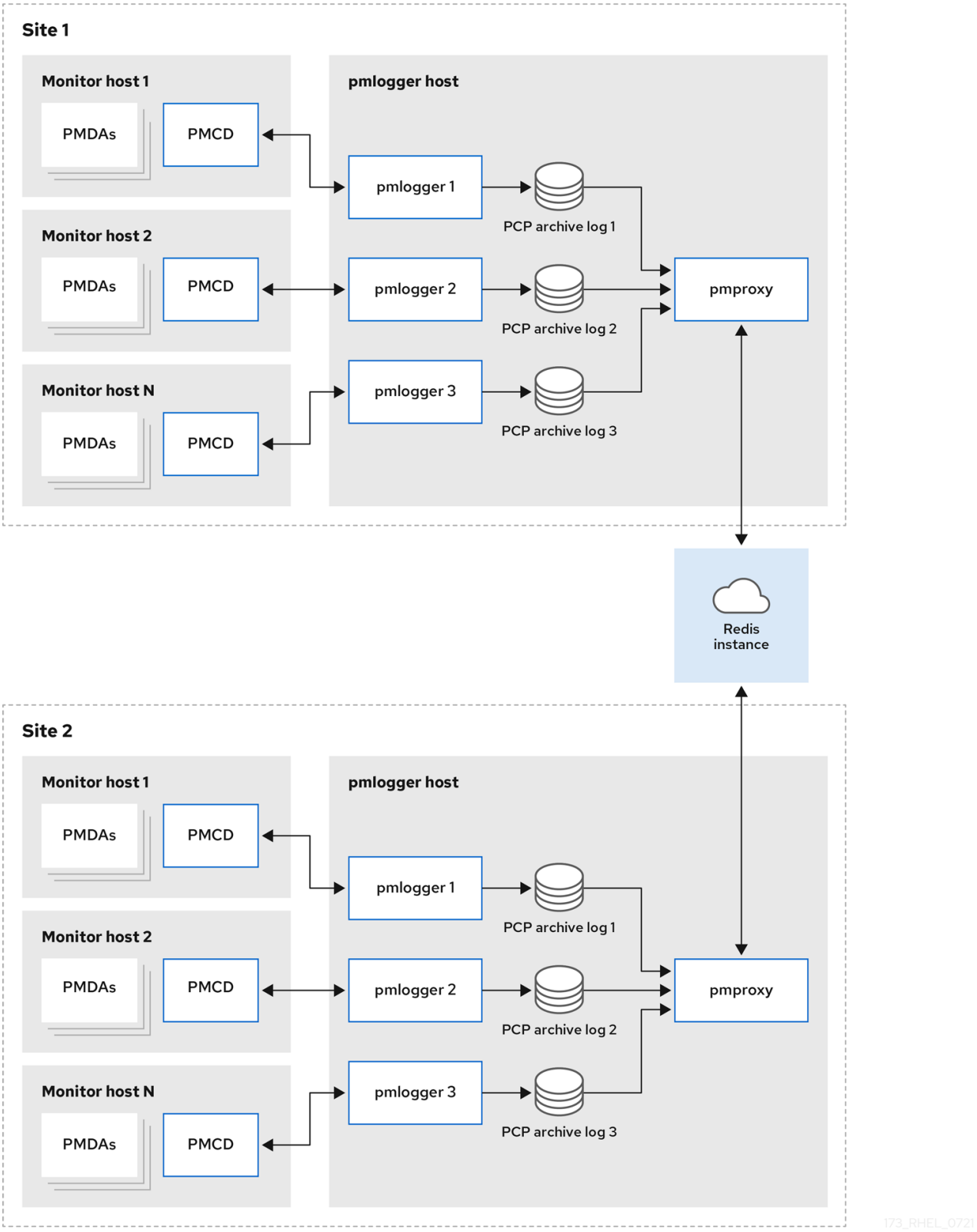
図6.2 集中型ロギング - pmlogger ファーム



統合型 - 複数の pmlogger ファーム

大規模なデプロイメントの場合、Red Hat は複数の **pmlogger** ファームを統合させてデプロイすることを推奨します。例えば、ラックやデータセンターごとに 1 つの **pmlogger** ファームをデプロイします。各 **pmlogger** ファームは、メトリックを中央の Redis インスタンスに読み込みます。

図6.3 統合型 - 複数の pmlogger ファーム



注記

デフォルトでは、Redis のデプロイメント設定は、スタンドアロン、localhost となっています。しかし、Redis はオプションとして、データを複数のホストで共有する、高可用性と高スケーラビリティを備えたクラスター形態で実行することができます。また、クラウド上に Redis クラスターをデプロイしたり、クラウドベンダーが提供するマネージド Redis クラスターを利用したりすることも可能です。

関連情報

- **pcp(1)**、**pmlogger(1)**、**pmproxy(1)**、および **pmcd(1)** の man ページ
- Recommended deployment architecture

6.6. 推奨されるデプロイメントアーキテクチャー

次の表は、監視するホストの数に応じて推奨されるデプロイメントアーキテクチャーを示しています。

表6.1 推奨されるデプロイメントアーキテクチャー

ホストの数 (N)	1-10	10-100	100-1000
PMCD サーバー	N	N	N
pmlogger サーバー	1 から N	N/10 から N	N/100 から N
pmproxy サーバー	1 から N	1 から N	N/100 から N
Redis サーバー	1 から N	1 から N/10	N/100 から N/10
Redis クラスター	No	Maybe	Yes
推奨されるデプロイメント設定	ローカルホスト、分散型、または集中型のロギング	分散型、集中型ロギング、または統合型	分散型または統合型

6.7. サイジングファクター

スケーリングに必要なサイジングファクターは以下のとおりです。

Remote system size

CPU、ディスク、ネットワーク・インターフェイスおよびその他のハードウェアリソースの数は、集中型ロギングホスト上の各 **pmlogger** が収集するデータ量に影響します。

Logged Metrics

ログメトリックの数と種類が重要なロールを果たします。具体的には、**per-process proc.*** メトリックには、大きなディスク容量が必要です。たとえば、標準的な **pcp-zeroconf** の設定で 10 秒のログ取得間隔の場合、proc メトリックなしでは 11MB、proc メトリックありでは 155MB と、係数は 10 倍以上になります。さらに、各メトリックのインスタンス数、たとえば CPU、ブロックデバイス、ネットワークインターフェイスの数なども、必要なストレージ容量に影響を与えます。

Logging Interval

メトリックのログを取る間隔は、ストレージの要件に影響します。各 **pmlogger** インスタンスの **pmlogger.log** ファイルには、毎日の PCP アーカイブファイルの予想サイズが書き込まれます。これらの値は圧縮されていない推定値です。PCP のアーカイブは約 10:1 と非常によく圧縮されるため、実際の長期的なディスク容量の要件は、特定のサイトで決定することができます。

pmlogrewrite

PCP をアップグレードするたびに **pmlogrewrite** ツールが実行され、旧バージョンと新バージョンの PCP でメトリックのメタデータに変更があった場合、古いアーカイブが書き換えられます。この処理時間は、保存されているアーカイブの数に応じてリニアに変化します。

関連情報

- **pmlogrewrite(1)** および **pmlogger(1)** の man ページ

6.8. PCP スケーリングの設定オプション

スケーリングに必要な設定オプションを以下に示します。

sysctl and rlimit settings

アーカイブ検出を有効にすると、**pmproxy** は、監視またはログテーリングを行っているすべての **pmlogger** に対して 4 つの記述子を必要とし、さらに、サービスログと **pmproxy** クライアントソケットのための追加のファイル記述子があれば、それも必要となります。各 **pmlogger** プロセスは、リモートの **pmcd** ソケット、アーカイブファイル、サービスログなどのために約 20 個のファイル記述子を使用します。合計すると、約 200 の **pmlogger** プロセスを実行しているシステムでは、デフォルトの 1024 ソフトの制限を超えてしまいます。**pcp-5.3.0** 以降の **pmproxy** サービスでは、ソフトリミットがハードリミットに自動的に引き上げられます。以前のバージョンの PCP では、多数の **pmlogger** プロセスをデプロイする場合、チューニングが必要です。これは、**pmlogger** のソフトリミットまたはハードリミットを増やすことで実現できます。詳細は、How to set limits (ulimit) for services run by systemd を参照してください。

ローカルアーカイブ

pmlogger サービスは、ローカルおよびリモートの **pmcd** のメトリックを **/var/log/pcp/pmlogger/** ディレクトリーに保存します。ローカルシステムのロギング間隔を制御するには、**/etc/pcp/pmlogger/control.d/configfile** ファイルを更新し、引数に **-t X** を追加してください (**X**は秒単位のロギング間隔)。どのメトリックを記録するかを設定するには、**pmlogconf /var/lib/pcp/config/pmlogger/config.clienthostname** を実行します。このコマンドは、デフォルトのメトリックのセットを含む設定ファイルをデプロイしますが、オプションでさらにカスタマイズすることもできます。古い PCP アーカイブをいつパージするかという保存設定を行うには、**/etc/sysconfig/pmlogger_timers** file and specify **PMLOGGER_DAILY_PARAMS="-E -k X"** を更新します。ここで、**X**は PCP アーカイブを保持する日数です。

Redis

pmproxy サービスは、**pmlogger** からのログされたメトリックを Redis インスタンスに送信します。設定ファイル **/etc/pcp/pmproxy/pmproxy.conf** で保持設定を指定する際に使用できる 2 つのオプションを以下に示します。

- **stream.expire** では、古いメトリックを削除するまでの期間を指定します (つまり、指定した秒数の間更新されなかったメトリック)。
- **stream.maxlen** は、ホストごとに 1 つのメトリックの最大メトリック値の数を指定します。この設定は、保存期間をログ間隔で割ったものでなければなりません。例えば、保存期間が 14 日、ログ間隔が 60 秒の場合は 20160 となります (60*60*24*14/60)。

関連情報

- **pmproxy(1)**、**pmlogger(1)**、および **sysctl(8)** の man ページ

6.9. 例: 集中ロギングデプロイメントの分析

以下の結果は、集約ロギングセットアップ (pmlogger ファームデプロイメントとも呼ばれる) で集約されています。デフォルトの **pcp-zeroconf 5.3.0** インストールでは、各リモートホストが、64 の CPU コア、376 GB RAM、および 1 つのディスクが接続されたサーバーで **pmcd** を実行している同一のコンテナーインスタンスになります。

ロギング間隔は 10 秒で、リモートノードの proc メトリックは含まれず、メモリー値は Resident Set Size (RSS) の値を参照します。

表6.2 10 秒のロギング間隔の詳細な使用統計

ホスト数	10	50
1 日あたりの PCP アーカイブストレージ	91 MB	522 MB
pmlogger メモリー	160 MB	580 MB
1 日あたりの **pmlogger** ネットワーク (In)	2 MB	9 MB
pmproxy メモリー	1.4 GB	6.3 GB
1 日あたりの Redis メモリー	2.6 GB	12 GB

表6.3 60 秒のロギング間隔で、監視対象ホストに応じて使用されるリソース

ホスト数	10	50	100
1 日あたりの PCP アーカイブストレージ	20 MB	120 MB	271 MB
pmlogger メモリー	104 MB	524 MB	1049 MB
1 日あたりの **pmlogger** ネットワーク (In)	0.38 MB	1.75 MB	3.48 MB
pmproxy メモリー	2.67 GB	5.5GB	9 GB
1 日あたりの Redis メモリー	0.54 GB	2.65 GB	5.3 GB

注記

pmproxy は Redis 要求をキューに入れ、Redis パイプラインを使用して Redis クエリーを高速化します。これにより、メモリー使用率が高くなる可能性があります。この問題をトラブルシューティングする場合は、Troubleshooting high memory usage を参照してください。

6.10. 例: 統合型セットアップデプロイメントの分析

以下の結果が、統合型セットアップ (複数の **pmlogger** ファームとも呼ばれる) で確認されました。これは、3 つの集中ロギング (**pmlogger** ファーム) セットアップで設定されます。各 **pmlogger** ファームは 100 のリモートホスト、つまり合計 300 のホストを監視していました。

pmlogger ファームのこのセットアップは、Redis サーバーがクラスターモードで動作していたことを除いて、60 秒のロギング間隔での

例: 集中ロギングデプロイメントの分析 で説明した設定と同じです。

表6.4 60 秒のロギング間隔で、統合型ホストに応じて使用されるリソース

1 日あたりの PCP アーカイブストレージ	pmlogger メモリー	1 日あたりのネットワーク (In/Out)	pmproxy メモリー	1 日あたりの Redis メモリー
277 MB	1058 MB	15.6 MB / 12.3 MB	6-8 GB	5.5 GB

ここでは、すべての値はホストごとになります。Redis クラスターのノード間通信により、ネットワーク帯域幅が高まります。

6.11. 高メモリー使用率のトラブルシューティング

以下のシナリオでは、メモリー使用率が高くなる可能性があります。

- **pmproxy** プロセスは新しい PCP アーカイブの処理がビジーで、Redis の要求および応答を処理するための予備の CPU サイクルがありません。
- Redis ノードまたはクラスターが過負荷になり、時間が経過しても着信要求を処理できません。

pmproxy サービスデーモンは、Redis ストリームを使用し、設定パラメーター (PCP チューニングパラメーター) をサポートします。これは、Redis のメモリー使用量および鍵の保存に影響します。**/etc/pcp/pmproxy/pmproxy.conf** ファイルには、**pmproxy** で利用可能な設定オプションと、関連する API がリスト表示されます。

次の手順では、メモリー使用率が高い問題をトラブルシューティングする方法について説明します。

前提条件

1. **pcp-pmda-redis** パッケージをインストールします。

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

2. redis PMDA をインストールします。

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

手順

- 高いメモリー使用率のトラブルシューティングを行うには、次のコマンドを実行して、**inflight** 列を確認します。

```
$ 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
```

この列は、Redis リクエストが転送中である数を示しています。つまり、キューに入れられているか送信されており、現時点では応答は受信されていません。

数値が高い場合は、次のいずれかの状態を示します。

  - **pmproxy** プロセスは新しい PCP アーカイブの処理がビジーで、Redis の要求および応答を処理するための予備の CPU サイクルがありません。

  - Redis ノードまたはクラスターが過負荷になり、時間が経過しても着信要求を処理できません。

- メモリー使用量が多い問題のトラブルシューティングを行うには、このファームの **pmlogger** プロセスの数を減らし、別の pmlogger ファームを追加します。統合型 (複数の pmlogger ファームの設定) を使用します。
Redis ノードが長時間にわたって CPU を 100% 使用している場合は、パフォーマンスが向上しているホストに移動するか、代わりにクラスター化された Redis 設定を使用します。

- **pmproxy.redis.*** メトリックスを表示するには、次のコマンドを使用します。

```
$ 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
[...]
```

インフライトのリクエスト数を表示するには、**pmproxy.redis.requests.inflight.total** メトリックスと **pmproxy.redis.requests.inflight.bytes** メトリックスを参照して、現在のすべてのインフライトの Redis リクエストで占有されているバイト数を表示します。

通常、redis 要求キューは 0 ですが、大きな pmlogger ファームの使用量に基づいて構築できます。これによりスケーラビリティーが制限され、**pmproxy** クライアントのレイテンシーが高くなる可能性があります。

- **pminfo** コマンドを実行すると、パフォーマンスメトリックスの詳細が表示されます。たとえば、**redis.*** メトリックスを表示するには、次のコマンドを使用します。

```
$ 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
[...]
```

ピークメモリー使用量を表示するには、**redis.used_memory_peak** メトリックスを参照してください。

関連情報

- man ページの **pmdaredis(1)**、**pmproxy(1)**、および **pminfo(1)**
- PCP デプロイメントのアーキテクチャー

第7章 PMLOGGER でのパフォーマンスデータのロギング

PCP ツールを使用してパフォーマンスのメトリック値をログに記録すると、後で再生できます。これにより、遡及的なパフォーマンス解析を実行できます。

pmlogger ツールを使用すると、以下が可能になります。

- 選択したメトリックのアーカイブログをシステムに作成する
- システムに記録されるメトリックとその頻度を指定する

7.1. PMLOGCONF で PMLOGGER 設定ファイルの変更

pmlogger サービスの実行中、PCP はホストでメトリックのデフォルトセットをログに記録します。

pmlogconf ユーティリティーを使用してデフォルト設定を確認します。**pmlogger** 設定ファイルが存在しない場合は、**pmlogconf** がデフォルトのメトリック値で作成します。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。

手順

1. **pmlogger** 設定ファイルを作成または変更します。

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

2. **pmlogconf** プロンプトに従い、関連するパフォーマンスメトリックのグループを有効または無効にし、有効な各グループのロギング間隔を制御します。

関連情報

- **pmlogconf(1)** および **pmlogger(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール

7.2. PMLOGGER の設定ファイルの手動編集

指定したメトリックと間隔でカスタマイズしたロギング設定を作成する場合は、**pmlogger** 設定ファイルを手動で編集します。デフォルトの **pmlogger** 設定ファイルは **/var/lib/pcp/config/pmlogger/config.default** です。設定ファイルでは、プライマリーのロギングインスタンスによって記録されるメトリックを指定します。

手動の設定では、以下が可能になります。

- 自動設定のリストに記載されていないメトリックを記録する。
- カスタムロギングの周波数を選択する。
- アプリケーションのメトリックを使用して PMDA を追加する。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。

手順

- **/var/lib/pcp/config/pmlogger/config.default** ファイルを開いて編集し、特定のメトリックを追加します。

```
# 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;
```

関連情報

- **pmlogger(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール

7.3. PMLOGGER サービスの有効化

ローカルマシンでメトリック値のログを記録するには、**pmlogger** サービスを開始して有効にする必要があります。

この手順では、**pmlogger** サービスを有効にする方法を説明します。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。

手順

- **pmlogger** サービスを開始して、有効にします。

```
# systemctl start pmlogger

# systemctl enable pmlogger
```

検証手順

- **pmlogger** サービスが有効になっているかどうかを確認します。

```
# 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
```

関連情報

- **pmlogger(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール
- **/var/lib/pcp/config/pmlogger/config.default** ファイル

7.4. メトリクス収集のためのクライアントシステムの設定

この手順では、中央サーバーが、PCP を実行しているクライアントからメトリックを収集できるように、クライアントシステムを設定する方法を説明します。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。

手順

1. **pcp-system-tools** パッケージをインストールします。

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

2. **pmcd** の IP アドレスを設定します。

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

192.168.4.62 を、クライアントがリッスンする IP アドレスに置き換えます。

デフォルトでは、**pmcd** は、ローカルホストをリッスンします。

3. パブリック **zone** を永続的に追加するように、ファイアウォールを設定します。

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

# firewall-cmd --reload
success
```

4. SELinux ブール値を設定します。

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

5. **pmcd** サービスおよび **pmlogger** サービスを有効にします。

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

検証手順

- **pmcd** が、設定した IP アドレスを正しくリッスンしているかどうかを確認します。

```
# 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))
```

関連情報

- **pmlogger(1)**、**firewall-cmd(1)**、**ss(8)**、および **setsebool(8)** の man ページ
- PCP で配布されるシステムサービスおよびツール
- **/var/lib/pcp/config/pmlogger/config.default** ファイル

7.5. データ収集用の中央サーバーの設定

この手順では、PCP を実行しているクライアントからメトリックを収集する中央サーバーを作成する方法を説明します。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。
- クライアントがメトリック収集用に設定されている。詳細は、メトリクス収集のためのクライアントシステムの設定 を参照してください。

手順

1. **pcp-system-tools** パッケージをインストールします。

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

2. 以下の内容で **/etc/pcp/pmlogger/control.d/remote** ファイルを作成してください。

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

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

192.168.4.13、192.168.4.14、および 192.168.4.62 を、クライアントの IP アドレスに置き換えます。

注記

Red Hat Enterpirse Linux 8.0、8.1、および 8.2 では、制御ファイルでリモートホストに PCP_LOG_DIR/pmlogger/**host_name** 形式を使用します。

3. **pmcd** サービスおよび **pmlogger** サービスを有効にします。

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

検証手順

- 各ディレクトリーから最新のアーカイブファイルにアクセスできることを確認します。

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

/var/log/pcp/pmlogger/ ディレクトリーのアーカイブファイルは、詳細な分析とグラフ作成に使用できます。

関連情報

- **pmlogger(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール
- **/var/lib/pcp/config/pmlogger/config.default** ファイル

7.6. SYSTEMD ユニットと PMLOGGER

pmlogger サービスを、それ自体を監視する単一のホストとして、または複数のリモートホストからメトリクスを収集する単一のホストを含む **pmlogger** ファームとしてデプロイすると、関連する **systemd** サービスとタイマーユニットがいくつか自動的にデプロイされます。これらのサービスとタイマーは、**pmlogger** インスタンスが実行していることを確認するための定期的なチェックを提供し、不足しているインスタンスを再起動し、ファイル圧縮などのアーカイブ管理を実行します。

pmlogger によって通常展開されるチェックおよびハウスキーピングサービスは次のとおりです。

pmlogger_daily.service

デフォルトでは、毎日、深夜直後に実行され、1つ以上の PCP アーカイブセットを集約、圧縮、およびローテートします。また、制限 (デフォルトでは 2 週間) よりも古いアーカイブも削除されます。**pmlogger.service** ユニットに必要な **pmlogger_daily.timer** ユニットによってトリガーされます。

pmlogger_check

pmlogger インスタンスが実行中であるかどうかを 30 分ごとにチェックします。不足しているインスタンスを再起動し、必要な圧縮タスクを実行します。**pmlogger.service** ユニットに必要な **pmlogger_check.timer** ユニットによってトリガーされます。

pmlogger_farm_check

設定されたすべての **pmlogger** インスタンスのステータスを確認します。不足しているインスタンスを再起動します。すべての非プライマリーインスタンスを **pmlogger_farm** サービスに移行します。**pmlogger_farm_check.timer** によってトリガーされます。これは、**pmlogger_farm.service** ユニットによって必要とされ、pmlogger_farm.service ユニット自体は **pmlogger.service** ユニットによって必要とされます。

これらのサービスは一連の肯定的な依存関係を通じて管理されます。つまり、プライマリー **pmlogger** インスタンスをアクティブ化すると、すべて有効になります。**pmlogger_daily.service** はデフォルトで無効になっていますが、**pmlogger.service** との依存関係によって **pmlogger_daily.timer** がアクティブになると、**pmlogger_daily.service** の実行がトリガーされることに注意してください。

pmlogger_daily は、マージ前にアーカイブを自動的に書き換えるために **pmlogrewrite** とも統合されています。これにより、実稼働環境や PMDA が変化する中でもメタデータの一貫性を確保できます。たとえば、ログ記録間隔中に監視対象ホストの1台で **pmcd** が更新されると、ホスト上の一部のメトリクスのセマンティクスが更新され、新しいアーカイブがそのホストから以前に記録されたアーカイブと互換性がなくなる可能性があります。詳細は、**pmlogrewrite(1)** の man ページを参照してください。

pmlogger によってトリガーされる systemd サービスの管理

pmlogger インスタンスによって収集されたデータ用の自動化されたカスタムアーカイブ管理システムを作成できます。これは制御ファイルを使用して行われます。これらの制御ファイルは次のとおりです。

- プライマリー **pmlogger** インスタンスの場合:
  - **etc/pcp/pmlogger/control**
  - **/etc/pcp/pmlogger/control.d/local**
- リモートホストの場合:
  - **/etc/pcp/pmlogger/control.d/remote**
    remote を希望のファイル名に置き換えます。

    注記

    プライマリー **pmlogger** インスタンスは、接続先の **pmcd** と同じホストで実行している必要があります。1つのセントラルホストがリモートホストで実行している **pmcd** イン

スタンスに接続された複数の **pmlogger** インスタンスでデータを収集している場合は、プライマリーインスタンスは必要ありません。また、設定でプライマリーインスタンスが必要ない場合もあります。

ファイルには、ログに記録するホストごとに 1 行が含まれている必要があります。自動的に作成されるプライマリーロガーインスタンスのデフォルトの形式は次のようになります。

```
# === LOGGER CONTROL SPECIFICATIONS ===
#
#Host     P?  S?   directory      args

# local primary logger
LOCALHOSTNAME    y   n    PCP_ARCHIVE_DIR/LOCALHOSTNAME    -r -T24h10m -c
config.default -v 100Mb
```

フィールドの詳細は以下のとおりです。

ホスト

ログに記録するホスト名

P?

"Primary?" の略です。このフィールドは、ホストがプライマリーロガーインスタンスである (**y**) か、そうでない (**n**) かを示します。設定内のすべてのファイルにわたってプライマリーロガーは 1 つだけ存在でき、接続先の **pmcd** と同じホスト上で実行している必要があります。

S?

"Socks?" の略です。このフィールドは、このロガーインスタンスがファイアウォール経由で **pmcd** に接続するために **SOCKS** プロトコルを使用する必要がある (**y**) か、必要がない (**n**) かを示します。

directory

この行に関連付けられたすべてのアーカイブがこのディレクトリーに作成されます。

args

pmlogger に渡される引数。
args フィールドのデフォルト値は次のとおりです。

-r

アーカイブのサイズと増加率を報告します。

T24h10m

各日のログ記録を終了するタイミングを指定します。これは通常、**pmlogger_daily.service** が実行する時間です。デフォルト値の **24h10m** は、ログ記録が開始してから遅くとも 24 時間 10 分後に終了することを示します。

-c config.default

使用する設定ファイルを指定します。これは基本的に、記録するメトリクスを定義します。

-v 100Mb

1 つのデータボリュームがいっぱいになり、別のボリュームが作成されるサイズを指定します。新しいアーカイブに切り替わった後、以前に記録されたものは **pmlogger_daily** または **pmlogger_check** のいずれかによって圧縮されます。

関連情報

- **pmlogger(1)** の man ページ
- **pmlogger_daily(1)** の man ページ

- **pmlogger_check(1)** の man ページ
- **pmlogger.control(5)** の man ページ
- **pmlogrewrite(1)** の man ページ

7.7. PMREP で PCP ログアーカイブの再生

メトリックデータの記録後、PCP ログアーカイブを再生できます。ログをテキストファイルにエクスポートして、スプレッドシートにインポートするには、**pcp2csv**、**pcp2xml**、**pmrep** または **pmlogsummary** などの PCP ユーティリティーを使用します。

pmrep ツールを使用すると、以下のことが可能になります。

- ログファイルを表示する
- 選択した PCP ログアーカイブを解析し、値を ASCII テーブルにエクスポートする
- アーカイブログ全体をデプロイメントするか、コマンドラインで個別のメトリックを指定して、ログからメトリック値のみを選択する

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。
- **pmlogger** サービスが有効になっている。詳細は、pmlogger サービスの有効化 を参照してください。
- **pcp-system-tools** パッケージをインストールします。

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

手順

- メトリックのデータを表示します。

```
$ 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
```

上記の例では、**5 秒** 間隔でアーカイブに収集された **disk.dev.write** メトリックスのデータをコンマ区切り値の形式で表示します。

注記

この例の **20211128** を、データを表示する **pmlogger** アーカイブを含むファイル名に置き換えます。

関連情報

- man ページの **pmlogger(1)**、**pmrep(1)**、および **pmlogsummary(1)**
- PCP で配布されるシステムサービスおよびツール

第8章 PERFORMANCE CO-PILOT によるパフォーマンスの監視

Performance Co-Pilot (PCP) は、システムレベルのパフォーマンス測定を監視、視覚化、保存、および分析するためのツール、サービス、およびライブラリーのスイートです。

システム管理者は、Red Hat Enterprise Linux 8 の PCP アプリケーションを使用して、システムのパフォーマンスを監視できます。

8.1. PMDA-POSTFIX での POSTFIX の監視

この手順では、**pmda-postfix** を使用して **postfix** メールサーバーのパフォーマンスメトリックを監視する方法を説明します。これは、1 秒間に受信した電子メールの数を確認するのに役立ちます。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。
- **pmlogger** サービスが有効になっている。詳細は、pmlogger サービスの有効化 を参照してください。

手順

1. 以下のパッケージをインストールします。
   a. **pcp-system-tools** をインストールします。

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

   b. **pmda-postfix** パッケージをインストールして、**postfix** を監視します。

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

   c. ロギングデーモンをインストールします。

   ```
   # yum install rsyslog
   ```

   d. テスト用にメールクライアントをインストールします。

   ```
   # yum install mutt
   ```

2. **postfix** サービスおよび **rsyslog** サービスを有効にします。

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

3. SELinux ブール値を有効にして、**pmda-postfix** が必要なログファイルにアクセスできるようにします。

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

4. **PMDA** をインストールします。

```
# 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
```

検証手順

- **pmda-postfix** 操作を確認します。

```
echo testmail | mutt root
```

- 利用可能なメトリックを確認します。

```
# pminfo postfix

postfix.received
postfix.sent
postfix.queues.incoming
postfix.queues.maildrop
postfix.queues.hold
postfix.queues.deferred
postfix.queues.active
```

関連情報

- **rsyslogd(8)**、**postfix(1)**、および **setsebool(8)** の man ページ
- PCP で配布されるシステムサービスおよびツール

8.2. PCP CHARTS アプリケーションで PCP ログアーカイブを視覚的にトレース

メトリックデータの記録後、PCP ログアーカイブをグラフとして再生できます。メトリックは、PCP ログアーカイブのメトリックデータを履歴データのソースとして使用する代替オプションを持つ 1 台または複数のライブホストから提供されます。**PCP Charts** アプリケーションインターフェイスをカスタマイズしてパフォーマンスメトリックのデータを表示するには、ラインプロット、バーグラフ、または使用状況グラフを使用します。

PCP Charts アプリケーションを使用すると、以下が可能になります。

- **PCP Charts** アプリケーションのデータを再生し、グラフを使用して、システムのライブデータとともに遡及データを視覚化する。
- パフォーマンスメトリック値をグラフに描画する。
- 複数のチャートを同時に表示する。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。
- **pmlogger** でパフォーマンスデータをログに記録している。詳細は、pmlogger でのパフォーマンスデータのロギング を参照してください。
- **pcp-gui** パッケージがインストールされている。

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

手順

1. コマンドラインで **PCP Charts** アプリケーションを起動します。

```
# pmchart
```

図8.1 PCP Charts アプリケーション

pmtime サーバー設定は下部にあります。**start** ボタンおよび **pause** ボタンを使用すると、以下を制御できます。

- PCP がメトリックデータをポーリングする間隔
- 履歴データのメトリックの日付および時間

2. **File** をクリックしてから、**New Chart** をクリックして、ホスト名またはアドレスを指定して、ローカルマシンおよびリモートマシンの両方からメトリックを選択します。高度な設定オプションには、チャートの軸値を手動で設定する機能、およびプロットの色を手動で選択する機能が含まれます。
3. **PCP Charts** アプリケーションで作成したビューを記録します。
以下は、**PCP Charts** アプリケーションで作成したイメージを撮影したり、ビューを記録するためのオプションです。
    - **File** をクリックしてから **Export** をクリックして、現在のビューのイメージを保存します。

- **Record** をクリックしてから **Start** をクリックし、録画を開始します。**Record** をクリックしてから **Stop** をクリックし、録画を停止します。録画の停止後、記録されたメトリックは後で表示できるようにアーカイブが作成されます。

4. 必要に応じて、**PCP Charts** アプリケーションでは、**ビュー** と呼ばれるメインの設定ファイルによって、1 つ以上のチャートに関連付けられたメタデータを保存できます。このメタデータでは、使用されるメトリックや、チャート列など、チャート側面をすべて記述します。**File** をクリックしてから **Save View** をクリックして、カスタム **view** 設定を保存し、後で **view** 設定を読み込みます。
以下の **PCP Charts** アプリケーションビューの設定ファイルの例では、指定の XFS ファイルシステム **loop1** に対して読み書きされた合計バイト数を示す積み上げチャートグラフを説明します。

```
#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"
```

関連情報

- **pmchart(1)** および **pmtime(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール

8.3. PCP を使用した SQL SERVER からのデータの収集

Red Hat Enterprise Linux 8.2 以降では、SQL Server エージェントは Performance Co-Pilot (PCP) で利用できます。これは、データベースのパフォーマンスの問題を監視および分析するのに役立ちます。

この手順では、システムの **pcp** を使用して Microsoft SQL Server のデータを収集する方法を説明します。

前提条件

- Red Hat Enterprise Linux に Microsoft SQL Server をインストールし、SQL Server への信頼できる接続を確立している。
- Red Hat Enterprise Linux 用の SQL Server の Microsoft ODBC ドライバーがインストールされている。

手順

1. PCP をインストールします。

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

2. **pyodbc** ドライバーに必要なパッケージをインストールします。

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

# yum install python3-pyodbc
```

3. **mssql** エージェントをインストールします。

   a. PCP の Microsoft SQL Server ドメインエージェントをインストールします。

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

   b. **/etc/pcp/mssql/mssql.conf** ファイルを編集して、**mssql** エージェントの SQL サーバーアカウントのユーザー名およびパスワードを設定します。設定するアカウントに、パフォーマンスデータに対するアクセス権限があることを確認します。

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

   user_name を SQL Server アカウントに置き換え、**user_password** をこのアカウントの SQL Server ユーザーパスワードに置き換えます。

4. エージェントをインストールします。

```
# 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
[...]
```

検証手順

- **pcp** コマンドを使用して、SQL Server PMDA (**mssql**) が読み込まれて実行されていることを確認します。

```
$ 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 が SQL Server から収集できるメトリックの完全なリストを表示します。

```
# pminfo mssql
```

- メトリックのリストを表示した後は、トランザクションのレートを報告できます。たとえば、5 秒間の時間枠で、1秒あたりの全体的なトランザクション数を報告するには、以下のコマンドを実行します。

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

- **pmchart** コマンドを使用して、システムでこれらのメトリックのグラフィックチャートを表示します。詳細は、Visually tracing PCP log archives with the PCP Charts application を参照してください。

関連情報

- **pcp(1)**、**pminfo(1)**、**pmval(1)**、**pmchart(1)**、および **pmdamssql(1)** の man ページ
- Performance Co-Pilot for Microsoft SQL Server with RHEL 8.2 (Red Hat Developers Blog)

第9章 PCP を使用した XFS のパフォーマンス分析

XFS PMDA は、**pcp** パッケージの一部として提供され、インストール時にデフォルトで有効になります。これは、Performance Co-Pilot (PCP) で XFS ファイルシステムのパフォーマンスメトリックデータを収集するために使用されます。

PCP を使用して、XFS ファイルシステムのパフォーマンスを分析できます。

9.1. XFS PMDA の手動インストール

XFS PMDA が **pcp** 設定出力に記載されていない場合は、PMDA エージェントを手動でインストールします。

この手順では、PMDA エージェントを手動でインストールする方法を説明します。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。

手順

1. xfs ディレクトリーに移動します。

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

2. XFS PMDA を手動でインストールします。

```
xfs]# ./Install

You will need to choose an appropriate configuration for install of
the “xfs” Performance Metrics Domain Agent (PMDA).

  collector     collect performance statistics on this system
  monitor       allow this system to monitor local and/or remote systems
  both          collector and monitor configuration for this system

Please enter c(ollector) or m(onitor) or (both) [b]
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 xfs metrics have appeared ... 149 metrics and 149 values
```

3. collector の場合は **c** を、monitor の場合は **m** を、またはこれら両方の場合は **b** を入力して、PMDA ロールを選択します。PMDA インストールスクリプトから、以下の PMDA ロールのいずれかを指定するように求められます。

   - **collector** ロールを指定すると、現在のシステムでパフォーマンスメトリックを収集できます。

   - **monitor** ロールを指定すると、システムがローカルシステム、リモートシステム、またはその両方を監視できるようになります。

デフォルトオプションは **collector** と **monitor** の両方です。これにより、ほとんどのシナリオで XFS PMDA は適切に動作できます。

検証手順

- **pmcd** プロセスがホストで実行しており、設定リストに XFS PMDA が有効として記載されていることを確認します。

```
# 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
```

関連情報

- **pmcd(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール

9.2. PMINFO を使用した XFS パフォーマンスメトリックの検証

PCP は XFS PMDA を有効にして、マウントされた各 XFS ファイルシステムに対して特定の XFS メトリックの報告を可能にします。これにより、特定のマウントされたファイルシステムの問題を特定して、パフォーマンスを評価することが容易になります。

pminfo コマンドは、マウントされた各 XFS ファイルシステムの各デバイスに対する XFS メトリックを提供します。

この手順では、XFS PMDA が提供する利用可能なすべてのメトリックのリストを表示します。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。

手順

- XFS PMDA が提供する利用可能なメトリックのリストを表示します。

```
# pminfo xfs
```

- 個別のメトリックの情報を表示します。以下の例は、**pminfo** ツールを使用して、特定の XFS の **read** メトリックおよび **write** メトリックを検証します。
  - **xfs.write_bytes** メトリックの簡単な説明を表示します。

```
# pminfo --oneline xfs.write_bytes

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

- **xfs.read_bytes** メトリックの長い説明を表示します。

```
# 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** メトリックの現在のパフォーマンス値を取得します。

```
# pminfo --fetch xfs.read_bytes

xfs.read_bytes
    value 4891346238
```

- **pminfo** で、デバイスごとの XFS メトリックを取得します。

```
# 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
```

関連情報

- **pminfo(1)** の man ページ
- XFS の PCP メトリックグループ
- XFS のデバイスごとの PCP メトリックグループ

9.3. PMSTORE を使用した XFS パフォーマンスメトリックのリセット

PCP を使用すると、特に特定のメトリックが、**xfs.control.reset** メトリックなどの制御変数として動作する場合は、そのメトリックの値を変更できます。メトリックの値を変更するには、**pmstore** ツールを使用します。

この手順では、**pmstore** ツールを使用して XFS メトリックをリセットする方法を説明します。

前提条件

- PCP がインストールされている。詳細は PCP のインストールおよび有効化 を参照してください。

手順

1. メトリックの値を表示します。

```
$ pminfo -f xfs.write

xfs.write
    value 325262
```

2. すべての XFS メトリックをリセットします。

```
# pmstore xfs.control.reset 1

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

検証手順

- メトリックをリセットした後に情報を表示します。

```
$ pminfo --fetch xfs.write

xfs.write
    value 0
```

関連情報

- **pmstore(1)** および **pminfo(1)** の man ページ
- PCP で配布されるシステムサービスおよびツール
- XFS の PCP メトリックグループ

9.4. XFS の PCP メトリックグループ

以下の表は、XFS で利用可能な PCP メトリックグループについて説明しています。

表9.1 XFS のメトリックグループ

メトリックグループ	提供されたメトリック
xfs.*	読み書き操作の数、読み書きバイト数を含む一般的な XFS メトリック。inode がフラッシュされた回数、クラッシュした回数、クラスター化に失敗した数に関するカウンターを併用。

xfs.allocs.* **xfs.alloc_btree.***	ファイルシステムのオブジェクトの割り当てに関するメトリックの範囲。これには、エクステントおよびブロックの作成/解放の数が含まれます。割り当てツリーの検索と、拡張レコードの作成と btree からの削除との比較。
xfs.block_map.* **xfs.bmap_btree.***	メトリックには、ブロックマップの読み取り/書き込みとブロックの削除の数、挿入、削除、および検索のためのエクステントリスト操作が含まれます。また、ブロックマップからの比較、検索、挿入、および削除に関する操作カウンター。
xfs.dir_ops.*	作成、エントリー削除、getdent の操作の数に対する XFS ファイルシステムのディレクトリー操作のカウンター。
xfs.transactions.*	メタデータトランザクションの数のカウンター。これには、空のトランザクションの数と、同期および非同期のトランザクションの数のカウントが含まれます。
xfs.inode_ops.*	オペレーティングシステムが、複数の結果で inode キャッシュの XFS inode を検索する回数のカウンター。このカウントキャッシュのヒット数、キャッシュミスなど。
xfs.log.* **xfs.log_tail.***	XFS ファイルシステムを介したログバッファーの書き込み数のカウンターには、ディスクに書き込まれたブロックの数が含まれます。また、ログフラッシュおよびピニングの数のメトリックです。
xfs.xstrat.*	XFS フラッシュデーモンによりフラッシュされたファイルデータのバイト数と、ディスク上の連続および非連続の領域にフラッシュされたバッファーの数のカウンター。
xfs.attr.*	すべての XFS ファイルシステムでの属性の取得、設定、削除、およびリスト表示の操作数のカウント。
xfs.quota.*	XFS ファイルシステムでのクォータ操作のメトリック。これには、クォータ回収、クォータキャッシュミス、キャッシュヒット、およびクォータデータの回収の数に関するカウンターが含まれます。
xfs.buffer.*	XFS バッファーオブジェクトに関するメトリックの範囲。カウンターには、ページ検索時に要求されたバッファーコールの数、成功したバッファーロック、待機バッファーロック、失敗したときのロック、失敗したときの再試行、バッファーヒットが含まれます。

xfs.btree.*	XFS btree の操作に関するメトリック。
xfs.control.reset	XFS 統計のメトリックカウンターをリセットするのに使用される設定メトリック。コントロールメトリックは、pmstore ツールを使用して切り替えられます。

9.5. XFS のデバイスごとの PCP メトリックグループ

以下の表は、XFS で利用可能なデバイスごとの PCP メトリックグループについて説明しています。

表9.2 XFS のデバイスごとの PCP メトリックグループ

メトリックグループ	提供されたメトリック
xfs.perdev.*	読み書き操作の数、読み書きバイト数を含む一般的な XFS メトリック。inode がフラッシュされた回数、クラッシュした回数、クラスター化に失敗した数に関するカウンターを併用。
xfs.perdev.allocs.* **xfs.perdev.alloc_btree.***	ファイルシステムのオブジェクトの割り当てに関するメトリックの範囲。これには、エクステントおよびブロックの作成/解放の数が含まれます。割り当てツリーの検索と、拡張レコードの作成と btree からの削除との比較。
xfs.perdev.block_map.* **xfs.perdev.bmap_btree.***	メトリックには、ブロックマップの読み取り/書き込みとブロックの削除の数、挿入、削除、および検索のためのエクステントリスト操作が含まれます。また、ブロックマップからの比較、検索、挿入、および削除に関する操作カウンター。
xfs.perdev.dir_ops.*	作成、エントリー削除、getdent の操作の数に対する XFS ファイルシステムのディレクトリー操作のカウンター。
xfs.perdev.transactions.*	メタデータトランザクションの数のカウンター。これには、空のトランザクションの数と、同期および非同期のトランザクションの数のカウントが含まれます。
xfs.perdev.inode_ops.*	オペレーティングシステムが、複数の結果で inode キャッシュの XFS inode を検索する回数のカウンター。このカウントキャッシュのヒット数、キャッシュミスなど。
xfs.perdev.log.* **xfs.perdev.log_tail.***	XFS ファイルシステムを介したログバッファーの書き込み数のカウンターには、ディスクに書き込まれたブロックの数が含まれます。また、ログフラッシュおよびピニングの数のメトリックです。

xfs.perdev.xstrat.*	XFS フラッシュデーモンによりフラッシュされたファイルデータのバイト数と、ディスク上の連続および非連続の領域にフラッシュされたバッファーの数のカウンター。
xfs.perdev.attr.*	すべての XFS ファイルシステムでの属性の取得、設定、削除、およびリスト表示の操作数のカウント。
xfs.perdev.quota.*	XFS ファイルシステムでのクォータ操作のメトリック。これには、クォータ回収、クォータキャッシュミス、キャッシュヒット、およびクォータデータの回収の数に関するカウンターが含まれます。
xfs.perdev.buffer.*	XFS バッファーオブジェクトに関するメトリックの範囲。カウンターには、ページ検索時に要求されたバッファーコールの数、成功したバッファーロック、待機バッファーロック、失敗したときのロック、失敗したときの再試行、バッファーヒットが含まれます。
xfs.perdev.btree.*	XFS btree の操作に関するメトリック。

第10章 PCP メトリックのグラフィカル表示の設定

pcp、**grafana**、**pcp redis**、**pcp bpftrace**、および **pcp vector** を組み合わせて使用すると、ライブデータまたは Performance Co-Pilot (PCP) によって収集されたデータをグラフィカルに表示できます。

10.1. PCP の PCP-ZEROCONF での設定

この手順では、**pcp-zeroconf** パッケージでシステムに PCP を設定する方法を説明します。**pcp-zeroconf** パッケージがインストールされると、システムはメトリックのデフォルトセットをアーカイブファイルに記録します。

手順

- **pcp-zeroconf** パッケージをインストールします。

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

検証手順

- **pmlogger** サービスがアクティブであることを確認し、メトリックのアーカイブを開始します。

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

関連情報

- **pmlogger** の man ページ
- Performance Co-Pilot によるパフォーマンスの監視

10.2. GRAFANA-SERVER の設定

Grafana は、ブラウザーからアクセスできるグラフを生成します。**grafana-server** は、Grafana ダッシュボードのバックエンドサーバーです。これは、デフォルトですべてのインターフェイスでリッスンし、Web ブラウザーからアクセスする Web サービスを提供します。**grafana-pcp** プラグインは、バックエンドの **pmproxy** プロトコルと対話します。

この手順では、**grafana-server** を設定する方法を説明します。

前提条件

- PCP が設定されている。詳細は PCP の pcp-zeroconf での設定 を参照してください。

手順

1. 以下のパッケージをインストールします。

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

2. 以下のサービスを再起動して有効にします。

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

3. Grafana サービスへのネットワークトラフィック用にサーバーのファイアウォールを開きます。

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

# firewall-cmd --reload
success
```

検証手順

- **grafana-server** がリッスンし、要求に応答していることを確認します。

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

- **grafana-pcp** プラグインがインストールされていることを確認します。

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

performancecopilot-pcp-app @ 3.1.0
```

関連情報

- **pmproxy(1)** および **grafana-server** の man ページ

10.3. GRAFANA WEB UI へのアクセス

この手順では、Grafana Web インターフェイスにアクセスする方法を説明します。

Grafana Web インターフェイスを使用すると、以下が可能になります。

- PCP Redis、PCP bpftrace、および PCP Vector データソースを追加します。
- ダッシュボードの作成
- 有用なメトリックの概要の表示
- PCP Redis でのアラートの作成

前提条件

1. PCP が設定されている。詳細は、PCP の pcp-zeroconf での設定 を参照してください。
2. **grafana-server** が設定されている。詳細は、grafana-server の設定 を参照してください。

手順

1. クライアントシステムで **http://192.0.2.0**:3000 リンクを使用してブラウザーを開き、ポート **3000** の **grafana-server** にアクセスします。

192.0.2.0 をマシン IP に置き換えます。

2. 最初のログインでは、**Email or username** と **Password** の両方のフィールドに **admin** と入力します。
   Grafana は、**新しいパスワード** を設定してセキュアなアカウントを作成するようにプロンプトを表示します。後で設定する場合は、**Skip** をクリックします。

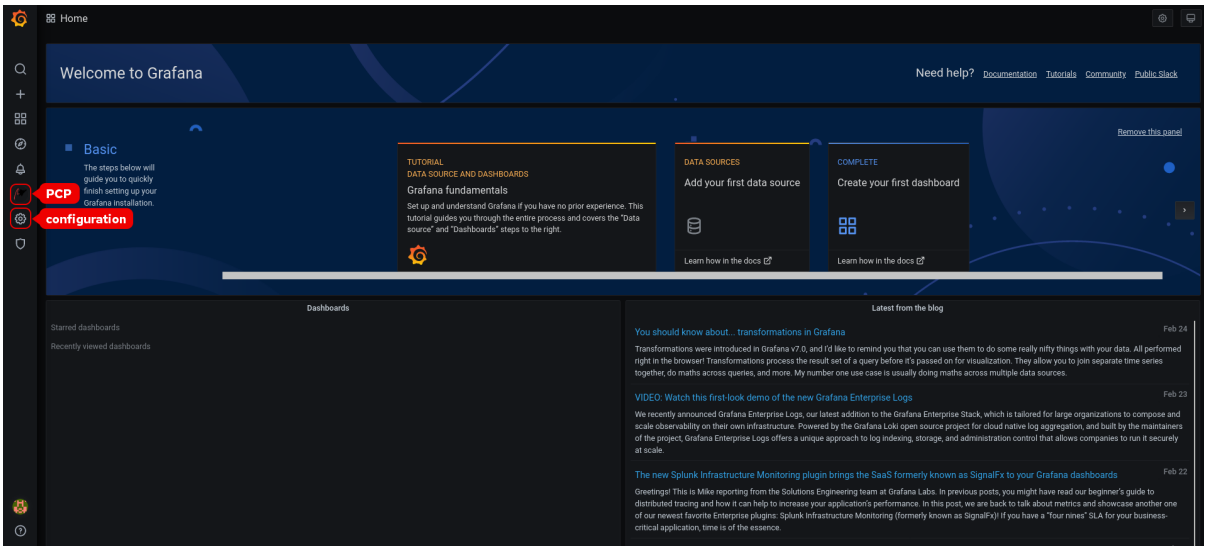
3. メニューで  **Configuration** アイコンにカーソルを合わせてから、**Plugins** をクリックします。

4. **プラグイン** タブで、**Search by name or type**テキストボックスに performance co-pilot と入力し、**Performance Co-Pilot** (PCP) プラグインをクリックします。

5. **Plugins / Performance Co-Pilot** ペインで、**Enable** をクリックします。

6. Grafana  アイコンをクリックします。Grafana **Home** ページが表示されます。

図10.1 Home Dashboard



注記

画面上部の隅には同様の  アイコンがありますが、これは一般的な **ダッシュボード設定** を制御します。

7. Grafana **Home** ページで、**Add your first data source**をクリックして PCP Redis、PCP bpftrace、および PCP Vector データソースを追加します。データソースの追加に関する詳細は、以下を参照してください。

   - pcp redis データソースを追加するには、デフォルトのダッシュボードを表示し、パネルとアラートルールを作成します。詳細は、PCP Redis データソースでのパネルおよびアラートの作成 を参照してください。

   - pcp bpftrace データソースを追加してデフォルトのダッシュボードを表示するには、PCP bpftrace システム分析ダッシュボードの表示 を参照してください。

- pcp vector データソースを追加するには、デフォルトのダッシュボードを表示します。Vector Checklist を表示するには、PCP Vector Checklist の表示 を参照してください。

8. オプション: メニューで、**admin** プロファイル  アイコンにカーソルを合わせ、**Edit Profile**、**Change Password** を含む **Preferences** を変更するか、**Sign out** します。

関連情報

- **grafana-cli** および **grafana-server** の man ページ

10.4. PCP REDIS の設定

PCP Redis データソースを使用して以下を行います。

- データアーカイブの表示
- pmseries 言語を使用したクエリー時系列
- 複数のホストにまたがるデータの分析

前提条件

1. PCP が設定されている。詳細は PCP の pcp-zeroconf での設定 を参照してください。
2. **grafana-server** が設定されている。詳細は、grafana-server の設定 を参照してください。
3. メール転送エージェント (**sendmail** または **postfix** など) がインストールされ、設定されている。

手順

1. **redis** パッケージをインストールします。

```
# yum module install redis:6
```

注記

Red Hat Enterprise Linux 8.4 以降、Redis 6 がサポートされていますが、**yum update** コマンドは Redis 5 を Redis 6 に更新しません。Redis 5 から Redis 6 に更新するには、次を実行します。

yum モジュールの Redis への切り替え:6

2. 以下のサービスを開始して有効にします。

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

3. **grafana-server** を再起動します。

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

検証手順

- **pmproxy** および **redis** が動作していることを確認します。

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

  redis パッケージがインストールされていない場合は、このコマンドはデータを返しません。

関連情報

- **pmseries(1)** の man ページ

10.5. PCP REDIS データソースでのパネルおよびアラートの作成

PCP Redis データソースを追加した後に、ダッシュボードに有用なメトリックの概要を表示し、負荷グラフを視覚化するためのクエリーを追加して、システムに問題が発生した場合にその問題を表示する上で役立つアラートを作成できます。

前提条件

1. PCP Redis が設定されている。詳細は PCP Redis の設定 を参照してください。
2. **grafana-server** にアクセスできる。詳細は、Grafana Web UI へのアクセス を 参照してください。

手順

1. Grafana Web UI にログインします。
2. Grafana **Home** ページで、**Add your first data source**をクリックします。
3. **Add data source** ペインで、**Filter by name or type**のテキストボックスに redis と入力してから **PCP Redis** をクリックします。
4. **Data Sources / PCP Redis**ペインで、以下を実行します。
   a. **URL** フィールドに **http://localhost:44322** を追加し、**Save & Test** をクリックします。
   b. **Dashboards tab** → **Import** → **PCP Redis: Host Overview** をクリックして、有用なメトリックの概要を含むダッシュボードを表示します。

図10.2 PCP Redis: ホストの概要

5. 新しいパネルを追加します。

a. メニューで、 Create icon→ Dashboard → Add new panel iconの順にマウスを合わせ、パネルを追加します。

b. **Query** タブで、選択した **default** オプションではなく、クエリーリストから **PCP Redis** を選択し、**A** のテキストフィールドで **kernel.all.load** などのメトリックを入力して、カーネル負荷グラフを可視化します。

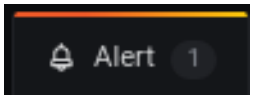
c. 必要に応じて、**Panel title** と **Description** を追加し、**Settings** から他のオプションを更新します。

d. **Save** をクリックして変更を適用し、ダッシュボードを保存します。**Dashboard name** を追加します。

e. **Apply** をクリックして変更を適用し、ダッシュボードに戻ります。

図10.3 PCP Redis クエリーパネル

6. アラートルールを作成します。

a. **PCP Redis query panel** で 

Alert をクリックしてから、**Create Alert** をクリックします。

b. **Rule** の **Name**、**Evaluate query**、および **For** フィールドを編集して、アラートの **Conditions** を指定します。

c. **Save** をクリックして変更を適用し、ダッシュボードを保存します。**Apply** をクリックして変更を適用し、ダッシュボードに戻ります。

図10.4 PCP Redis パネルでのアラートの作成

d. 必要に応じて、同じパネルでスクロールダウンし、**Delete** アイコンをクリックして、作成したルールを削除します。

e. オプション: メニューで  **Alerting** アイコンをクリックし、作成されたアラートルールをさまざまなアラートステータスで表示したり、アラートルールを編集したり、**Alert Rules** タブから既存のルールを一時停止したりします。
作成したアラートルールの通知チャネルを追加して Grafana からアラート通知を受信するには、アラートの通知チャネルの追加 を参照してください。

10.6. アラートの通知チャネルの追加

通知チャネルを追加すると、アラートルールの条件が満たされ、システムにさらなる監視が必要になると、Grafana からアラート通知を受け取ることができます。

サポートされている通知機能のリストからいずれかのタイプを選択すると、これらのアラートを受け取ることができます。通知機能には、**DingDing**、**Discord**、**Email**、**Google Hangouts Chat**、**HipChat**、**Kafka REST Proxy**、**LINE**、**Microsoft Teams**、**OpsGenie**、**PagerDuty**、**Prometheus Alertmanager**、**Pushover**、**Sensu**、**Slack**、**Telegram**、**Threema Gateway**、**VictorOps**、および **webhook** が含まれます。

前提条件

1. **grafana-server** にアクセスできる。詳細は、Grafana Web UI へのアクセス を 参照してください。

2. アラートルールが作成されている。詳細は、PCP Redis データソースでのパネルおよびアラートの作成 を参照してください。

3. SMTP を設定し、**grafana/grafana.ini** ファイルに有効な送信者のメールアドレスを追加します。

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

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

abc@gmail.com を有効なメールアドレスに置き換えます。

4. **grafana-server** を再起動します。

```
# systemctl restart grafana-server.service
```

手順

1. メニューで  **Alerting icon → click Notification channels → Add channel** の順でカーソルを合わせます。

2. Add notification channel details ペインで、以下を実行します。

   a. **Name** テキストボックスに、名前を入力します。

   b. 通信 **Type** (例: Email) を選択し、メールアドレスを入力します。区切り文字 **;** を使用して複数のメールアドレスを追加できます。

   c. オプション: **Optional Email settings** および **Notification settings** を設定します。

3. **Save** をクリックします。

4. アラートルールで通知チャネルを選択します。

   a. メニューで  **Alerting** アイコンにマウスを合わせ、**Alert rules** をクリックします。

   b. **Alert Rules** タブで、作成されたアラートルールをクリックします。

   c. **Notifications** タブで **Send to** オプションから通知チャネル名を選択し、アラートメッセージを追加します。

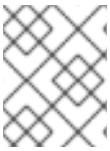
   d. **Apply** をクリックします。

関連情報

- Upstream Grafana documentation for alert notifications

10.7. PCP コンポーネント間の認証の設定

Simple Authentication Security Layer (SASL) フレームワークを介して PCP によってサポートされる **scram-sha-256** 認証メカニズムを使用して認証を設定できます。



注記

Red Hat Enterprise Linux 8.3 以降から、PCP は **scram-sha-256** 認証メカニズムに対応します。

手順

1. **scram-sha-256** 認証メカニズムの **sasl** フレームワークをインストールします。

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

2. **pmcd.conf** ファイルに、サポートされている認証メカニズムとユーザーデータベースのパスを指定します。

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

mech_list: scram-sha-256

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

3. 新しいユーザーを作成します。

```
# useradd -r metrics
```

metrics をユーザー名に置き換えます。

4. 作成したユーザーをユーザーデータベースに追加します。

```
# saslpasswd2 -a pmcd metrics

Password:
Again (for verification):
```

作成したユーザーを追加するには、**メトリック** アカウントのパスワードを入力する必要があります。

5. ユーザーデータベースのパーミッションを設定します。

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

6. **pmcd** サービスを再起動します。

```
# systemctl restart pmcd
```

検証手順

- **sasl** 設定を確認します。

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

関連情報

- **saslauthd(8)**、**pminfo(1)**、および **sha256** の man ページ
- How can I setup authentication between PCP components, like PMDAs and pmcd in RHEL 8.2?

10.8. PCP BPFTRACE のインストール

PCP **bpftrace** エージェントをインストールして、システムをイントロスペクトし、カーネルおよびユーザー空間トレースポイントからメトリックを収集します。

bpftrace エージェントは bpftrace スクリプトを使用してメトリックを収集します。**bpftrace** スクリプトは、強化された Berkeley Packet Filter (**eBPF**) を使用します。

この手順では、**pcp bpftrace** をインストールする方法を説明します。

前提条件

1. PCP が設定されている。詳細は PCP の pcp-zeroconf での設定 を参照してください。
2. **grafana-server** が設定されている。詳細は、grafana-server の設定 を参照してください。
3. **scram-sha-256** 認証メカニズムが設定されている。詳細は、PCP コンポーネント間の認証の設定 を参照してください。

手順

1. **pcp-pmda-bpftrace** パッケージをインストールします。

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

2. **bpftrace.conf** ファイルを編集し、{setting-up-authentication-between-pcp-components} で作成したユーザーを追加します。

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

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

metrics をユーザー名に置き換えます。

3. **bpftrace** PMDA をインストールします。

```
# 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 がインストールされたため、ユーザーの認証後にのみ使用できるようになりました。詳細は PCP bpftrace システム分析ダッシュボードの表示 を参照してください。

関連情報

- **pmdabpftrace(1)** および **bpftrace** の man ページ

10.9. PCP BPFTRACE システム分析ダッシュボードの表示

PCP bpftrace データソースを使用すると、**pmlogger** またはアーカイブからの通常のデータとして利用できないソースからのライブデータにアクセスできます。

PCP bpftrace データソースでは、ダッシュボードに有用なメトリックの概要を表示できます。

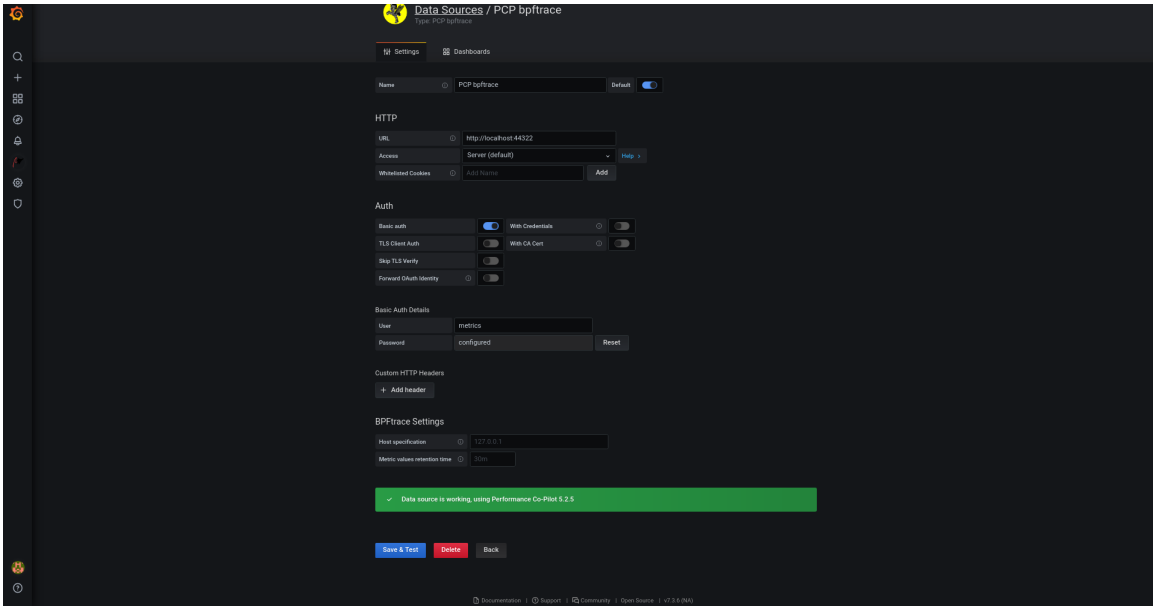
前提条件

1. PCP bpftrace がインストールされている。詳細は、PCP bpftrace のインストール を参照してください。
2. **grafana-server** にアクセスできる。詳細は、Grafana Web UI へのアクセス を 参照してください。

手順

1. Grafana Web UI にログインします。
2. Grafana **Home** ページで、**Add your first data source**をクリックします。
3. **Add data source** ペインで、**Filter by name or type**テキストボックスに bpftrace と入力して、**PCP bpftrace** をクリックします。
4. **Data Sources / PCP bpftrace**ペインで、以下を実行します。
   a. **URL** フィールドに **http://localhost:44322** を追加します。
   b. **Basic Auth** オプションを切り替えて、作成されたユーザーの認証情報を、**User** フィールドおよび **Password** フィールドに追加します。
   c. **Save & Test** をクリックします。

図10.5 データソースへの PCP bpftrace の追加



d. **Dashboards tab** → **Import** → **PCP bpftrace: System Analysis** をクリックし、有用なメトリックの概要を含むダッシュボードを表示します。

図10.6 PCP bpftrace: システム分析

10.10. PCP VECTOR のインストール

この手順では、**pcp vector** をインストールする方法を説明します。

前提条件

1. PCP が設定されている。詳細は PCP の pcp-zeroconf での設定 を参照してください。
2. **grafana-server** が設定されている。詳細は、grafana-server の設定 を参照してください。

手順

1. **pcp-pmda-bcc** パッケージをインストールします。

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

2. **bcc** PMDA をインストールします。

```
# 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
```

関連情報

- **pmdabcc(1)** の man ページ

10.11. PCP VECTOR CHECKLIST の表示

PCP Vector データソースはライブメトリックを表示し、**pcp** メトリックを使用します。各ホストのデータを分析します。

PCP Vector データソースを追加した後に、ダッシュボードに有用なメトリックの概要を表示し、チェックリストで関連するトラブルシューティングまたは参照リンクを表示できます。

前提条件

1. PCP Vector がインストールされている。詳細は PCP Vector のインストール を参照してください。
2. **grafana-server** にアクセスできる。詳細は、Grafana Web UI へのアクセス を 参照してください。

手順

1. Grafana Web UI にログインします。
2. Grafana **Home** ページで、**Add your first data source**をクリックします。
3. **Add data source** ペインで、**Filter by name or type**テキストボックスに vector と入力してから **PCP Vector** をクリックします。
4. **Data Sources / PCP Vector**ペインで、以下を実行します。
   a. **URL** フィールドに **http://localhost:44322** を追加し、**Save & Test** をクリックします。
   b. **Dashboards tab** → **Import** → **PCP Vector: Host Overview** をクリックして、有用なメトリックの概要を含むダッシュボードを表示します。

   図10.7 PCP Vector: ホストの概要

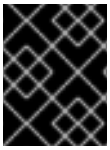
5. メニューで  **Performance Co-Pilot** プラグインにマウスを合わせ、**PCP Vector Checklist** をクリックします。

PCP チェックリストで、  ヘルプまたは  警告アイコをクリックし、関連するトラブルシューティングまたは参照リンクを表示します。

図10.8 Performance Co-Pilot / PCP Vector Checklist

10.12. GRAFANA のヒートマップの使用

Grafana のヒートマップを使用すると、経時的なデータのヒストグラムを表示し、データの傾向とパターンを特定し、時間の経過に伴う変化を確認できます。ヒートマップ内の各列は単一のヒストグラムを表します。それぞれのセルの色は、そのヒストグラム内における特定の値の観測密度を表します。



重要

このワークフローは、RHEL9 の Grafana バージョン 9.0.9 以降のヒートマップ専用です。

前提条件

- PCP Redis が設定されている。詳細は、PCP Redis の設定 を参照してください。
- **grafana-server** にアクセスできる。詳細は、Grafana Web UI へのアクセス を参照してください。
- PCP Redis データソースが設定されている。詳細は、PCP Redis データソースでのパネルおよびアラートの作成 を参照してください。

手順

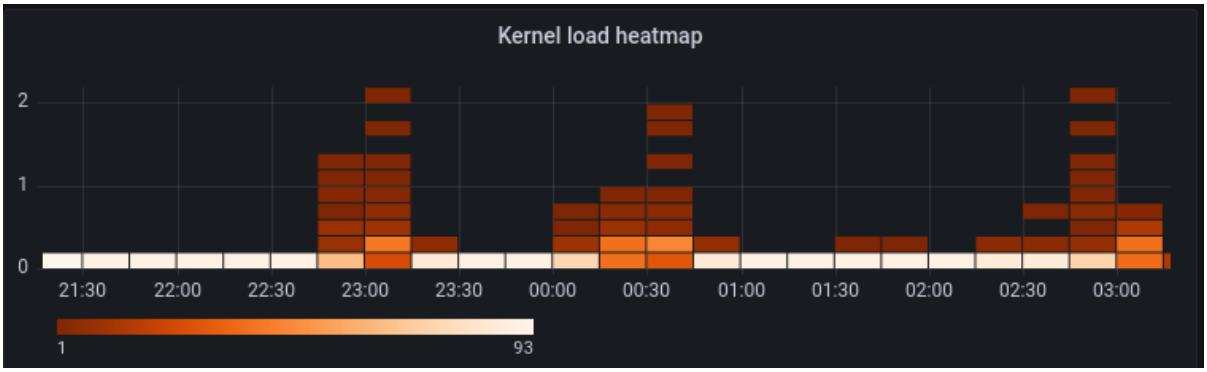
1. **Dashboards tab** タブにカーソルを合わせて、**+ New dashboard** をクリックします。
2. **Add panel** メニューで、**Add a new panel** をクリックします。
3. **Query** タブで以下を実行します。
   a. 選択中のデフォルトオプションの代わりに、クエリーリストから **PCP Redis** を選択します。
   b. **A** のテキストフィールドに、カーネル負荷グラフを可視化するためのメトリクス (例: **kernel.all.load**) を入力します。

4. 可視化ドロップダウンメニュー (デフォルトで **Time series** に設定されています) をクリックし、**Heatmap** をクリックします。

5. オプション: **Panel Options** ドロップダウンメニューで、**Panel Title** と **Description** を追加します。

6. **Heatmap** ドロップダウンメニューの **Calculate from data** 設定で、**Yes** をクリックします。

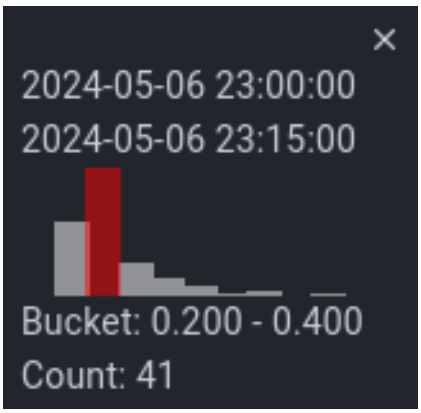
   ヒートマップ



7. オプション: **Colors** ドロップダウンメニューで、**Scheme** をデフォルトの **Orange** から変更し、ステップ数 (色の濃淡) を選択します。

8. オプション: **Tooltip** ドロップダウンメニューの **Show histogram (Y Axis)**設定で、トグルをクリックすると、ヒートマップ内のセルの上にカーソルを置いたときに、特定のヒストグラム内でのセルの位置が表示されます。以下に例を示します。

   Show histogram (Y Axis) のセルの表示



10.13. GRAFANA に関する問題のトラブルシューティング

Grafana にデータが表示されない、ダッシュボードが黒くなる、または同様の問題など、Grafana の問題のトラブルシューティングが必要になる場合があります。

手順

- 以下のコマンドを実行して、**pmlogger** サービスが起動していることを確認します。

```
$ systemctl status pmlogger
```

- 以下のコマンドを実行して、ディスクにファイルが作成または変更されているかどうかを確認します。

```
$ 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
[..]
```

- 以下のコマンドを実行して、**pmproxy** サービスが動作していることを確認します。

```
$ systemctl status pmproxy
```

- **pmproxy** が動作していること、時系列サポートが有効になっていること、Redis への接続が確立されていることを、**/var/log/pcp/pmproxy/pmproxy.log** ファイルを見て、以下のテキストが含まれていることで確認してください。

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

ここで、**1716**は pmproxy の PID であり、 **pmproxy** を起動するたびに異なる値になります。

- 以下のコマンドを実行して、Redis データベースにキーが含まれているかどうかを確認します。

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

- 以下のコマンドを実行して、PCP メトリックが Redis データベースに存在し、**pmproxy** がアクセスできるかどうかを確認します。

```
$ 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
```

- 以下のコマンドを実行して、Grafana のログにエラーがあるかどうかを確認します。

```
$ 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>
[...]
```

第11章 WEB コンソールを使用したシステムパフォーマンスの最適化

以下では、RHEL Web コンソールでパフォーマンスプロファイルを設定し、選択したタスクに対してシステムのパフォーマンスを最適化する方法を説明します。

11.1. WEB コンソールでのパフォーマンスチューニングオプション

Red Hat Enterprise Linux 8 には、以下のタスクに対してシステムを最適化する複数のパフォーマンスプロファイルが同梱されています。

- デスクトップを使用するシステム
- スループットパフォーマンス
- レイテンシーパフォーマンス
- ネットワークパフォーマンス
- 電力の低消費
- 仮想マシン

TuneD サービスは、選択したプロファイルに一致するようにシステムオプションを最適化します。

Web コンソールでは、システムが使用するパフォーマンスプロファイルを設定できます。

関連情報

- TuneD を使い始める

11.2. WEB コンソールでのパフォーマンスプロファイルの設定

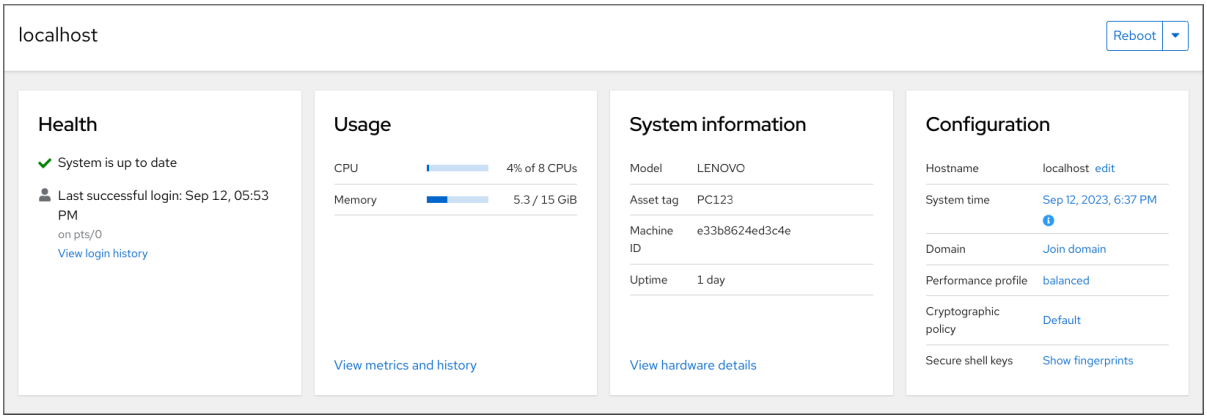
実行するタスクに応じて、Web コンソールを使用して適切なパフォーマンスプロファイルを設定することでシステムパフォーマンスを最適化できます。

前提条件

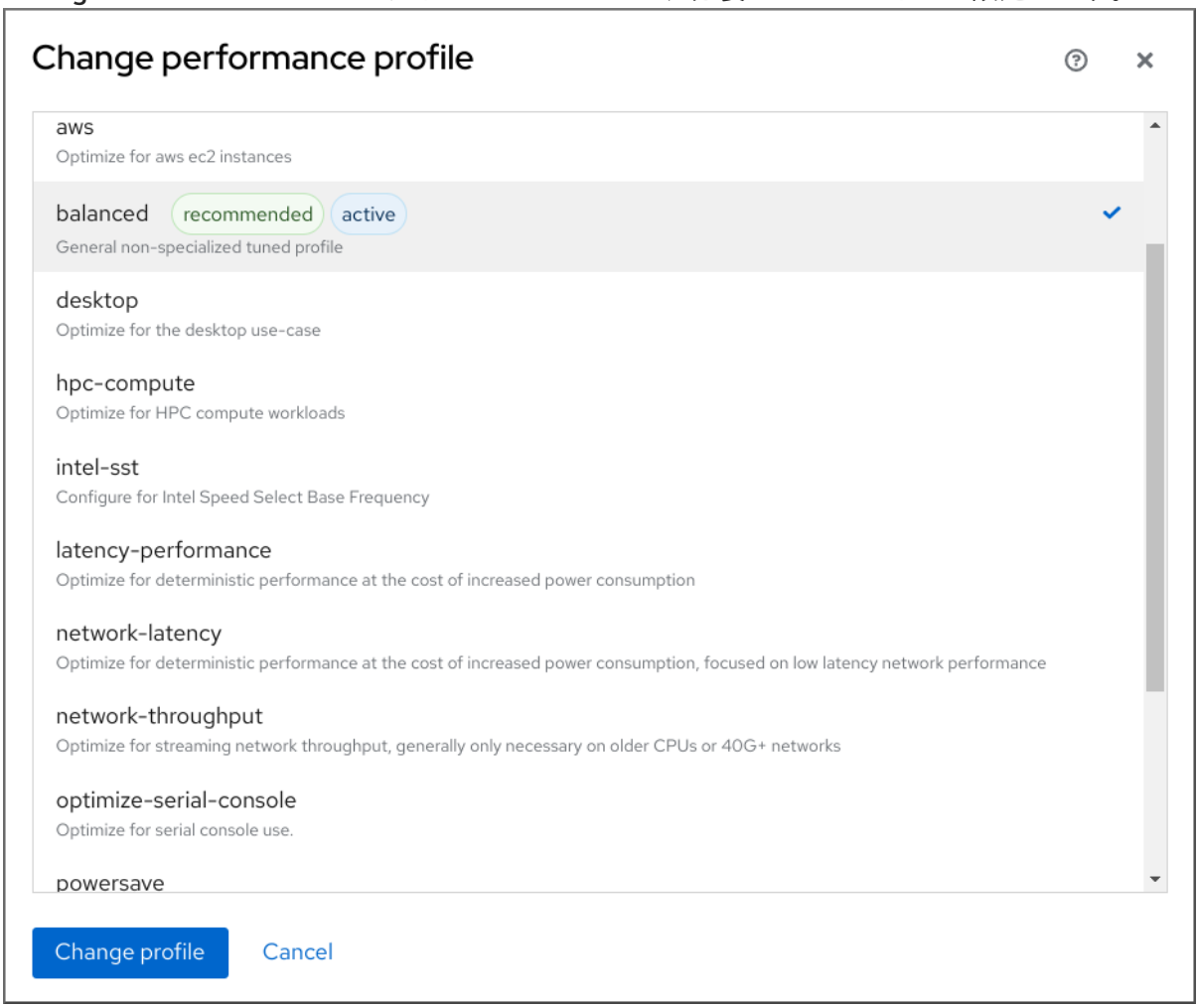
- Web コンソールをインストールし、アクセスできることを確認します。詳細は、Web コンソールのインストール を参照してください。

手順

1. 8 の Web コンソールにログインします。詳細は、Web コンソールへのログイン を参照してください。
2. **Overview** をクリックします。
3. **Configuration** セクションで、現在のパフォーマンスプロファイルをクリックします。



4. **Change Performance Profile** ダイアログボックスで、必要なプロファイルを設定します。



5. **Change Profile** をクリックします。

検証手順

- **Overview** タブの **Configuration** セクションに、選択したパフォーマンスプロファイルが表示されます。

11.3. WEB コンソールを使用したローカルシステムのパフォーマンスの監視

Red Hat Enterprise Linux の Web コンソールは、トラブルシューティングに Utilization Saturation and Errors (USE) メソッドを使用します。新しいパフォーマンスメトリックページには、データの履歴ビューが時系列に整理されており、最新のデータが上部に表示されます。

Metrics and history ページでは、イベント、エラー、リソースの使用率と飽和状態のグラフィカル表示を表示できます。

前提条件

- RHEL 8 Web コンソールをインストールし、アクセスできる。詳細は、Web コンソールのインストール を参照してください。
- パフォーマンスメトリクスの収集を可能にする **cockpit-pcp** パッケージがインストールされている。
  a. Web コンソールインターフェイスからパッケージをインストールするには、以下を行います。
    i. Web コンソールに管理者権限でログインする。詳細は、Web コンソールへのログイン を参照してください。
    ii. **Overview** ページで、**View metrics and history** をクリックします。
    iii. **cockpit-pcp のインストール** ボタンをクリックします。
    iv. **ソフトウェアのインストール** ダイアログウィンドウで、**Install** をクリックします。
  b. コマンドラインインターフェイスからパッケージをインストールするには、次を使用します。

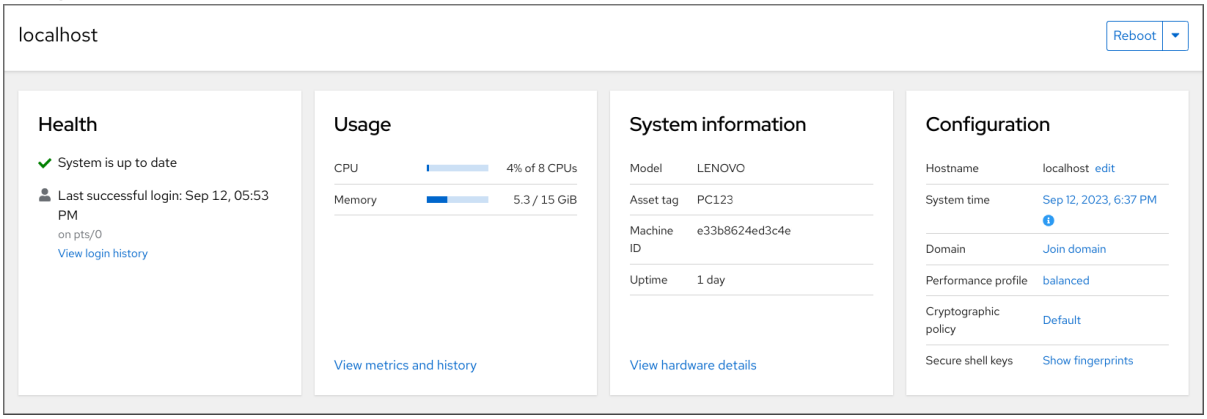
```
# yum install cockpit-pcp
```

- Performance Co-Pilot (PCP) サービスが有効になっている。

```
# systemctl enable --now pmlogger.service pmproxy.service
```

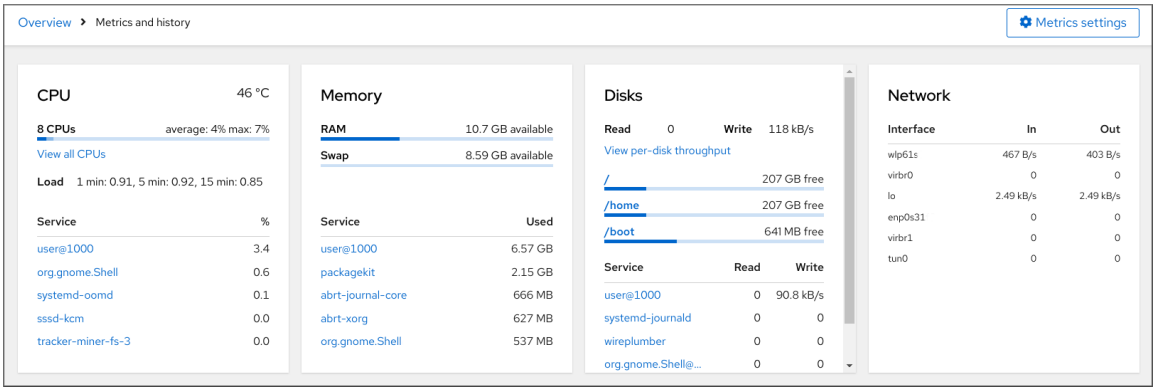
手順

1. 8 の Web コンソールにログインします。詳細は、Web コンソールへのログイン を参照してください。
2. **Overview** をクリックします。
3. **Usage** セクションで、**View metrics and history** をクリックします。

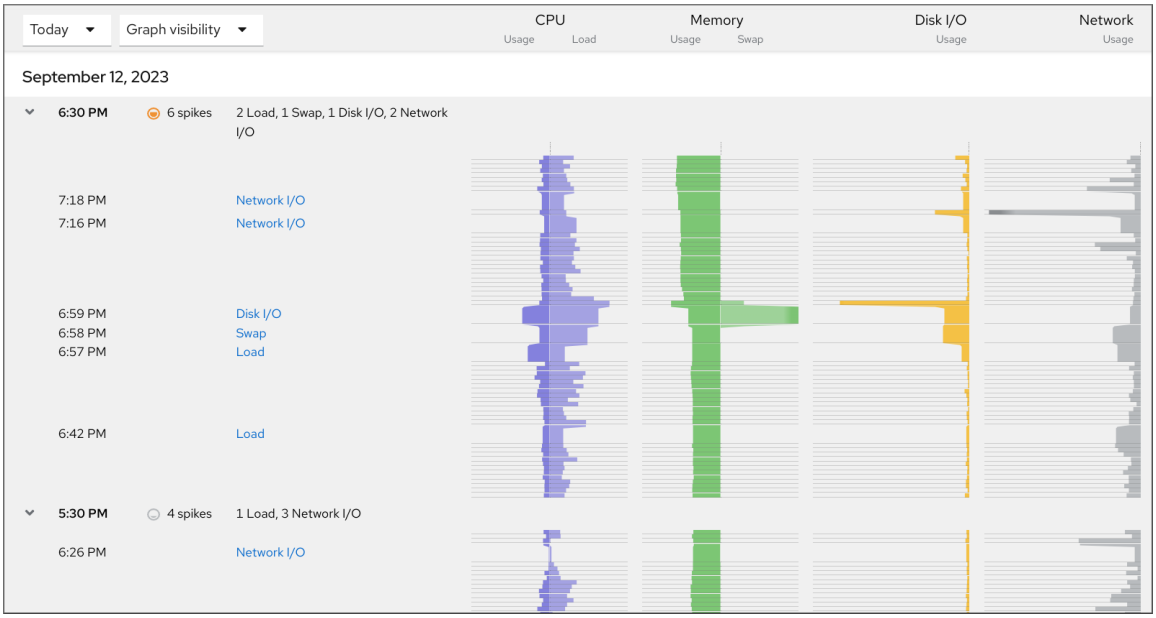


Metrics and history セクションが開きます。

- 現在のシステム設定と使用状況:



- ユーザー指定の時間間隔におけるグラフィック形式のパフォーマンスメトリクス:



11.4. WEB コンソールと GRAFANA を使用して複数のシステムのパフォーマンスを監視する

Grafana を使用すると、一度に複数のシステムからデータを収集し、収集した Performance Co-Pilot (PCP) メトリックのグラフィカル表現を確認できます。Web コンソールインターフェイスで、複数のシステムのパフォーマンスメトリックの監視およびエクスポートを設定できます。

前提条件

- Web コンソールがインストールされており、アクセス可能である。詳細は、Web コンソールのインストールおよび有効化 を参照してください。
- **cockpit-pcp** パッケージをインストールします。

1. Web コンソールインターフェイスから:

   a. Web コンソールに管理者権限でログインする。詳細は、Web コンソールへのログイン を参照してください。

   b. **概要** ページで、**詳細と履歴を表示** をクリックします。

   c. **cockpit-pcp のインストール** ボタンをクリックします。

   d. **ソフトウェアのインストール** ダイアログウィンドウで、**Install** をクリックします。

e. ログアウトしてから再度ログインして、メトリクスの履歴を表示します。

2. コマンドラインインターフェイスからパッケージをインストールするには、次を使用します。

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

- PCP サービスを有効にします。

```
# systemctl enable --now pmlogger.service pmproxy.service
```

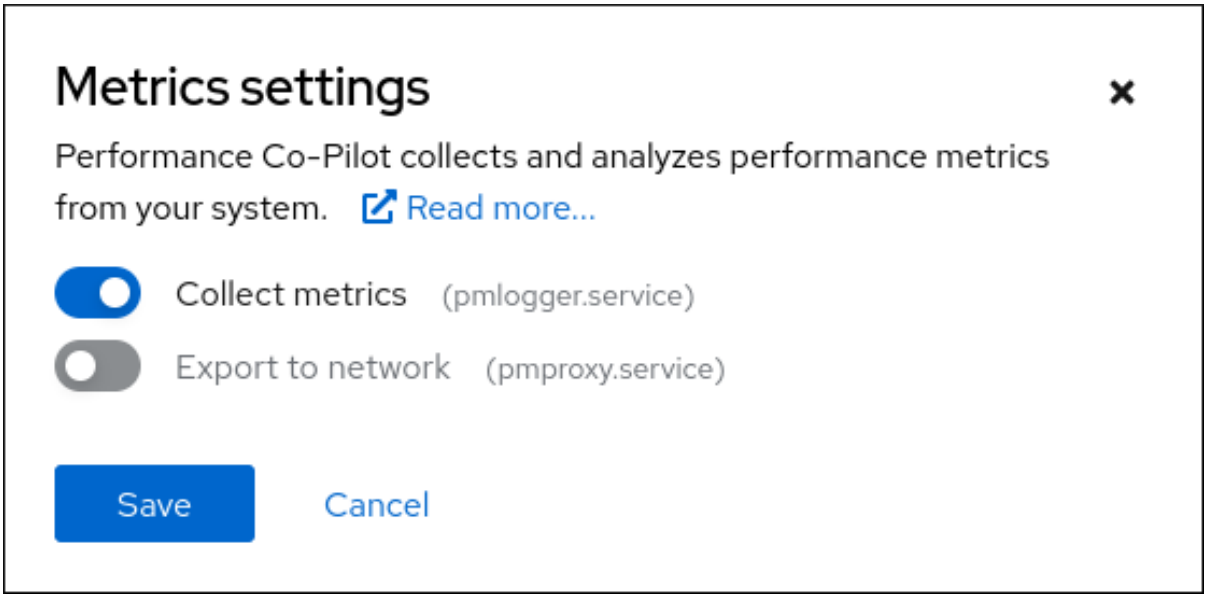
- Grafana ダッシュボードをセットアップします。詳細は、grafana-server の設定 を参照してください。
- **redis** パッケージをインストールします。

```
# yum install redis
```

または、手順の後半で Web コンソールインターフェイスからパッケージをインストールすることもできます。

手順

1. **Overview** ページで、**Usage** テーブルの **View metrics and history** をクリックします。
2. **Metrics settings** ボタンをクリックします。
3. **Export to network** スライダーをアクティブな位置に移動します。



redis パッケージがインストールされていない場合は、Web コンソールでインストールするように求められます。

4. **pmproxy** サービスを開くには、ドロップダウンリストからゾーンを選択し、**Add pmproxy** ボタンをクリックします。
5. **Save** をクリックします。

検証

1. **Networking** をクリックします。
2. **Firewall** テーブルで、**Edit rules and zones** ボタンをクリックします。
3. 選択したゾーンで **pmproxy** を検索します。

重要

監視するすべてのシステムでこの手順を繰り返します。

関連情報

- PCP メトリックのグラフィカル表示の設定

第12章 ディスクスケジューラーの設定

ディスクスケジューラーは、ストレージデバイスに送信された I/O 要求を順序付けます。

スケジューラーは以下の複数の方法で設定できます。

- Setting the disk scheduler using TuneD の説明に従って、**TuneD** を使用してスケジューラーを設定します。
- udev ルールを使用したディスクスケジューラーの設定 で説明されているように、**udev** を使用してスケジューラーを設定します。
- 特定ディスクに任意のスケジューラーを一時的に設定 で説明されているように、実行中のシステムのスケジューラーを一時的に変更します。

注記

Red Hat Enterprise Linux 8 では、ブロックデバイスはマルチキュースケジューリングのみに対応します。これにより、ブロックレイヤーのパフォーマンスを高速ソリッドステートドライブ (SSD) およびマルチコアシステムで適切に拡張できます。

Red Hat Enterprise Linux 7 以前のバージョンで利用できた従来のシングルキュースケジューラーが削除されました。

12.1. 利用可能なディスクスケジューラー

Red Hat Enterprise Linux 8 では、以下のマルチキューディスクスケジューラーに対応しています。

none

FIFO (First-in First-out) スケジューリングアルゴリズムを実装します。これにより、汎用のブロック層で単純な last-hit キャッシュを介して要求がマージされます。

mq-deadline

これにより、要求がスケジューラーに到達した時点からの要求のレイテンシーが保証されます。**mq-deadline** スケジューラーは、キュー待ちの I/O リクエストを読み取りバッチまたは書き込みバッチに分類します。そして、論理ブロックアドレス (LBA) を増大順に実行するためのスケジュール設定を行います。デフォルトでは、アプリケーションは読み取り I/O 操作でブロックする可能性の方が高いため、読み取りバッチの方が書き込みバッチより優先されます。**mq-deadline** がバッチを処理すると、このプロセスは書き込み動作が待機している長さを確認して、次の読み取りバッチまたは書き込みバッチをスケジュールします。

このスケジューラーはほとんどのユースケースに適していますが、必要に応じて特に書き込み動作より読み取り動作の方が頻繁に起こるユースケースに適しています。

bfq

デスクトップシステムおよび対話式のタスクを対象とします。
bfq スケジューラーは、単一のアプリケーションがすべての帯域幅を使用しないようにします。これにより、ストレージデバイスがアイドル状態であるかのように常に応答できるようになります。デフォルトの設定では、**bfq** は、最大スループットを実現するのではなく、レイテンシーを最小限に抑えることに焦点を合わせています。

bfq は **cfq** コードに基づいています。固定タイムスライスについて、ディスクは各プロセスに付与されることはありませんが、セクター数を測定する **budget** をプロセスに割り当てます。

このスケジューラーは大きなファイルをコピーする際に適しており、この場合、システムが応答しなくなることはありません。

kyber

スケジューラーは、ブロック I/O レイヤーに送信されたすべての I/O 要求のレイテンシーを計算することで、レイテンシーゴールを達成するために自身を調整します。cache-misses の場合、読み込み/同期書き込みリクエストにターゲットレイテンシーを設定できます。
このスケジューラーは、NVMe、SSD などの低レイテンシーデバイスなど、高速なデバイスに適しています。

12.2. 各種ユースケースで異なるディスクスケジューラー

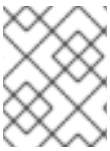
システムが実行するタスクに応じて、分析タスクおよびチューニングタスクの前に、以下のディスクスケジューラーがベースラインとして推奨されます。

表12.1 各種ユースケースのディスクスケジューラー

ユースケース	ディスクスケジューラー
SCSI インターフェイスを備えた従来の HDD	**mq-deadline** または **bfq** を使用します。
高速ストレージで高パフォーマンスの SSD または CPU がバインドされたシステム	特にエンタープライズアプリケーションを実行する場合は **none** を使用します。または **kyber** を使用します。
デスクトップまたはインタラクティブなタスク	**bfq** を使用します。
仮想ゲスト	**mq-deadline** を使用します。マルチキューに対応しているホストバスアダプター (HBA) ドライバーでは、**none** を使用します。

12.3. デフォルトのディスクスケジューラー

ブロックデバイスは、別のスケジューラーを指定しない限り、デフォルトのディスクスケジューラーを使用します。



注記

NVMe (Non-volatile Memory Express) ブロックデバイスの場合、デフォルトのスケジューラーは **none** であり、Red Hat ではこれを変更しないことを推奨します。

カーネルは、デバイスのタイプに基づいてデフォルトのディスクスケジューラーを選択します。自動的に選択されたスケジューラーは、通常、最適な設定です。別のスケジューラーが必要な場合は、Red Hat では、**udev** ルールまたは **TuneD** アプリケーションを使用して設定することを推奨しています。選択したデバイスを一致させ、それらのデバイスのスケジューラーのみを切り替えます。

12.4. アクティブなディスクスケジューラーの決定

この手順では、特定のブロックデバイスで現在アクティブなディスクスケジューラーを確認します。

手順

- **/sys/block/device/queue/scheduler** ファイルの内容を読み取ります。

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

[mq-deadline] kyber bfq none
```

ファイル名の **device** を、**sdc** などのブロックデバイス名に置き換えます。

アクティブなスケジューラーは、角括弧 (**[ ]**) にリスト表示されます。

12.5. TUNED を使用したディスクスケジューラーの設定

この手順では、選択したブロックデバイスに特定のディスクスケジューラーを設定するTuneD プロファイルを作成して有効にします。この設定は、システムを再起動しても持続します。

以下のコマンドと設定で、以下の内容を置き換えます。

- **device** をブロックデバイスの名前に置き換えます (例: **sdf**)。
- **selected-scheduler** を、デバイスに設定するディスクスケジューラーに置き換えます (例: **bfq**)。

前提条件

- **Tuned** サービスがインストールされ、有効になっている。詳細は、TuneD のインストールと有効化 を参照してください。

手順

1. 必要に応じて、プロファイルのベースとなる既存のTuneDプロファイルを選択します。利用可能なプロファイルのリストは、RHEL とともに配布される TuneD プロファイル を参照してください。
   現在アクティブなプロファイルを確認するには、次のコマンドを実行します。

```
$ tuned-adm active
```

2. TuneD プロファイルを保持する新しいディレクトリーを作成します。

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

3. 選択したブロックデバイスのシステム固有の識別子を見つけます。

```
$ 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
```

注記

この例のコマンドは、指定したブロックデバイスに関連付けられた World Wide Name (WWN) またはシリアル番号として識別されるすべての値を返します。WWN を使用することが推奨されますが、WWN は特定のデバイスで常に利用できる訳ではなく、コマンド例で返される値は、**デバイスのシステム固有の ID** として使用することが許容されます。

4. **/etc/tuned/my-profile/tuned.conf** 設定ファイルを作成します。このファイルで、以下のオプションを設定します。

   a. 必要に応じて、既存のプロファイルを追加します。

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

   b. WWN 識別子に一致するデバイスに対して選択したディスクスケジューラーを設定します。

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

   ここでは、以下のようになります。

   - **IDNAME** を、使用されている識別子名に置き換えます (例:**ID_WWN**)。
   - **device system unique id** を、選択した識別子の値に置き換えます (例:**0x5002538d00000000**)。
     devices_udev_regex オプションで複数のデバイスに一致させるには、識別子を括弧で囲み、垂直バーで区切ります。

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

5. プロファイルを有効にします。

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

検証手順

1. TuneD プロファイルがアクティブで、適用されていることを確認します。

   ```
   $ 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** ファイルの内容を読み取ります。

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

[mq-deadline] kyber bfq none
```

ファイル名の **device** を、**sdc** などのブロックデバイス名に置き換えます。

アクティブなスケジューラーは、角括弧 (**[]**) にリスト表示されます。

関連情報

- TuneD プロファイルのカスタマイズ

12.6. UDEV ルールを使用したディスクスケジューラーの設定

この手順では、**udev** ルールを使用して、特定ブロックデバイスに、特定のディスクスケジューラーを設定します。この設定は、システムを再起動しても持続します。

以下のコマンドと設定で、以下の内容を置き換えます。

- **device** をブロックデバイスの名前に置き換えます (例: **sdf**)。
- **selected-scheduler** を、デバイスに設定するディスクスケジューラーに置き換えます (例: **bfq**)。

手順

1. ブロックデバイスのシステム固有の識別子を見つけます。

```
$ 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
```

注記

この例のコマンドは、指定したブロックデバイスに関連付けられた World Wide Name (WWN) またはシリアル番号として識別されるすべての値を返します。WWN を使用することが推奨されますが、WWN は特定のデバイスで常に利用できる訳ではなく、コマンド例で返される値は、**デバイスのシステム固有の ID** として使用することが許容されます。

2. **udev** ルールを設定します。以下の内容で **/etc/udev/rules.d/99-scheduler.rules** ファイルを作成します。

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{IDNAME}=="device system unique id", ATTR{queue/scheduler}="selected-scheduler"
```

ここでは、以下のようになります。

- **IDNAME** を、使用されている識別子名に置き換えます (例:**ID_WWN**)。
- **device system unique id** を、選択した識別子の値に置き換えます (例:**0x5002538d00000000**)。

3. **udev** ルールを再読み込みします。

```
# udevadm control --reload-rules
```

4. スケジューラー設定を適用します。

```
# udevadm trigger --type=devices --action=change
```

検証手順

- アクティブなスケジューラーを確認します。

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

12.7. 特定ディスクに任意のスケジューラーを一時的に設定

この手順では、特定のブロックデバイスに、特定のディスクスケジューラーを設定します。この設定は、システムを再起動すると元に戻ります。

手順

- 選択したスケジューラーの名前を、**/sys/block/device/queue/scheduler** ファイルに書き込みます。

```
# echo selected-scheduler > /sys/block/device/queue/scheduler
```

ファイル名の **device** を、**sdc** などのブロックデバイス名に置き換えます。

検証手順

- スケジューラーがデバイスでアクティブになっていることを確認します。

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

第13章 SAMBA サーバーのパフォーマンスチューニング

特定の状況で Samba のパフォーマンスを向上させることができる設定と、パフォーマンスに悪影響を与える可能性がある設定について説明します。

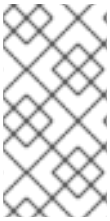
このセクションの一部は、Samba Wiki に公開されているドキュメント Performance Tuning に掲載されています。ライセンスは、CC BY 4.0 にあります。著者および貢献者は、Wiki ページの history タブを参照してください。

前提条件

- Samba がファイルまたはプリントサーバーとして設定されている。
  Samba をサーバーとして使用 を参照してください。

13.1. SMB プロトコルバージョンの設定

新しい SMB バージョンごとに機能が追加され、プロトコルのパフォーマンスが向上します。最新の Windows および Windows Server オペレーティングシステムは、常に最新のプロトコルバージョンに対応しています。Samba がプロトコルの最新バージョンも使用している場合は、Samba に接続する Windows クライアントで、このパフォーマンス改善を活用できます。Samba では、server max protocol のデフォルト値が、対応している安定した SMB プロトコルの最新バージョンに設定されます。



注記

常に最新の安定した SMB プロトコルバージョンを有効にするには、**server max protocol** パラメーターを設定しないでください。このパラメーターを手動で設定する場合は、最新のプロトコルバージョンを有効にするために、それぞれ新しいバージョンの SMB プロトコルで設定を変更する必要があります。

次の手順では、**server max protocol** パラメーターでデフォルト値を使用する方法を説明します。

手順

1. **/etc/samba/smb.conf** ファイルの **[global]** セクションから、**server max protocol** パラメーターを削除します。

2. Samba 設定を再読み込みします。

```
# smbcontrol all reload-config
```

13.2. 大量のファイルを含むディレクトリーとの共有の調整

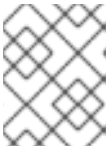
Linux は、大文字と小文字を区別するファイル名に対応しています。このため、Samba はファイルの検索時またはアクセス時に、大文字と小文字のファイル名のディレクトリーをスキャンする必要があります。小文字または大文字のいずれかでのみ新しいファイルを作成するように共有を設定すると、パフォーマンスが向上します。

前提条件

- Samba が、ファイルサーバーとして設定されている。

手順

1. 共有の全ファイルの名前を小文字に変更します。



注記

この手順の設定を使用すると、小文字以外の名前が付けられたファイルは表示されなくなります。

2. 共有のセクションに、以下のパラメーターを設定します。

```
case sensitive = true
default case = lower
preserve case = no
short preserve case = no
```

パラメーターの詳細は、man ページの **smb.conf(5)** を参照してください。

3. **/etc/samba/smb.conf** ファイルを検証します。

```
# testparm
```

4. Samba 設定を再読み込みします。

```
# smbcontrol all reload-config
```

この設定が適用されと、この共有に新たに作成されるすべてのファイルの名前が小文字になります。この設定により、Samba はディレクトリーを大文字と小文字で分けたスキャンが不要になり、パフォーマンスが向上します。

13.3. パフォーマンスが低下する可能性のある設定

デフォルトでは、Red Hat Enterprise Linux のカーネルは、ネットワークパフォーマンスが高くなるように調整されています。たとえば、カーネルはバッファーサイズに自動チューニングメカニズムを使用しています。**/etc/samba/smb.conf** ファイルに **socket options** パラメーターを設定すると、このカーネル設定が上書きされます。その結果、このパラメーターの設定により、ほとんどの場合は、Samba ネットワークのパフォーマンスが低下します。

カーネルの最適化された設定を使用するには、**/etc/samba/smb.conf** の **[global]** セクションから **socket options** パラメーターを削除します。

第14章 仮想マシンのパフォーマンスの最適化

仮想マシンでは、ホストと比べて、パフォーマンス低下が常に見られます。以下のセクションでは、この低下の理由を説明します。また、ハードウェアのインフラストラクチャーリソースを可能な限り効率的に使用できるように、RHEL 8 での仮想化によるパフォーマンスへの影響を最小限に抑える方法を説明します。

14.1. 仮想マシンのパフォーマンスに影響を及ぼすもの

仮想マシンは、ホストのユーザー空間プロセスとして実行します。したがって、ハイパーバイザーは、仮想マシンがホストシステムのリソースを使用できるように、ホストのシステムリソースを変換する必要があります。したがって、変換によりリソースの一部が消費されるため、仮想マシンのパフォーマンス効率は、ホストと同じにはなりません。

システムパフォーマンスにおける仮想化の影響
仮想マシンのパフォーマンス低下の理由には、以下のようなものがあります。

- 仮想 CPU (vCPU) がホスト上のスレッドとして実装され、Linux スケジューラーで処理される。
- 仮想マシンは、ホストカーネルから NUMA や Huge Page などの最適化機能を自動的に継承しない。
- ホストのディスクおよびネットワーク I/O の設定が、仮想マシンのパフォーマンスに大きく影響する可能性がある。
- ネットワークトラフィックは、一般的に、ソフトウェアベースのブリッジから仮想マシンに流れる。
- ホストデバイスとそのモデルによっては、その特定のハードウェアのエミュレーションにより、オーバーヘッドが著しくなる可能性がある。

仮想化が仮想マシンのパフォーマンスに与える影響の重大度は、次のようなさまざまな要因の影響を受けます。

- 同時に実行している仮想マシンの数
- 各仮想マシンで使用される仮想デバイスのサイズ
- 仮想マシンが使用するデバイスの種類

仮想マシンのパフォーマンス損失を減らす
RHEL 8 は、仮想化のパフォーマンスへの悪影響を減らすのに使用できる多くの機能を提供します。以下に例を示します。

- **tuned** サービス は、仮想マシンのリソースディストリビューションとパフォーマンスを自動的に最適化できる。
- ブロック I/O チューニング により、ディスクなどの仮想マシンのブロックデバイスのパフォーマンスを改善できる。
- NUMA のチューニング により、vCPU のパフォーマンスを向上させることができる。
- 仮想ネットワーク は、さまざまな方法で最適化できる。

重要

仮想マシンのパフォーマンスのチューニングは、その他の仮想化機能に悪影響を与える可能性があります。たとえば、変更した仮想マシンの移行がより困難になります。

14.2. TUNED を使用した仮想マシンのパフォーマンスの最適化

TuneD ユーティリティーは、CPU 集中型タスクや、ストレージネットワークスループットの応答などの特定のワークロードの特性に対して RHEL を調整するプロファイル配信メカニズムです。これにより、特定のユースケースで、パフォーマンスを強化し、電力消費を減らすように事前設定されたチューニングプロファイルを多数利用できます。これらのプロファイルを編集するか、新規プロファイルを作成して、仮想化環境に適したパフォーマンスソリューション (仮想化環境を含む) を作成できます。

RHEL 8 を仮想化に最適化するには、次のプロファイルを使用します。

- RHEL 8 仮想マシンの場合は、**virtual-guest** プロファイルを使用します。これは、一般的に適用された **throughput-performance** プロファイルをベースにしていますが、仮想メモリーのスワップは減少します。
- RHEL 8 仮想ホストの場合は、**virtual-host** プロファイルを使用します。これにより、ダーティーメモリーページのより集中的なライトバックが有効になり、ホストのパフォーマンスを活用できます。

前提条件

- **TuneD** サービスがインストールされており、有効になっている。

手順

特定の **TuneD** プロファイルを有効にするには、以下を実行します。

1. 使用可能な **Tuned** プロファイルをリスト表示します。

```
# 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. **(必要に応じて)** 新しい **TuneD** プロファイルを作成するか、既存の **TuneD** プロファイルを編集します。
   詳しくは、TuneD プロファイルのカスタマイズ を参照してください。

3. **TuneD** プロファイルをアクティベートします。

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

   - 仮想化ホストを最適化するには、**virtual-host** プロファイルを使用します。

```
# tuned-adm profile virtual-host
```

- RHEL ゲストオペレーティングシステムで、**virtual-guest** プロファイルを使用します。

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

関連情報

- システムの状態とパフォーマンスの監視と管理

14.3. 仮想マシンのメモリーの設定

仮想マシンのパフォーマンスを改善するために、追加のホスト RAM を仮想マシンに割り当てることができます。同様に、仮想マシンに割り当てるメモリー量を減らして、ホストメモリーを他の仮想マシンやタスクに割り当てることができます。

これらのアクションを実行するには、Web コンソール または コマンドラインインターフェイス を使用します。

14.3.1. Web コンソールを使用した仮想マシンのメモリーの追加および削除

仮想マシンのパフォーマンスを向上させるか、仮想マシンが使用するホストリソースを解放するために、Web コンソールを使用して、仮想マシンに割り当てられたメモリーの量を調整できます。

前提条件

- ゲスト OS がメモリーバルーンドライバーを実行している。これを確認するには、以下を実行します。

  1. 仮想マシンの設定に **memballoon** デバイスが含まれていることを確認します。

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

     このコマンドで出力が表示され、モデルが **none** に設定されていない場合は、**memballoon** デバイスが存在します。

  2. バルーンドライバーがゲスト OS で実行していることを確認します。

     - Windows ゲストでは、ドライバーは **virtio-win** ドライバーパッケージの一部としてインストールされます。手順は、Installing paravirtualized KVM drivers for Windows virtual machines を参照してください。

     - Linux ゲストでは、通常、このドライバーはデフォルトで含まれており、**memballoon** デバイスがあれば、アクティベートされます。

- Web コンソールの VM プラグインが システムにインストールされている。

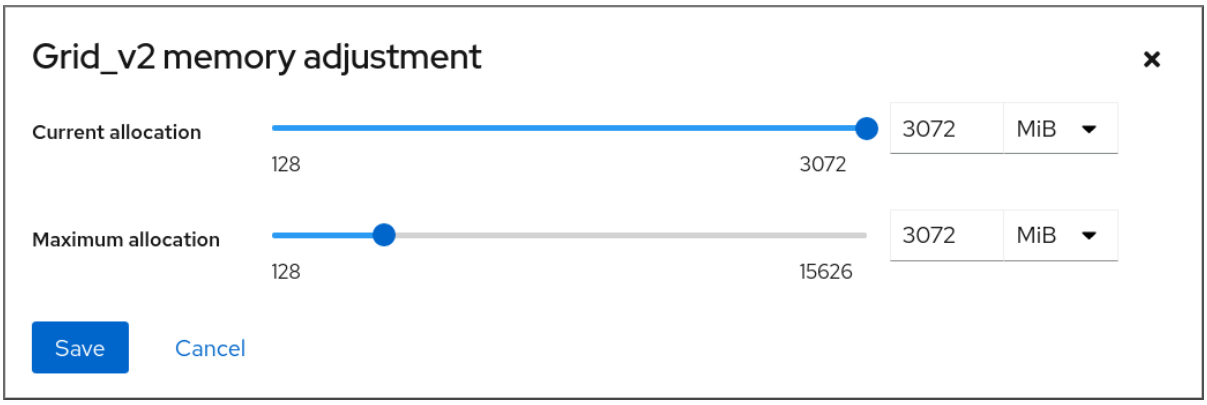
手順

1. **任意:** 最大メモリーと、仮想マシンに現在使用されている最大メモリーの情報を取得します。これは、変更のベースラインとしても、検証のためにも機能します。

```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。

3. 概要ペインで、**Memory** 行の横にある **編集** をクリックします。
メモリー調整 ダイアログが表示されます。



4. 選択した仮想マシンの仮想メモリーを設定します。

- **最大割り当て**: 仮想マシンがそのプロセスに使用できるホストメモリーの最大量を設定します。VM の作成時に最大メモリーを指定することも、後で増やすこともできます。メモリーは、MiB または GiB の倍数で指定できます。
仮想マシンをシャットダウンしてからでないと、最大メモリー割り当てを調整できません。

- **現在の割り当て** - 仮想マシンに割り当てる実際のメモリー量を設定します。この値は、最大割り当てより小さい値にすることができますが、上限を超えることはできません。値を調整して、仮想マシンで利用可能なメモリーをプロセス用に調整できます。メモリーは、MiB または GiB の倍数で指定できます。
この値を指定しない場合、デフォルトの割り当ては**最大割り当て** の値になります。

5. **Save** をクリックします。
仮想マシンのメモリー割り当てが調整されます。

関連情報

- コマンドラインインターフェイスを使用した仮想マシンのメモリーの追加と削除
- 仮想マシンの CPU パフォーマンスの最適化

14.3.2. コマンドラインインターフェイスを使用した仮想マシンのメモリーの追加と削除

仮想マシンのパフォーマンスを改善したり、使用しているホストリソースを解放したりするために、CLI を使用して仮想マシンに割り当てられたメモリーの量を調整できます。

前提条件

- ゲスト OS がメモリーバルーンドライバーを実行している。これを確認するには、以下を実行します。

  1. 仮想マシンの設定に **memballoon** デバイスが含まれていることを確認します。

     ```
     # virsh dumpxml testguest | grep memballoon
     <memballoon model='virtio'>
       </memballoon>
     ```

     このコマンドで出力が表示され、モデルが **none** に設定されていない場合は、**memballoon** デバイスが存在します。

  2. ballon ドライバーがゲスト OS で実行されていることを確認します。

     - Windows ゲストでは、ドライバーは **virtio-win** ドライバーパッケージの一部としてインストールされます。手順は、Installing paravirtualized KVM drivers for Windows virtual machines を参照してください。
     - Linux ゲストでは、通常、このドライバーはデフォルトで含まれており、**memballoon** デバイスがあれば、アクティベートされます。

手順

1. **任意:** 最大メモリーと、仮想マシンに現在使用されている最大メモリーの情報を取得します。これは、変更のベースラインとしても、検証のためにも機能します。

   ```
   # virsh dominfo testguest
   Max memory:     2097152 KiB
   Used memory:    2097152 KiB
   ```

2. 仮想マシンに割り当てる最大メモリーを調整します。この値を増やすと、仮想マシンのパフォーマンスが低下する可能性が向上し、値を減らすことで、仮想マシンがホスト上にあるパフォーマンスフットプリントが低減します。この変更は、停止している仮想マシンでのみ実行できるため、実行中の仮想マシンを調整するには再起動する必要があります。
   たとえば、仮想マシン **testguest** が使用可能な最大メモリーを 4096 MiB に変更するには、次のコマンドを実行します。

   ```
   # virt-xml testguest --edit --memory memory=4096,currentMemory=4096
   Domain 'testguest' defined successfully.
   Changes will take effect after the domain is fully powered off.
   ```

   実行中の仮想マシンの最大メモリーを増やすには、仮想マシンにメモリーデバイスを割り当てます。これは、**メモリーのホットプラグ**とも呼ばれます。詳細は、デバイスの仮想マシンへの接続 を参照してください。

警告

実行中の仮想マシン (メモリーのホットアンプラグとも呼ばれる) から、メモリーデバイスを削除することはサポートされておらず、Red Hat では推奨していません。

3. **任意:** 仮想マシンが現在使用しているメモリーを最大割り当てまで調整することもできます。これにより、仮想マシンの最大割り当てを変更せずに、仮想マシンが次回の再起動までホスト上にあるメモリー負荷が調整されます。

```
# virsh setmem testguest --current 2048
```

検証

1. 仮想マシンが使用するメモリーが更新されていることを確認します。

```
# virsh dominfo testguest
Max memory:     4194304 KiB
Used memory:    2097152 KiB
```

2. (**必要に応じて**) 現在の仮想マシンメモリーを調整すると、仮想マシンのメモリーバルーンの統計を取得して、そのメモリー使用量をどの程度効果的に調整するかを評価できます。

```
 # 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
```

関連情報

- Web コンソールを使用した仮想マシンのメモリーの追加および削除
- 仮想マシンの CPU パフォーマンスの最適化

14.3.3. 関連情報

- Attaching devices to virtual machines.

14.4. 仮想マシンの I/O パフォーマンスの最適化

仮想マシンの入出力 (I/O) 機能は、仮想マシンの全体的な効率を大幅に制限する可能性があります。これに対処するために、ブロック I/O パラメーターを設定して、仮想マシンの I/O を最適化できます。

14.4.1. 仮想マシンにおけるブロック I/O のチューニング

複数のブロックデバイスが、複数の仮想マシンで使用されている場合は、**I/O ウェイト** を変更して特定の仮想デバイスの I/O の優先度を調整することが重要になる場合があります。

デバイスの I/O ウェイトを上げると、I/O 帯域幅の優先度が高まるため、より多くのホストリソースが提供されます。同様に、デバイスのウェイトを下げると、ホストのリソースが少なくなります。

注記

各デバイスの **ウェイト** の値は **100** から **1000** の範囲内でなければなりません。もしくは、値を **0** にすると、各デバイスのリストからそのデバイスを削除できます。

手順

仮想マシンのブロック I/O パラメーターを表示および設定するには、以下を行います。

1. 仮想マシンの現在の **<blkio>** パラメーターを表示します。
   # 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. 指定したデバイスの I/O ウェイトを編集します。

```
# virsh blkiotune VM-name --device-weights device, I/O-weight
```

たとえば、次の例では、**testguest1** 仮想マシンの **/dev/sda** デバイスの重みを 500 に変更します。

```
# virsh blkiotune testguest1 --device-weights /dev/sda, 500
```

14.4.2. 仮想マシンのディスク I/O スロットリング

複数の仮想マシンが同時に実行する場合は、過剰なディスク I/O により、システムパフォーマンスに影響が及ぶ可能性があります。KVM 仮想化のディスク I/O スロットリングでは、仮想マシンからホストマシンに送られるディスク I/O 要求に制限を設定する機能を利用できます。これにより、仮想マシンが共有リソースを過剰に使用し、その他の仮想マシンのパフォーマンスに影響を及ぼすことを防ぐことができます。

ディスク I/O スロットリングを有効にするには、仮想マシンに割り当てられた各ブロックデバイスからホストマシンに送られるディスク I/O 要求に制限を設定します。

手順

1. **virsh domblklist** コマンドを使用して、指定された仮想マシン上のすべてのディスクデバイスの名前をリスト表示します。

```
# virsh domblklist rollin-coal
Target     Source
------------------------------------------------
vda        /var/lib/libvirt/images/rollin-coal.qcow2
sda        -
sdb        /home/horridly-demanding-processes.iso
```

2. スロットルする仮想ディスクがマウントされているホストブロックデバイスを見つけます。たとえば、前の手順の **sdb** 仮想ディスクをスロットリングする場合は、以下の出力では、ディスクが **/dev/nvme0n1p3** パーティションにマウントされていることを示しています。

```
$ 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** コマンドを使用して、ブロックデバイスの I/O 制限を設定します。

```
# virsh blkiotune VM-name --parameter device,limit
```

以下の例は、**rollin-coal** 仮想マシン上の **sdb** ディスクを毎秒 1000 の読み書き操作にスロットリングし、毎秒 50 MB の読み書きスループットにスロットリングします。

```
# 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
```

関連情報

- ディスク I/O スロットリングは、異なる顧客に属する仮想マシンが同じホストで実行されている場合や、異なる仮想マシンに QoS 保証が提供されている場合など、さまざまな状況で役立ちます。ディスク I/O スロットリングは、低速なディスクをシミュレートするために使用することもできます。
- I/O スロットリングは、仮想マシンに割り当てられた各ブロックデバイスに個別に適用でき、スループットおよび I/O 操作の制限に対応します。
- Red Hat は、**virsh blkdeviotune** コマンドを使用した仮想マシンでの I/O スロットリングの設定はサポートしていません。RHEL 8 を仮想マシンホストとして使用する場合にサポートされていない機能の詳細は、RHEL 8 仮想化でサポートされていない機能 を参照してください。

14.4.3. マルチキュー virtio-scsi の有効化

仮想マシンで **virtio-scsi** ストレージデバイスを使用する場合は、**マルチキュー virtio-scsi** 機能により、ストレージパフォーマンスおよびスケーラビリティーが向上します。このため、各仮想 CPU (vCPU) に別のキューを持たせることが可能になります。また仮想 CPU は、その他の vCPU に影響を及ぼすことなく使用するために、割り込みできるようになります。

手順

- 特定の仮想マシンに対してマルチキュー virtio-scsi サポートを有効にするには、仮想マシンの XML 設定に以下を追加します。ここでの **N** は、vCPU キューの合計数です。

```
<controller type='scsi' index='0' model='virtio-scsi'>
  <driver queues='N' />
</controller>
```

14.5. 仮想マシンの CPU パフォーマンスの最適化

vCPU は、ホストマシンの物理 CPU と同様、仮想マシンのパフォーマンスにおいて極めて重要です。したがって、vCPU を最適化すると、仮想マシンのリソース効率に大きな影響を及ぼす可能性があります。vCPU を最適化するには、以下を実行します。

1. 仮想マシンに割り当てられているホスト CPU の数を調整します。これは、CLI または Web コンソール を使用して実行できます。

2. vCPU モデルが、ホストの CPU モデルに調整されていることを確認します。たとえば、仮想マシン **testguest1** を、ホストの CPU モデルを使用するように設定するには、次のコマンドを実行します。

```
# virt-xml testguest1 --edit --cpu host-model
```

3. Kernel Same-page Merging (KSM) を無効にします 。

4. ホストマシンが Non-Uniform Memory Access (NUMA) を使用する場合は、その仮想マシンに対して **NUMA を設定** することもできます。これにより、ホストの CPU およびメモリープロセスが、仮想マシンの CPU およびメモリープロセスにできるだけ近くにマッピングされます。事実上、NUMA チューニングにより、仮想マシンに割り当てられたシステムメモリーへのより効率的なアクセスが可能になります。これにより、vCPU 処理の効果が改善されます。
詳細は、仮想マシンで NUMA の設定 および サンプルの vCPU パフォーマンスチューニングシナリオ を参照してください。

14.5.1. コマンドラインインターフェイスを使用した仮想 CPU の追加と削除

仮想マシンの CPU パフォーマンスを増減するには、仮想マシンに割り当てられた仮想 CPU (vCPU) を追加または削除します。

実行中の仮想マシンで実行する場合、これは vCPU ホットプラグおよびホットアンプラグとも呼ばれます。ただし、RHEL 8 では vCPU のホットアンプラグに対応しておらず、Red Hat ではその使用を強く推奨していません。

前提条件

- **オプション:** ターゲット仮想マシン内の vCPU の現在の状態を表示します。たとえば、仮想マシン **testguest** 上の仮想 CPU 数を表示するには、以下を実行します。

```
# virsh vcpucount testguest
maximum      config         4
maximum      live           2
current      config         2
current      live           1
```

この出力は、**testguest** が現在 1 vCPU を使用していることを示し、1 つ以上の vCPU をホットプラグして仮想マシンのパフォーマンスを向上できることを示しています。ただし、再起動後に使用される vCPU の **testguest** 数は 2 に変更され、2 以上の vCPU のホットプラグが可能になります。

手順

1. 仮想マシンに割り当てることができる vCPU の最大数を調整します。これは、仮想マシンの次回起動時に有効になります。
   たとえば、仮想マシン **testguest** の vCPU の最大数を 8 に増やすには、次のコマンドを実行します。

```
# virsh setvcpus testguest 8 --maximum --config
```

   最大値は、CPU トポロジー、ホストハードウェア、ハイパーバイザー、およびその他の要素によって制限される可能性があることに注意してください。

2. 仮想マシンに割り当てられている現在の仮想 CPU の数を調整し、直前のステップで設定された最大数まで調整します。以下に例を示します。

   - 実行中の仮想マシン **testguest** にアタッチされている vCPU を 4 に増やすには、以下を実行します。

```
# virsh setvcpus testguest 4 --live
```

     これにより、仮想マシンの次回の起動まで、仮想マシンのパフォーマンスおよび **testguest** のホスト負荷のフットプリントが高まります。

   - **testguest** 仮想マシンにアタッチされている vCPU の数を永続的に 1 に減らすには、次のコマンドを実行します。

```
# virsh setvcpus testguest 1 --config
```

     これにより、仮想マシンの次回の起動後に、仮想マシンのパフォーマンスおよび **testguest** のホスト負荷のフットプリントが低下します。ただし、必要に応じて、仮想マシンに追加の vCPU をホットプラグして、一時的にパフォーマンスを向上させることができます。

検証

- 仮想マシンの vCPU の現在の状態に変更が反映されていることを確認します。

```
# virsh vcpucount testguest
maximum      config         8
maximum      live           4
current      config         1
current      live           4
```

関連情報

- Web コンソールを使用した仮想 CPU の管理

14.5.2. Web コンソールを使用した仮想 CPU の管理

RHEL 9 Web コンソールを使用して、Web コンソールが接続している仮想マシンが使用する仮想 CPU を確認し、設定できます。

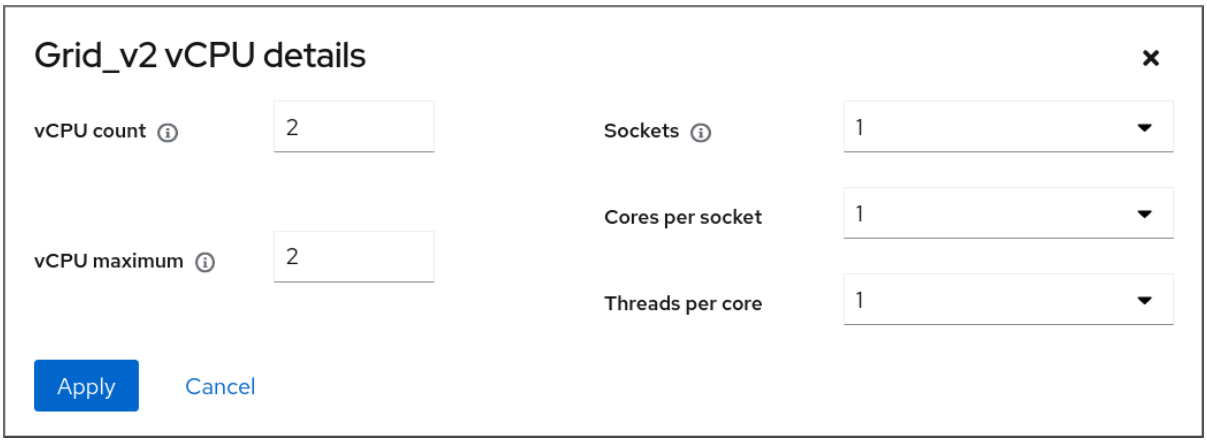
前提条件

- Web コンソールの VM プラグインが システムにインストールされている。

手順

1. **仮想マシン** インターフェイスで、情報を表示する仮想マシンを選択します。
新しいページが開き、選択した仮想マシンに関する基本情報を含む Overview セクションと、仮想マシンのグラフィカルインターフェイスにアクセスするための Console セクションが表示されます。

2. 概要ペインで、vCPU の数の横にある **編集** をクリックします。
vCPU の詳細ダイアログが表示されます。



1. 選択した仮想マシンの仮想 CPU を設定します。

- **vCPU 数**: 現在使用中の vCPU の数

注記

vCPU 数は、vCPU 最大値以下にする必要があります。

- **vCPU 最大値** - 仮想マシンに設定できる仮想 CPU の最大数を入力します。この値が **vCPU 数** よりも大きい場合には、vCPU を追加で仮想マシンに割り当てることができます。
- **ソケット** - 仮想マシンに公開するソケットの数を選択します。
- **ソケットごとのコア** - 仮想マシンに公開する各ソケットのコア数を選択します。
- **コアあたりのスレッド** - 仮想マシンに公開する各コアのスレッド数を選択します。
Sockets、**Cores per socket**、および **Threads per core** オプションは、仮想マシンの CPU トポロジーを調整することに注意してください。これは、vCPU のパフォーマンスにメリットがあり、ゲスト OS の特定のソフトウェアの機能に影響を与える可能性があります。デプロイメントで別の設定が必要ない場合は、デフォルト値のままにします。

2. **Apply** をクリックします。
仮想マシンに仮想 CPU が設定されます。

注記

仮想 CPU 設定の変更は、仮想マシンの再起動後にのみ有効になります。

関連情報

- コマンドラインインターフェイスを使用した仮想 CPU の追加と削除

14.5.3. 仮想マシンでの NUMA の設定

以下の方法は、RHEL 8 ホストで、仮想マシンの Non-Uniform Memory Access (NUMA) 設定の設定に使用できます。

前提条件

- ホストが NUMA 対応のマシンである。これを確認するには、**virsh nodeinfo** コマンドを使用して、**NUMA cell(2)** の行を確認します。

```
# 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
```

  行の値が 2 以上であると、そのホストは NUMA に対応しています。

手順

使いやすさのため、自動化ユーティリティーとサービスを使用して、仮想マシンの NUMA を設定できます。ただし、手動で NUMA を設定すると、パフォーマンスが大幅に向上する可能性が高くなります。

自動方式

- 仮想マシンの NUMA ポリシーを **Preferred** に設定します。たとえば、仮想マシン **testguest5** に対してこれを行うには、次のコマンドを実行します。

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- ホストで NUMA の自動負荷分散を有効にします。

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- **umad** サービスを起動して、メモリーリソースで仮想マシンの CPU を自動的に調整します。

```
# systemctl start numad
```

手動方式

1. 特定ホストの CPU、またはある範囲の CPU に特定の vCPU スレッドをピニングします。これは、NUMA 以外のホストおよび仮想マシンでも可能で、vCPU のパフォーマンスを向上させる安全な方法として推奨されています。
   たとえば、次のコマンドでは、仮想マシン **testguest6** の vCPU スレッドの 0 から 5 を、ホストの CPU 1、3、5、7、9、11 にそれぞれピニングします。

```
# 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
```

   その後、これが成功したかどうかを確認できます。

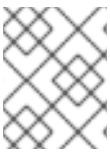
```
# virsh vcpupin testguest6
VCPU   CPU Affinity
----------------------
0      1
1      3
2      5
3      7
4      9
5      11
```

2. vCPU スレッドのピニング後に、指定の仮想マシンに関連付けられた QEMU プロセススレッドを、特定ホスト CPU、またはある範囲の CPU に固定することもできます。たとえば、以下のコマンドは、**testguest6** の QEMU プロセススレッドを CPU 13 および 15 にピニングし、これが成功したことを確認します。

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
----------------------------------
       *: 13,15
```

3. これで、特定の仮想マシンに対して割り当てられるホストの NUMA ノードを指定することができます。これにより、仮想マシンの vCPU によるホストメモリーの使用率が向上します。たとえば、次のコマンドでは、ホスト NUMA ノード 3 ～ 5 を使用するように **testguest6** を設定し、これが成功したかどうかを確認します。

```
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```



注記

最善のパフォーマンス結果を得るためにも、上記の手動によるチューニングメソッドをすべて使用することが推奨されます。

既知の問題

- 現時点では IBM Z ホストで NUMA チューニングを実行できない

関連情報

- vCPU のパフォーマンスチューニングシナリオ例
- View the current NUMA configuration of your system using the **numastat** utility

14.5.4. vCPU のパフォーマンスチューニングシナリオ例

最適な vCPU パフォーマンスを得るためにも、たとえば以下のシナリオのように、手動で **vcpupin**、**emulatorpin**、および **numatune** 設定をまとめて使用することが推奨されます。

開始シナリオ

- ホストには以下のハードウェア仕様があります。
  - 2 つの NUMA ノード
  - 各ノードにある 3 つの CPU コア
  - 各コアにある 2 スレッド

  このようなマシンの **virsh nodeinfo** の出力は以下のようになります。

```
# 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
```

- 既存の仮想マシンを変更して、8 つの vCPU を使用できるようにします。これは、1 つの NUMA ノードに収まらないことを意味します。
  したがって、各 NUMA ノードに 4 つの vCPU を分散し、vCPU トポロジーをホストトポロジーに可能な限り近づけるようにする必要があります。つまり、指定の物理 CPU のシブリングスレッドとして実行される vCPU は、同じコア上のホストスレッドに固定 (ピニング) される必要があります。詳細は、以下の **ソリューション** を参照してください。

解決方法

1. ホストトポロジーに関する情報を取得します。

```
# virsh capabilities
```

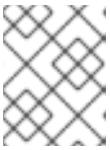
   この出力には、以下のようなセクションが含まれます。

```
<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. **(必要に応じて)** 適用可能なツールおよびユーティリティー を使用して、仮想マシンのパフォーマンスをテストします。

3. ホストに 1 GiB の Huge Page を設定してマウントします。



注記

1 GiB huge page は、ARM 64 ホストなどの一部のアーキテクチャーおよび設定では使用できない場合があります。

a. ホストのカーネルコマンドラインに次の行を追加します。

```
default_hugepagesz=1G hugepagesz=1G
```

b. **/etc/systemd/system/hugetlb-gigantic-pages.service** ファイルを以下の内容で作成します。

```
[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. **/etc/systemd/hugetlb-reserve-pages.sh** ファイルを以下の内容で作成します。

```
#!/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
```

これにより、4 つの 1GiB の Huge Page が **node1** から予約され、さらに別の 4 つの 1 GiB の Huge Page が **node2** から予約されます。

d. 前の手順で作成したスクリプトを実行ファイルにします。

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e. システムの起動時に Huge Page 予約を有効にします。

```
# systemctl enable hugetlb-gigantic-pages
```

4. **virsh edit** コマンドを使用して、最適化する仮想マシンの XML 設定 (この例では **super-VM**) を編集します。

```
# virsh edit super-vm
```

5. 次の方法で仮想マシンの XML 設定を調整します。

a. 仮想マシンが 8 つの静的 vCPU を使用するように設定します。これを行うには、**<vcpu/>** 要素を使用します。

b. トポロジーでミラーリングする、対応するホスト CPU スレッドに、各 vCPU スレッドをピニングします。これを行うには、**<cputune>** セクションの **<vcpupin/>** 要素を使用します。
上記の **virsh 機能** ユーティリティーで示されているように、ホストの CPU スレッドは、各コアで連続的に順次付けされません。また、vCPU スレッドは、同じ NUMA ノード上のホ

ストのコアの利用可能な最大セットに固定される必要があります。表の図については、以下の **トポロジーの例** セクションを参照してください。

手順 a と b の XML 設定は次のようになります。

```
<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. 1 GiB の Huge Page を使用するように仮想マシンを設定します。

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
  </hugepages>
</memoryBacking>
```

d. ホスト上で対応する NUMA ノードからメモリーを使用するように、仮想マシンの NUMA ノードを設定します。これを行うには、**<numatune/>** セクションの **<memnode/>** 要素を使用します。

```
<numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

e. CPU モードが **host-passthrough** に設定され、CPU が **passthrough** モードでキャッシュを使用していることを確認します。

```
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
```

6. 仮想マシンの XML 設定に、以下のようなセクションが含まれていることを確認します。

```
[...]
 <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>
```

7. **(必要に応じて)** アプリケーションツールおよびユーティリティー を使用して仮想マシンのパフォーマンスをテストし、仮想マシンの最適化への影響を評価します。

トポロジーの例

- 以下の表は、ピニングされる必要のある vCPU とホスト CPU 間の接続を示しています。

表14.1 ホストトポロジー

CPU スレッド	0	3	1	4	2	5	6	9	7	10	8	11
コア	0		1		2		3		4		5	
ソケット	0						1					
NUMA ノード	0						1					

表14.2 仮想マシントポロジー

vCPU スレッド	0	1	2	3	4	5	6	7

コア	0	1	2	3
ソケット	0		1	
NUMA ノード	0		1	

表14.3 ホストと仮想マシントポロジーの組み合わせ

vCPU スレッド		0	1	2	3		4	5	6	7
ホストの CPU スレッド	0	3	1	4	2	5	6	9	7	10	8	11
コア	0		1		2		3		4		5	
ソケット	0						1					
NUMA ノード	0						1					

このシナリオでは、2 つの NUMA ノードと 8 つの vCPU があります。したがって、4 つの vCPU スレッドは各ノードに固定 (ピニング) される必要があります。

また、Red Hat では、ホストシステムの操作のために、各ノードで少なくとも 1 つの CPU スレッドを使用できるようにしておくことを推奨します。

以下の例では、NUMA ノードにはそれぞれ 3 コアで、2 個のホスト CPU スレッドがあるため、ノード 0 のセットは、以下のように変換できます。

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

14.5.5. Kernel Same-page Merging の無効化

Kernel same-page merging (KSM) はメモリーの密度を向上させますが、CPU 使用率が増え、ワークロードによっては全体のパフォーマンスに悪影響を与える可能性があります。このような場合には、KSM を無効にすることで、仮想マシンのパフォーマンスを向上させることができます。

要件に応じて、KSM を 1 回のセッションだけ無効にしたり、永続的に無効にしたりできます。

手順

- KSM をセッション 1 回分無効にするには、**systemctl** ユーティリティーを使用して **ksm** サービスおよび **ksmtuned** サービスを停止します。

```
# systemctl stop ksm

# systemctl stop ksmtuned
```

- KSM を永続的に無効にするには、**systemctl** ユーティリティーを使用して **ksm** サービスおよび **ksmtuned** サービスを無効にします。

```
# 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.
```

注記

KSM を無効にする前に仮想マシン間で共有されていたメモリーページは、そのまま共有されます。共有を停止するには、以下のコマンドを使用して、システムの **PageKSM** ページをすべて削除します。

```
# echo 2 > /sys/kernel/mm/ksm/run
```

KSM ページを匿名ページに置き換えると、**khugepaged** カーネルサービスは仮想マシンの物理メモリーに透過的なヒュージページを再ビルドします。

14.6. 仮想マシンのネットワークパフォーマンスの最適化

仮想マシンのネットワークインターフェイスカード (NIC) の性質上、仮想マシンは、割り当てられているホストネットワークの帯域幅の一部を失います。これにより、仮想マシンの全体的なワークロード効率が削減されることがあります。以下のヒントは、仮想 NIC (vNIC) のスループットで仮想化の影響を最小限に抑えることができます。

手順

以下の方法のいずれかを使用し、仮想マシンのネットワークパフォーマンスにメリットがあるかどうかを調べます。

vhost_net モジュールの有効化

ホストで **vhost_net** カーネル機能が有効になっていることを確認します。

```
# lsmod | grep vhost
vhost_net              32768  1
vhost                  53248  1 vhost_net
tap                    24576  1 vhost_net
tun                    57344  6 vhost_net
```

このコマンドの出力が空白である場合は、**vhost_net** カーネルモジュールを有効にします。

```
# modprobe vhost_net
```

マルチキュー virtio-net の設定

仮想マシンに **マルチキュー virtio-net** 機能を設定するには、**virsh edit** コマンドを使用して、仮想マシンの XML 設定を編集します。XML で、以下を **<devices>** セクションに追加し、**N** を、仮想マシンの vCPU 数 (最大 16) に変更します。

```
<interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
```

```
    <driver name='vhost' queues='N'/>
</interface>
```

仮想マシンが実行中の場合は、再起動して変更を適用します。

ネットワークパケットのバッチ処理

転送パスが長い Linux の仮想マシン設定では、パケットをバッチ処理してからカーネルに送信することで、キャッシュが有効に活用される場合があります。パケットバッチ機能を設定するには、ホストで次のコマンドを実行し、**tap0** を、仮想マシンが使用するネットワークインターフェイスの名前に置き換えます。

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

ホスト NIC が SR-IOV に対応している場合は、vNIC に SR-IOV デバイス割り当てを使用します。詳細は、SR-IOV デバイスの管理 を参照してください。

関連情報

- 仮想ネットワークの概要

14.7. 仮想マシンのパフォーマンス監視ツール

最も多くの仮想マシンリソースを消費するものと、仮想マシンで最適化を必要とする部分を認識するために、一般的なパフォーマンス診断ツールや仮想マシン固有のパフォーマンス診断ツールを使用できます。

デフォルトの OS パフォーマンス監視ツール

標準のパフォーマンス評価には、ホストおよびゲストのオペレーティングシステムでデフォルトで提供されるユーティリティーを使用できます。

- RHEL 8 ホストで、root として **top** ユーティリティーまたは **システムモニター** アプリケーションを使用し、出力結果から **qemu** と **virt** を見つけます。これは、仮想マシンが消費しているホストシステムのリソースのサイズを示します。
  - 監視ツールにおいて、**qemu** プロセスまたは **virt** プロセスのいずれかで、ホストの CPU またはメモリーの容量を大幅に消費していることが示されている場合は、**perf** ユーティリティーを使用して調査を行います。詳細は以下を参照してください。
  - また、**vhost_net** スレッドプロセス (例: **vhost_net-1234**) が、ホストの CPU 容量を過剰に消費する際に表示される場合は、**multi-queue virtio-net** などの 仮想ネットワークの最適化機能 を使用することを検討してください。
- ゲストオペレーティングシステムでは、システムで利用可能なパフォーマンスユーティリティーとアプリケーションを使用して、どのプロセスが最も多くのシステムリソースを消費するかを評価します。
  - Linux システムでは、**top** ユーティリティーを使用できます。
  - Windows システムでは、**Task Manager** アプリケーションを使用できます。

perf kvm

perf ユーティリティーを使用して、RHEL 8 ホストのパフォーマンスに関する仮想化固有の統計を収集および分析できます。これを行うには、以下を行います。

1. ホストに、**perf** パッケージをインストールします。

```
# yum install perf
```

2. **perf kvm stat** コマンドの 1 つを使用して、仮想化ホストの perf 統計を表示します。

   - お使いのハイパーバイザーのリアルタイム監視には、**perf kvm stat live** コマンドを使用します。
   - 一定期間でハイパーバイザーの perf データをログに記録するには、**perf kvm stat record** コマンドを使用してロギングを有効にします。コマンドをキャンセルまたは中断した後、データは **perf.data.guest** ファイルに保存されます。これは、**perf kvm stat report** コマンドを使用して分析できます。

3. **VM-EXIT** イベントとそのディストリビューションのタイプについて **perf** 出力を分析します。たとえば、**PAUSE_INSTRUCTION** イベントは頻繁に存在すべきではありませんが、以下の出力では、このイベントが頻繁に現れ、ホスト CPU が vCPU を適切に処理していないことを示しています。このようなシナリオでは、アクティブな一部の仮想マシンの電源オフ、その仮想マシンからの vCPU の削除、または vCPU のパフォーマンスの調整 を検討してください。

```
# perf kvm stat report

Analyze events for all VMs, all VCPUs:


             VM-EXIT    Samples  Samples%     Time%    Min Time    Max Time         Avg time

  EXTERNAL_INTERRUPT     365634    31.59%    18.04%      0.42us  58780.59us    204.08us ( +-   0.99% )
            MSR_WRITE     293428    25.35%     0.13%      0.59us  17873.02us      1.80us ( +-   4.63% )
     PREEMPTION_TIMER     276162    23.86%     0.23%      0.51us  21396.03us      3.38us ( +-   5.19% )
   PAUSE_INSTRUCTION     189375    16.36%    11.75%      0.72us  29655.25us    256.77us ( +-   0.70% )
                 HLT      20440     1.77%    69.83%      0.62us  79319.41us  14134.56us ( +-   0.79% )
              VMCALL      12426     1.07%     0.03%      1.02us   5416.25us      8.77us ( +-   7.36% )
      EXCEPTION_NMI         27     0.00%     0.00%      0.69us      1.34us      0.98us ( +-   3.50% )
      EPT_MISCONFIG          5     0.00%     0.00%      5.15us     10.85us      7.88us ( +-  11.67% )

Total Samples:1157497, Total events handled time:413728274.66us.
```

   perf kvm stat の出力で問題を知らせる他のイベントタイプには、以下が含まれます。

   - **INSN_EMULATION** - 準最適な 仮想マシンの I/O 設定 を示します。

perf を使用した仮想化パフォーマンスを監視する方法は、**perf-kvm** man ページを参照してください。

numastat

システムの現在の NUMA 設定を表示するには、**numastat** ユーティリティーを使用できます。これは **numactl** パッケージをインストールすることで利用できます。

以下は、4 つの実行中の仮想マシンが含まれるホストを示しています。それぞれは、複数の NUMA ノードからメモリーを取得しています。これは、vCPU のパフォーマンスに対して最適なのではなく、保証調整 です。

```
# 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
```

一方、以下では、1 つのノードで各仮想マシンに提供されているメモリーを示しています。これは、より一層効率的です。

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51747 (qemu-kvm)      0      0      7      0   8072      0      1      0  8080
53736 (qemu-kvm)      0      0      7      0      0      0   8113      0  8120
53773 (qemu-kvm)      0      0      7      0      0      0      1   8110  8118
59065 (qemu-kvm)      0      0   8050      0      0      0      0      0  8051
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                 0      0   8072      0   8072      0   8114   8110 32368
```

14.8. 関連情報

- Windows 仮想マシンの最適化

第15章 電源管理の重要性

コンピューターシステム全体の消費電力を削減することで、コストを削減できます。各システムコンポーネントのエネルギー消費を効果的に最適化するには、システムが実行するさまざまなタスクを検討し、各コンポーネントのパフォーマンスがそのジョブに対して正しいことを確認するように設定します。特定コンポーネントやシステム全体の消費電力を下げると、熱およびパフォーマンスが低下します。

適切な電源管理を行うと、以下のような結果になります

- サーバーやコンピューティングセンターにおける熱の削減
- 冷却、空間、ケーブル、ジェネレーター、無停電電源装置 (UPS) などの二次コストの削減
- ノートパソコンのバッテリー寿命の延長
- 二酸化炭素排出量の削減
- 政府の規制や Green IT (Energy Star など) に関する法的要件に合致する
- 新システムに関する企業ガイドラインの遵守

本セクションでは、Red Hat Enterprise Linux システムの電源管理に関する情報を説明します。

15.1. 電源管理の基本

効果的な電源管理は、以下の原則に基づいて行われます。

An idle CPU should only wake up when needed

Red Hat Enterprise Linux 6 以降では、カーネルが **tickless** を実行しています。つまり、以前の定期的なタイマー割り込みが、オンデマンド割り込みに置き換えられたことを意味します。そのため、新しいタスクが処理のキューに追加されるまで、アイドル状態の CPU はアイドル状態を維持できます。低電力状態にある CPU は、この状態を持続できます。ただし、システムに、不要なタイマーイベントを作成するアプリケーションが存在する場合は、この機能の利点が相殺される可能性があります。ボリュームの変更やマウスの動きの確認などのポーリングイベントは、このようなイベントの例です。

Red Hat Enterprise Linux には、CPU 使用率に基づいてアプリケーションを識別し、監査するツールが同梱されています。詳細は、Audit and analysis overview および Tools for auditing を参照してください。

Unused hardware and devices should be disabled completely

これは、ハードディスクなど、動作する部品があるデバイスに当てはまります。また、一部のアプリケーションでは、使用されていない有効なデバイスが "open" 状態のままにすることがあります。これが発生すると、カーネルは、そのデバイスが使用中であることを想定します。これにより、そのデバイスが省電力状態にならないようにできます。

Low activity should translate to low wattage

ただし、多くの場合、これは最新のハードウェアと、x86 以外のアーキテクチャーを含む最新のシステムの正しい BIOS 設定または UEFI に依存します。システムに最新の公式ファームウェアを使用していること、および BIOS の電源管理またはデバイス設定セクションで電源管理機能が有効になっていることを確認してください。以下のような機能を確認してください。

- ARM64 用の CPPC (Collaborative Processor Performance Controls) のサポート
- IBM Power Systems の PowerNV サポート

- SpeedStep
- PowerNow!
- Cool'n'Quiet
- ACPI (C-state)
- Smart
  ハードウェアでこの機能に対応し、BIOS で有効になっている場合は、Red Hat Enterprise Linux がデフォルトで使用します。

Different forms of CPU states and their effects

最新の CPU は、ACPI (Advanced Configuration and Power Interface) とともに、さまざまな電源状態を提供します。3 つの異なる状態は以下のとおりです。

- スリープ (C-state)
- 周波数と電圧 (P-state)
- 熱の出力 (T-states または thermal state)
  最小のスリープ状態で実行している CPU は、最小のワット数を消費しますが、必要に応じてその状態からウェイクアップするのにかかる時間も大幅に長くなります。まれに、スリープ状態に切り替わるたびに CPU が即座にウェイクアップしなければならなくなることがあります。この状況は、実質的に永続的に CPU がビジー状態になり、別の状態を使用すると潜在的な省電力の一部が失われます。

A turned off machine uses the least amount of power

電力を節約する最善の方法の 1 つは、システムの電源を切ることです。たとえば、会社では、昼休みや帰宅時にマシンをオフにするガイドラインを使用して、Green IT を意識することに焦点をあてた企業文化を育成できます。また、複数の物理サーバーを 1 つの大きなサーバーに統合し、Red Hat Enterprise Linux に同梱される仮想化技術を使用して仮想化することもできます。

15.2. 監査および分析の概要

通常、1 つのシステムで詳細な手動の監査、分析、およびチューニングを行う場合は例外となります。これは、このようなシステム調整の最後の部分から得られる利点よりも、その実行にかかる時間とコストの方が長いためです。

ただし、すべてのシステムで同じ設定を再利用できるほぼ同一のシステムでこれらのタスクを 1 回実行することは、非常に便利です。たとえば、数千ものデスクトップシステムや、マシンがほぼ同一の HPC クラスターをデプロイメントする場合を考えてください。監査と分析を行うもう 1 つの理由は、将来のシステム動作のリグレッションまたは変更を特定できる比較の基礎を提供することです。この分析の結果は、ハードウェア、BIOS、またはソフトウェアの更新が定期的に行われ、消費電力に関する予期しない事態を回避したい場合に非常に役立ちます。通常、徹底的な監査と分析により、特定システムで実際に起こっていることをより的確に把握できます。

利用可能な最新のシステムを使用しても、消費電力に関するシステムの監査と分析は比較的困難です。ほとんどのシステムは、ソフトウェアを介して電力使用量を測定するために必要な手段を提供していません。ただし、例外があります。

- Hewlett Packard サーバーシステムの iLO 管理コンソールには、Web からアクセスできる電源管理モジュールがあります。
- IBM は、BladeCenter 電源管理モジュールで同様のソリューションを提供します。

- 一部の Dell システムでは、IT Assistant は電力監視機能も提供します。

他のベンダーは、サーバープラットフォームで同様の機能を提供する可能性が高くなりますが、すべてのベンダーで対応している唯一のソリューションは存在しません。多くの場合、消費電力を直接測定する必要があるのは、可能な限り節約を最大化するためだけです。

15.3. 監査用ツール

Red Hat Enterprise Linux 8 には、システムの監査および分析を実行できるツールが同梱されています。これらのほとんどは、すでに発見したものを確認したい場合、または特定の部分についてさらに詳細な情報が必要な場合の補助情報源として使用できます。

このツールの多くは、パフォーマンスの調整にも使用されます。以下に例を示します。

PowerTOP

これは、CPU を頻繁にウェイクアップするカーネルおよびユーザー空間アプリケーションの特定のコンポーネントを識別します。root で **powertop** コマンドを使用して **PowerTop** ツールを起動し、**powertop --calibrate** で電力見積もりエンジンを調整します。PowerTop の詳細は、PowerTOP を使用した電力消費の管理 を参照してください。

Diskdevstat and netdevstat

これは、システムで実行しているすべてのアプリケーションのディスクアクティビティーとネットワークアクティビティーに関する詳細情報を収集する SystemTap ツールです。これらのツールによって収集された統計を使用すると、少数の大規模な操作ではなく、多くの小規模な I / O 操作で電力を浪費するアプリケーションを特定できます。root で **yum install tuned-utils-systemtap kernel-debuginfo** コマンドを使用し、**diskdevstat** および **netdevstat** をインストールします。ディスクとネットワークのアクティビティーの詳細情報を表示するには、次のコマンドを実行します。

```
# diskdevstat

PID  UID  DEV  WRITE_CNT  WRITE_MIN  WRITE_MAX  WRITE_AVG  READ_CNT
READ_MIN  READ_MAX  READ_AVG  COMMAND

3575  1000  dm-2  59       0.000     0.365      0.006      5       0.000      0.000     0.000
mozStorage #5
3575  1000  dm-2  7        0.000     0.000      0.000      0       0.000      0.000     0.000
localStorage DB
[...]


# netdevstat

PID  UID  DEV     XMIT_CNT  XMIT_MIN  XMIT_MAX  XMIT_AVG  RECV_CNT
RECV_MIN  RECV_MAX  RECV_AVG  COMMAND
3572  991  enp0s31f6  40      0.000     0.882      0.108      0       0.000      0.000     0.000
openvpn
3575  1000 enp0s31f6  27      0.000     1.363      0.160      0       0.000      0.000     0.000
Socket Thread
[...]
```

このコマンドでは、**update_interval**、**total_duration**、および **display_histogram** の 3 つのパラメーターを指定できます。

TuneD

これは、**udev** デバイスマネージャーを使用して、接続されたデバイスを監視し、システム設定の静的チューニングおよび動的チューニングの両方を有効にするプロファイルベースのシステムチューニングツールです。**tuned-adm recommend** コマンドを使用すると、Red Hat が特定の製品に最も適したプロファイルを判別できます。TuneD の詳細は、TuneD を使い始める および TuneD プロファイルのカスタマイズ を参照してください。**powertop2tuned utility** を使用して、**PowerTOP** の提案からカスタムの TuneD プロファイルを作成できます。**powertop2tuned** ユーティリティーの詳細は、電力消費の最適化 を参照してください。

Virtual memory statistics (vmstat)

これは、**procps-ng** パッケージにより提供されます。このツールを使用すると、プロセス、メモリー、ページング、ブロック I/O、トラップ、および CPU アクティビティーの詳細情報を表示できます。
この情報を表示するには、次を使用します。

```
$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
r  b  swpd  free    buff   cache   si   so  bi   bo   in  cs  us  sy id  wa  st
1  0    0   5805576 380856 4852848   0    0  119  73  814  640  2   2 96   0   0
```

vmstat -a コマンドを使用すると、アクティブメモリーと非アクティブメモリーを表示できます。その他の **vmstat** オプションの詳細は、**vmstat** の man ページを参照してください。

iostat

このツールは **sysstat** パッケージで提供されます。このツールは **vmstat** と似ていますが、ブロックデバイスの I/O を監視する目的でのみ使用されます。また、より詳細な出力と統計も提供します。
システム I/O を監視するには、次のコマンドを実行します。

```
$ iostat
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          2.05    0.46    1.55    0.26    0.00   95.67

Device     tps     kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
nvme0n1    53.54     899.48     616.99      3445229     2363196
dm-0       42.84     753.72     238.71      2886921      914296
dm-1        0.03       0.60       0.00         2292           0
dm-2       24.15     143.12     379.80       548193     1454712
```

blktrace

これは、I/O サブシステム内で費やされた時間配分に関する詳細情報を提供します。
この情報を人間が判読できる形式で表示するには、以下のコマンドを実行します。

```
# blktrace -d /dev/dm-0 -o - | blkparse -i -

253,0   1    1   0.000000000  17694  Q   W 76423384 + 8 [kworker/u16:1]
253,0   2    1   0.001926913     0   C   W 76423384 + 8 [0]
[...]
```

ここでは、最初の列の **253,0** は、デバイスのメジャータプルおよびマイナータプルになります。2番目の列 (**1**) には、CPU に関する情報が記載されています。次に、IO プロセスを実行するプロセスのタイムスタンプと PID の列が記載されています。

6 番目の列 **Q** はイベントタイプを示し、7 番目の列 **W** は書き込み操作を示し、8 番目の列 **76423384** はブロック番号であり、**+ 8**は要求されたブロックの数になります。

最後のフィールドの [kworker/u16:1] はプロセスの名前です。

初期設定では、プロセスが明示的に強制終了されるまで、**blktrace** は永続的に実行されます。**-w** オプションを使用して、ランタイム期間を指定します。

turbostat

これは、**kernel-tools** により提供されます。x86-64 プロセッサーで、プロセッサーのトポロジー、周波数、アイドル電力状態の統計、温度、および電力使用量を報告します。
この概要を表示するには、次のコマンドを実行します。

```
# turbostat

CPUID(0): GenuineIntel 0x16 CPUID levels; 0x80000008 xlevels; family:model:stepping 0x6:8e:a
(6:142:10)
CPUID(1): SSE3 MONITOR SMX EIST TM2 TSC MSR ACPI-TM HT TM
CPUID(6): APERF, TURBO, DTS, PTM, HWP, HWPnotify, HWPwindow, HWPepp, No-HWPpkg,
EPB
[...]
```

デフォルトでは、**tervostat** は、画面全体のカウンター結果の概要を出力し、その後に 5 秒間隔でカウンター結果を出力します。**-i** オプションでカウンター結果の間隔を指定します。たとえば、**turbostat -i 10** を実行して、10 秒間隔で結果を出力します。

Turbostat は、電力使用率またはアイドル時間に関して非効率的なサーバーを識別するのにも役立ちます。また、発生しているシステム管理割り込み (SMI) の比率を特定する場合にも便利です。また、これを使用して、電源管理の調整の効果を検証することもできます。

cpupower

IT は、プロセッサーの省電力関連機能を検証および調整するツール群です。**frequency-info**、**frequency-set**、**idle-info**、**idle-set**、**set**、**info**、および **monitor** オプションを指定して **cpupower** コマンドを使用し、プロセッサー関連の値を表示および設定します。
たとえば、利用可能な cpufreq ガバナーを表示するには、次のコマンドを実行します。

```
$ cpupower frequency-info --governors
analyzing CPU 0:
  available cpufreq governors: performance powersave
```

cpupower の詳細は、Viewing CPU related information を参照してください。

GNOME Power Manager

これは、GNOME デスクトップ環境にインストールされるデーモンです。GNOME Power Manager は、システムの電源ステータスの変更 (バッテリーから AC 電源への変更など) を通知します。また、バッテリーのステータスを報告し、バッテリー残量が少なくなると警告を表示します。

関連情報

- man ページの
  powertop(1)、**diskdevstat(8)**、**netdevstat(8)**、**tuned(8)**、**vmstat(8)**、**iostat(1)**、**blktrace(8)**、**blkparse(8)**、および **turbostat(8)**
- man ページの **cpupower(1)**、**cpupower-set(1)**、**cpupower-info(1)**、**cpupower-idle(1)**、**cpupower-frequency-set(1)**、**cpupower-frequency-info(1)**、および **cpupower-monitor(1)**

第16章 POWERTOP を使用した電力消費の管理

システム管理者であれば、PowerTOP ツールを使用して、消費電力を解析および管理できます。

16.1. POWERTOP の目的

PowerTOP は、消費電力に関連する問題を診断し、バッテリーの寿命を延ばす方法について提案を示すプログラムです。

PowerTOP ツールは、システムの総電力使用量の想定と、各プロセス、デバイス、カーネルワーカー、タイマー、および割り込みハンドラーの個々の電力使用量の想定を示すことができます。このツールは、CPU を頻繁にウェイクアップするカーネルおよびユーザー空間アプリケーションの特定のコンポーネントを識別することもできます。

Red Hat Enterprise Linux 8 は、 PowerTOP のバージョン 2.x を使用します。

16.2. POWERTOP の使用

前提条件

- PowerTOP を使用できるようにするには、**powertop** パッケージがシステムにインストールされていることを確認してください。

```
# yum install powertop
```

16.2.1. PowerTOP の起動

手順

- PowerTOP を実行するには、次のコマンドを使用します。

```
# powertop
```

重要

powertop コマンドの実行時には、ラップトップはバッテリー電源で動作します。

16.2.2. PowerTOP の調整

手順

1. ラップトップでは、次のコマンドを実行して電力予測エンジンを調整することができます。

```
# powertop --calibrate
```

2. プロセス中にマシンと対話せずに、調整を終了させます。
プロセスがさまざまなテストを実行し、輝度を切り替え、デバイスのオンとオフを切り替える操作を繰り返すため、調整には時間がかかります。

3. 調整プロセスが完了すると、PowerTOP が通常どおり起動します。データを収集するために約 1 時間実行します。

十分なデータが収集されると、出力テーブルの最初の列に電力予測マークが表示されます。

注記

powertop --calibrate は、ノートパソコンでのみ使用できることに注意してください。

16.2.3. 測定間隔の設定

デフォルトでは、PowerTOP は、20 秒間隔で測定します。

この測定頻度を変更する場合は、以下の手順に従います。

手順

- **--time** オプションを指定して **powertop** コマンドを実行します。

```
# powertop --time=time in seconds
```

16.2.4. 関連情報

PowerTOP の使い方の詳細は、man ページの **powertop** を参照してください。

16.3. POWERTOP の統計

PowerTOP は、実行中に、システムから統計を収集します。

PowerTOP の出力には、複数のタブがあります。

- **Overview**
- **idle stats**
- **Frequency stats**
- **Device stats**
- **Tunables**
- **WakeUp**

Tab キーおよび **Shift+ Tab** キーを使用して、このタブを順番に切り替えることができます。

16.3.1. Overview タブ

Overview タブでは、ウェイクアップを最も頻繁に CPU に送信するか、最も電力を消費するコンポーネントのリストを表示できます。プロセス、割り込み、デバイス、その他のリソースなど、**Overview** タブの項目は、使用率に従って並べ替えられます。

Overview タブで隣接する列は、以下の情報を提供します。

Usage
: リソース使用の電力想定。

Events/s

1秒あたりウェイクアップ。1秒あたりのウェイクアップ数は、サービスまたはデバイス、ならびにカーネルのドライバーがいかに効率的に実行しているかを示します。ウェイクアップが少ないほど、消費電力が少なくなります。コンポーネントは、電力使用率をどの程度まで最適化できるかによって順序付けられます。

Category
: プロセス、デバイス、タイマーなどのコンポーネントの分類。

説明
: コンポーネントの説明。

適切に調整すると、最初の列にリストされているすべての項目に対する電力消費予測も表示されます。

これとは別に、**Overview** タブには、次のようなサマリー統計の行が含まれます。

- 合計電力消費
- バッテリーの残り寿命 (該当する場合)
- 1秒あたりの合計ウェイクアップ数、1秒あたり GPU 操作数、および1秒あたりの仮想ファイルシステム操作数の概要

16.3.2. Idle stats タブ

Idle stats タブには、すべてのプロセッサーおよびコアに対する C 状態の使用率が表示されます。**Frequency stats** タブには、(該当する場合は) すべてのプロセッサーおよびコアに対する Turbo モードを含む P 状態の使用率が表示されます。C または P 状態の長さは、CPU 使用率がどの程度最適化されているかを示します。CPU の C 状態または P 状態が高いままになるほど (C4 が C3 よりも高くなるなど)、CPU 使用率がより最適化されます。システムがアイドル状態の時に、最高の C 状態または P 状態の常駐が 90% 以上になることが理想的と言えます。

16.3.3. Device stats タブ

Device stats タブは **Overview** タブと同様の情報を提供しますが、デバイス専用です。

16.3.4. Tunables タブ

Tunables タブには、低消費電力にシステムを最適化するための、**PowerTOP** の推奨事項が含まれます。

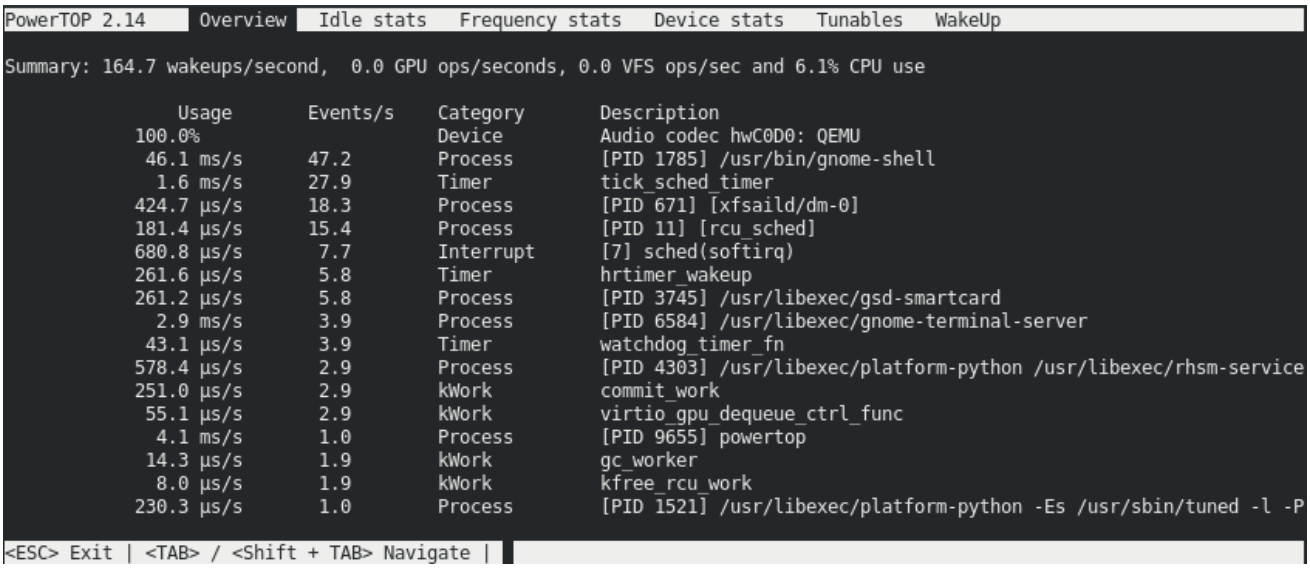
up キーおよび **down** キーを使用して提案を移動し、**enter** キーを使用して提案をオンまたはオフにします。

16.3.5. WakeUp タブ

WakeUp タブには、ユーザーが必要に応じて変更できるデバイスウェイクアップ設定が表示されます。

up キーおよび **down** キーを使用して利用可能な設定を移動し、**enter** キーを使用して設定を有効または無効にします。

図16.1 PowerTOP 出力



関連情報

PowerTOP の詳細は、PowerTOP のホームページ を参照してください。

16.4. POWERTOP で FREQUENCY STATS に値が表示されない場合がある理由

Intel P-State ドライバーを使用している場合、PowerTOP はドライバーがパッシブモードの場合にのみ **Frequency Stats** タブに値を表示します。しかし、この場合でも、値が不完全な場合があります。

Intel P-State ドライバーには、全部で 3 つのモードがあります。

- ハードウェア P-State(HWP) によるアクティブモード
- HWP なしのアクティブモード
- パッシブモード

ACPI CPUfreq ドライバーに切り替えると、PowerTOP で完全な情報が表示されます。ただし、システムをデフォルト設定にしておくことを推奨します。

どのドライバーがどのようなモードで読み込まれているかを確認するには、次のコマンドを実行します。

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_driver
```

- **intel_pstate** は、Intel P-State ドライバーが読み込まれ、アクティブモードになっている場合に返されます。
- **intel_cpufreq** は、インテル P-State ドライバーが読み込まれ、パッシブモードになっている場合に返されます。
- ACPI CPUfreq ドライバーが読み込まれている場合は、**acpi-cpufreq** が返されます。

Intel P-State ドライバーを使用している場合は、カーネルブートコマンドラインに以下の引数を追加して、ドライバーをパッシブモードで実行するようにします。

```
intel_pstate=passive
```

Intel P-State ドライバーを無効にして、代わりに ACPI CPUfreq ドライバーを使用するには、カーネルブートコマンドラインに次の引数を追加します。

```
intel_pstate=disable
```

16.5. HTML 出力の生成

端末の **powertop** の出力結果以外にも、HTML レポートを生成することもできます。

手順

- **--html** オプションを指定して **powertop** コマンドを実行します。

  ```
  # powertop --html=htmlfile.html
  ```

  htmlfile.html パラメーターを、出力ファイルに必要な名前に置き換えます。

16.6. 電力消費の最適化

電力消費を最適化するには、**powertop** サービスまたは **powertop2tuned** ユーティリティーを使用できます。

16.6.1. powertop サービスで消費電力の最適化

powertop サービスを使用すると、システムの起動の **Tunables** タブから、すべての PowerTOP の提案を自動的に有効にできます。

手順

- **powertop** サービスを有効にします。

  ```
  # systemctl enable powertop
  ```

16.6.2. powertop2tuned ユーティリティー

powertop2tuned ユーティリティーを使用すると、PowerTOPの提案からカスタムTuneDプロファイルを作成できます。

デフォルトでは、**powertop2tuned** は、**/etc/tuned/** ディレクトリーにプロファイルを作成し、現在選択している TuneD プロファイルを基にしてカスタムプロファイルを作成します。安全上の理由から、すべての PowerTOP チューニングは最初に新しいプロファイルで無効になっています。

チューニングを有効にするには、以下を行います。

- **/etc/tuned/profile_name/tuned.conf** ファイルでコメントを解除します。
- **--enable** オプションまたは **-e** オプションを使用して、PowerTOP により提案されたチューニングのほとんどを可能にする新しいプロファイルを生成します。
  USB 自動サスペンドなど、既知の問題のある特定のチューニングはデフォルトで無効になっているため、手動でコメントを解除する必要があります。

16.6.3. powertop2tuned ユーティリティーで電力消費の最適化

前提条件

- **powertop2tuned** ユーティリティーがシステムにインストールされている場合は、次のコマンドを実行します。

```
# yum install tuned-utils
```

手順

1. カスタムプロファイルを作成するには、次のコマンドを使用します。

```
# powertop2tuned new_profile_name
```

2. 新しいプロファイルをアクティベートします。

```
# tuned-adm profile new_profile_name
```

関連情報

- **powertop2tuned** に対応しているオプションの完全リストを表示するには、以下を使用します。

```
$ powertop2tuned --help
```

16.6.4. powertop.service と powertop2tuned の比較

以下の理由により、**powertop2tuned** を使用した電力消費の最適化は、**powertop.service** よりも推奨されます。

- **powertop2tuned** ユーティリティーは、PowerTOP を TuneD に統合したものです。これにより、両方のツールの利点を活かすことができます。
- **powertop2tuned** ユーティリティーを使用すると、有効になっているチューニングをきめ細かく制御できます。
- **powertop2tuned** を使用すると、潜在的に危険なチューニングは自動的に有効になりません。
- **powertop2tuned** を使用すると、再起動せずにロールバックを行うことができます。

第17章 CPU 周波数を調整してエネルギー消費を最適化

必要な CPUfreq ガバナーをセットアップした後、利用可能な **cpupower** コマンドを使用して、システムの CPU 速度を要件に応じて設定すると、システムの電力消費を最適化できます。

17.1. 対応している CPUPOWER ツールコマンド

cpupower ツールは、プロセッサーの省電力関連機能を調べ、調整するツールの集合です。

cpupower ツールは、以下のコマンドに対応します。

idle-info

cpupower idle-info コマンドを使用して、CPU アイドルドライバーで利用可能なアイドル状態と、そのほかの統計情報を表示します。詳細は、CPU Idle States を参照してください。

idle-set

cpupower idle-set コマンドを root で実行して、CPU アイドル状態を有効または無効にします。特定の CPU アイドル状態を **-d** を使用して無効にし、**-e** を使用して有効にします。

frequency-info

cpupower frequency-info コマンドを使用して、現在の **cpufreq** ドライバーと、利用可能な **cpufreq** ガバナーを表示します。詳細は、CPUfreq drivers、Core CPUfreq Governors、および Intel P-state CPUfreq governors を参照してください。

frequency-set

root で **cpupower frequency-set** コマンドを使用し、**cpufreq** とガバナーを設定します。詳細は、Setting up CPUfreq governor を参照してください。

set

root で **cpupower set** コマンドを実行して、プロセッサーの省電力ポリシーを設定します。**--perf-bias** オプションを使用すると、対応している Intel プロセッサーでソフトウェアを有効にして、最適なパフォーマンスと省電力のバランスを判断できます。割り当てる値は **0** から **15** まであり、**0** はパフォーマンスを最適化し、**15** は電力効率を最適化します。デフォルトでは、**--perf-bias** はすべてのコアに適用されます。個々のコアにのみ適用するには、**--cpu cpulist** を追加します。

info

cpupower set コマンドで有効にしたプロセッサー電源関連の設定およびハードウェア設定を表示します。たとえば、**--perf-bias** を **5** として割り当てるとします。

```
# cpupower set --perf-bias 5
# cpupower info
analyzing CPU 0:
perf-bias: 5
```

monitor

cpupower monitor コマンドを使用して、アイドル状態の統計情報と、CPU 要求を表示します。

```
# cpupower monitor
 | Nehalem       || Mperf    ||Idle_Stats
 CPU| C3   | C6   | PC3  | PC6  || C0   | Cx   | Freq || POLL | C1   | C1E  | C3   | C6   | C7s  | C8   |
C9   | C10
   0|  1.95| 55.12|  0.00|  0.00||  4.21| 95.79|  3875||  0.00|  0.68|  2.07|  3.39| 88.77|  0.00|  0.00|
0.00| 0.00
[...]
```

-l オプションを使用すると、システムで利用可能なモニターのリストを表示し、**-m** オプションを使用して、特定のモニターに関する情報を表示することができます。たとえば、**Mperf** モニターに関連する情報を監視する場合は、root で **cpupower monitor -m Mperf** コマンドを実行します。

関連情報

- man ページの **cpupower(1)**、**cpupower-idle-info(1)**、**cpupower-idle-set(1)**、**cpupower-frequency-set(1)**、**cpupower-frequency-info(1)**、**cpupower-set(1)**、**cpupower-info(1)**、および **cpupower-monitor(1)**

17.2. CPU アイドル状態

x86 アーキテクチャーを備えた CPU は、CPU の一部が非アクティブ化されている、または C-state と呼ばれるパフォーマンスの低い設定を使用しているなど、さまざまな状態をサポートしています。

この状態では、使用されていない CPU を部分的に非アクティブにすることで、電力を節約できます。ガバナーを必要とし、望ましくない電源やパフォーマンスの問題を回避するように設定される P-state とは異なり、C -state を設定する必要はありません。C-state には C0 から順に番号が付けられ、番号が大きいほど CPU 機能が低下し、省電力が大きくなります。指定された数の C-state はプロセッサー間でほぼ同じですが、状態の特定の機能セットの詳細はプロセッサーファミリーごとに異なります。C-states 0-3 は、以下のように定義されます。

C0

この状態では、CPU は動作しており、アイドル状態ではありません。

C1, Halt

この状態では、プロセッサーは命令を実行していませんが、通常は低消費電力状態ではありません。CPU は事実上遅延なしで処理を継続できます。C-state を提供するすべてのプロセッサーが、この状態に対応する必要があります。Pentium 4 プロセッサーは、C1E と呼ばれる拡張された C1 状態に対応しています。これは、低消費電力を実現する状態です。

C2, Stop-Clock

この状態では、クロックはこのプロセッサーでフリーズしますが、レジスターとキャッシュの完全な状態を保持するため、クロックを再起動するとすぐに処理を再開できます。この状態はオプションになります。

C3, Sleep

この状態では、プロセッサーはスリープ状態になり、キャッシュを最新の状態に保つ必要はありません。このため、この状態からのウェイクアップには、C2 状態からのウェイクアップよりもはるかに時間がかかります。この状態はオプションになります。

以下のコマンドを使用すると、CPUidle ドライバーで利用可能なアイドル状態と、その他の統計を表示できます。

```
$ cpupower idle-info
CPUidle governor: menu
analyzing CPU 0:

Number of idle states: 9
Available idle states: POLL C1 C1E C3 C6 C7s C8 C9 C10
[...]
```

"Nehalem" マイクロアーキテクチャーを持つ Intel CPU は、C6 状態を特徴とします。これにより、CPU の電圧供給をゼロに減らすことができますが、通常は、消費電力を 80% から 90% まで減らします。Red Hat Enterprise Linux 8 のカーネルには、この新しい C-state の最適化が含まれます。

関連情報

- man ページの **cpupower(1)** および **cpupower-idle(1)**

17.3. CPUFREQ の概要

システムの消費電力と熱出力を低減する最も効果的な方法の 1 つが CPUfreq です。これは、Red Hat Enterprise Linux 8 の x86 アーキテクチャーおよび ARM64 アーキテクチャーで対応しています。CPUfreq は CPU 速度スケーリングとも呼ばれ、Linux カーネルのインフラストラクチャーで、電力を節約するために CPU 周波数をスケーリングできます。

CPU スケーリングは、Advanced Configuration and Power Interface (ACPI) イベントに応じて、またはユーザー空間プログラムにより手動でシステムの負荷に応じて自動的に行われるため、プロセッサーのクロック速度を即座に調整できます。これにより、システムは減速したクロック速度で実行でき、電力を節約できます。周波数のシフトに関するルール (クロック速度の高速化または低速化、および周波数のシフト) は、CPUfreq ガバナーで定義されています。

root で **cpupower frequency-info** コマンドを使用すると、**cpufreq** の情報を表示できます。

17.3.1. CPUfreq ドライバー

root で **cpupower frequency-info --driver** コマンドを使用すると、現在の CPUfreq ドライバーを表示できます。

使用可能な CPUfreq 用のドライバーは、以下の 2 つです。

ACPI CPUfreq

Advanced Configuration and Power Interface (ACPI) の CPUfreq ドライバーは、ACPI を介して特定の CPU の周波数を制御するカーネルドライバーです。これにより、カーネルとハードウェア間の通信が保証されます。

Intel P-state

Red Hat Enterprise Linux 8 では、Intel P-state ドライバーに対応しています。このドライバーは、Intel Xeon E シリーズアーキテクチャーまたは新しいアーキテクチャーに基づくプロセッサーで、P-state 選択を制御するインターフェイスを提供します。
現在、Intel P-state は、対応している CPU にデフォルトで使用されています。**intel_pstate=disable** コマンドをカーネルコマンドラインに追加すると、ACPI CPUfreq の使用に切り替えることができます。

Intel P-state は、**setpolicy()** コールバックを実装します。ドライバーは、**cpufreq** コアから要求されたポリシーに基づいて、使用する P-state を決定します。プロセッサーが次の P-state を内部で選択できる場合、ドライバーはこの責任をプロセッサーにオフロードします。そうでない場合は、次の P-state を選択するアルゴリズムがドライバーに実装されます。

Intel P-state は、P-state の選択を制御する独自の **sysfs** ファイルを提供します。これらのファイルは、**/sys/devices/system/cpu/intel_pstate/** ディレクトリーにあります。ファイルに加えた変更は、すべての CPU に適用されます。

このディレクトリーには、P-state パラメーターの設定に使用される以下のファイルが含まれます。

- **max_perf_pct** は、ドライバーによって要求される最大 P-state を制限します。これは、使用可能なパフォーマンスのパーセンテージで表されます。利用可能な P-state パフォーマンスは、**no_turbo** 設定により削減できます。

- **min_perf_pct** は、ドライバーによって要求される最小の P-state を制限します。これは、最大の **no-turbo** パフォーマンスレベルのパーセンテージで表されます。

- **no_turbo** は、ドライバーを、ターボ周波数レンジの下にある P-state を選択するように制限します。
- **turbo_pct** は、対応しているハードウェアのパフォーマンス合計のうち、ターボ領域にあるものの割合を表示します。この数字は、**turbo** が無効になっているかどうかに関係ありません。
- **num_pstates** は、ハードウェアで対応している P-state の数を表示します。この数は、ターボが無効になっているかどうかに関係ありません。

関連情報

- man ページの **cpupower-frequency-info(1)**

17.3.2. コア CPUfreq ガバナー

CPUfreq ガバナーは、システム CPU の電源特性を定義します。これは、CPU パフォーマンスに影響を及ぼします。各ガバナーには、ワークロードに関する固有の動作、目的、および適合性があります。**cpupower frequency-info --governor** コマンドを root で実行すると、利用可能な CPUfreq ガバナーを表示できます。

Red Hat Enterprise Linux 8 には、複数のコア CPUfreq ガバナーが同梱されています。

cpufreq_performance

CPU は、可能な限り最も高いクロック周波数を使用するように強制されています。この周波数は静的に設定され、変更されません。このため、この特定のガバナーでは省電力の利点はありません。これは、ワークロードが重い時間帯にのみ適しており、CPU がめったにアイドル状態にならないか、まったくアイドル状態にならない時間帯にのみ適しています。

cpufreq_powersave

CPU は、可能な限り最低のクロック周波数を使用するように強制されています。この周波数は静的に設定され、変更されません。このガバナーを使用すると、最大限の電力を削減できますが、CPU パフォーマンスは低くなります。ただし、原則として、全負荷時の低速 CPU は、負荷がかかっていない高速 CPU よりも多くの電力を消費するため、"powersave" という用語は誤解を招く場合があります。したがって、予想される低アクティビティー時に **powersave** ガバナーを使用するように CPU を設定することが推奨されますが、その間に予想外の高負荷が発生すると、システムが実際により多くの電力を消費する可能性があります。powersave ガバナーは、省電力というよりも CPU の速度リミッターです。これは、システムや、過熱が問題になる可能性がある環境で最も役立ちます。

cpufreq_ondemand

これは動的ガバナーで、システムの負荷が高い場合には CPU を有効にして最大クロック周波数を実現し、システムがアイドル状態の場合には最小クロック周波数を実現できます。これにより、システムはシステム負荷に応じて消費電力を調整できますが、周波数切り替えの間の待ち時間はかかります。このため、システムがアイドル状態と高負荷のワークロードを頻繁に切り替える場合、レイテンシーは、**ondemand** ガバナーが提供するパフォーマンスや省電力の利点を相殺することができます。ほとんどのシステムでは、**ondemand** ガバナーにより、放熱、電力消耗、性能、および管理可能性について最適な妥協点を見つけることができます。システムが 1 日の特定の時間にのみビジー状態の場合、**ondemand** ガバナーは、それ以上の介入なしに、負荷に応じて最大周波数と最小周波数を自動的に切り替えます。

cpufreq_userspace

これにより、ユーザー空間プログラムや root で実行しているプロセスが、周波数を設定できます。すべてのガバナーの中で、**userspace** は最もカスタマイズ可能で、設定方法に応じて、システムのパフォーマンスと消費の最適なバランスを実現できます。

cpufreq_conservative

ondemand ガバナーと同様に、**conservative** ガバナーも用途に合わせてクロック周波数を調整します。ただし、**conservative** ガバナーは、周波数を徐々に切り替えます。つまり、**conservative** ガバナーは、単に最大/最小を選択するのではなく、負荷に対して最善と思われるクロック周波数に調整されます。これにより、消費電力を大幅に節約できる可能性がありますが、**ondemand** ガバナーよりもはるかに長い遅延が発生します。

注記

cron ジョブを使用して、ガバナーを有効にできます。これにより、指定した時間帯に特定のガバナーを自動的に設定できます。このため、勤務時間後など、アイドル時間帯には低周波ガバナーを指定し、作業負荷が高い時間帯には高周波ガバナーに戻すことができます。

指定したガバナーを有効にする手順については、Setting up CPUfreq governor を参照してください。

17.3.3. Intel P-state の CPUfreq ガバナー

デフォルトで、Intel P-state ドライバーは、CPU が HWP に対応しているかどうかに応じて、Hardware p-state (HWP) の有無にかかわらずアクティブモードで動作します。

cpupower frequency-info --governor コマンドを root で実行すると、利用可能な CPUfreq ガバナーを表示できます。

注記

performance および **powersave** Intel P-state CPUfreq ガバナーの機能は、同じ名前のコア CPUfreq ガバナーと比較されます。

Intel P-state ドライバーは、以下の 3 つの異なるモードで動作できます。

Active mode with hardware-managed P-states

HWP でアクティブモードが使用されている場合、Intel P-state ドライバーは、P-state 選択を実行するように CPU に指示します。ドライバーは、周波数のヒントを提供できます。ただし、最終的な選択は CPU の内部ロジックによって異なります。HWP でアクティブモードにすると、Intel P-state ドライバーにより、2 つの P-state 選択アルゴリズムが提供されます。

- **performance**: **performance** ガバナーを使用すると、ドライバーは内部 CPU ロジックにパフォーマンス指向になるように指示します。P-state の範囲は、ドライバーが使用できる範囲の上限に制限されます。
- **powersave**: **powersave** ガバナーを使用すると、ドライバーは、内部 CPU ロジックに省電力指向になるように指示します。

Active mode without hardware-managed P-states

HWP を使用しないアクティブモードの場合、Intel P-state ドライバーは次の 2 つの P-state 選択アルゴリズムを提供します。

- **performance**: **performance** ガバナーを使用すると、ドライバーは使用できる最大の P-state を選択します。
- **powersave**: **powersave** ガバナーを使用すると、ドライバーは、現在の CPU 使用率に比例する P-state を選択します。この動作は、**ondemand** CPUfreq コアガバナーに似ています。

パッシブモード

passive モードを使用すると、Intel P-state ドライバーは、従来の CPUfreq スケーリングドライバーと同じように機能します。利用可能なすべての汎用 CPUFreq コアガバナーを使用できます。

17.3.4. CPUfreq ガバナーの設定

すべての CPUfreq ドライバーは **kernel-tools** パッケージに組み込まれ、自動的に選択されます。CPUfreq を設定するには、ガバナーを選択する必要があります。

前提条件

- **cpupower** を使用するには、**kernel-tools** をインストールします。

```
# yum install kernel-tools
```

手順

1. 特定の CPU で使用できるガバナーを表示します。

```
# cpupower frequency-info --governors
analyzing CPU 0:
  available cpufreq governors: performance powersave
```

2. すべての CPU で、ガバナーのいずれかを有効にします。

```
# cpupower frequency-set --governor performance
```

必要に応じて、**performance** ガバナーを、**cpufreq** ガバナー名に置き換えます。

特定のコアでガバナーのみを有効にするには、CPU 番号の範囲またはコンマ区切りのリストで **-c** を使用します。たとえば、CPU 1-3 および 5 の **userspace** ガバナーを有効にするには、次のコマンドを使用します。

```
# cpupower -c 1-3,5 frequency-set --governor cpufreq_userspace
```

注記

kernel-tools がインストールされていない場合は、**/sys/devices/system/cpu/cpuid/cpufreq/** ディレクトリーに CPUfreq 設定が表示されます。設定および値は、この調整可能パラメーターに書き込むことで変更できます。たとえば、最小クロック速度の cpu0 から 360MHz を設定するには、次のコマンドを使用します。

```
# echo 360000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

検証

- ガバナーが有効になっていることを確認します。

```
# cpupower frequency-info
analyzing CPU 0:
  driver: intel_pstate
```

```
CPUs which run at the same hardware frequency: 0
CPUs which need to have their frequency coordinated by software: 0
maximum transition latency:  Cannot determine or is not supported.
hardware limits: 400 MHz - 4.20 GHz
available cpufreq governors: performance powersave
current policy: frequency should be within 400 MHz and 4.20 GHz.
      The governor "performance" may decide which speed to use within this range.
current CPU frequency: Unable to call hardware
current CPU frequency: 3.88 GHz (asserted by call to kernel)
boost state support:
  Supported: yes
  Active: yes
```

現行ポリシーでは、直近で有効になった **cpufreq** ガバナーが表示されます。この場合、**performance** になります。

関連情報

- man ページの **cpupower-frequency-info(1)** および **cpupower-frequency-set(1)**

第18章 PERF の使用

システム管理者は、**perf** ツールを使用して、システムのパフォーマンスデータを収集および分析できます。

18.1. PERF の概要

perf ユーザー空間ツールは、カーネルベースのサブシステム **Performance Counters for Linux** (PCL) とインターフェイスします。**perf** は、PMU (Performance Monitoring Unit) を使用してさまざまなハードウェアおよびソフトウェアイベントを測定、記録、監視する強力なツールです。**perf** は tracepoint、kprobes、および uprobes にも対応しています。

18.2. PERF のインストール

この手順では、**perf** ユーザー空間ツールをインストールします。

手順

- **perf** ツールをインストールします。

```
# yum install perf
```

18.3. 一般的な PERF コマンド

perf stat

このコマンドは、実行された命令や消費したクロックサイクルなど、一般的なパフォーマンスイベントに関する全体的な統計を提供します。オプションを指定すると、デフォルトの測定イベント以外のイベントを選択できます。

perf record

このコマンドは、パフォーマンスデータをファイル **perf.data** に記録します。このファイルは後で **perf report** コマンドを使用して分析できます。

perf report

このコマンドは、**perf record** で作成された **perf.data** ファイルからパフォーマンスデータを読み取り、表示します。

perf list

このコマンドは、特定のマシンで利用可能なイベントをリスト表示します。これらのイベントは、パフォーマンス監視ハードウェアや、システムのソフトウェア設定によって異なります。

perf top

このコマンドは、**top** ユーティリティーと同様の機能を実行します。リアルタイムでパフォーマンスカウンタープロファイルを生成および表示します。

perf trace

このコマンドは、**strace** ツールと同様の機能を実行します。指定されたスレッドまたはプロセスによって使用されるシステムコールとそのアプリケーションが受信するすべてのシグナルを監視します。

perf help

このコマンドは、**perf** コマンドのリストを表示します。

関連情報

- サブコマンドに **--help** オプションを追加すると、man ページが表示されます。

第19章 PERF TOP を使用した、リアルタイムでの CPU 使用率のプロファイリング

perf top コマンドを使用して、さまざまな機能の CPU 使用率をリアルタイムで測定できます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

19.1. PERF TOP の目的

perf top コマンドは、**top** ユーティリティーと同様に、リアルタイムシステムのプロファイリングおよび機能に使用されます。ただし、**top** ユーティリティーは、通常、指定のプロセスまたはスレッドが使用している CPU 時間を示しており、**perf top** は各関数が使用する CPU 時間を表示します。デフォルトの状態では、**perf top** はユーザー空間とカーネル空間のすべての CPU で使用される関数について通知します。**perf top** を使用するには、root アクセスが必要です。

19.2. PERF TOP を使用した CPU 使用率のプロファイリング

この手順では、**perf top** をアクティブにし、CPU 使用率をリアルタイムでプロファイルします。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- root アクセスがある。

手順

- **perf top** モニタリングインターフェイスを起動します。

```
# perf top
```

監視インターフェイスは、以下のようになります。

```
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
```

この例では、カーネル機能の **do_syscall_64** が最も多くの CPU 時間を使用しています。

関連情報

- man ページの **perf-top(1)**

19.3. PERF TOP 出力の解釈

perf top 監視インターフェイスでは、データが以下のようなさまざまな列で表示されます。

Overhead 列

指定された関数が使用している CPU のパーセントを表示します。

共有オブジェクトのコラム

機能を使用しているプログラムまたはライブラリー名を表示します。

Symbol 列

関数名またはシンボルを表示します。カーネル空間で実行される関数は **[k]** によって識別され、ユーザースペースで実行される関数は **[.]** によって識別されます。

19.4. PERF が一部の関数名を RAW 関数アドレスとして表示する理由

カーネル関数の場合は、**perf** が **/proc/kallsyms** ファイルからの情報を使用して、サンプルをそれぞれの関数名またはシンボルにマッピングします。ただし、ユーザー空間で実行される関数については、バイナリーがストライピングされるので、raw 機能のアドレスが表示される可能性があります。

実行ファイルの **debuginfo** パッケージがインストールされているか、実行ファイルがローカルで開発したアプリケーションである場合は、アプリケーションがデバッグ情報 (GCC の **-g** オプション) を有効にしてコンパイルされ、このような状況で関数名またはシンボルが表示される必要があります。

注記

実行ファイルに関連付けられた **debuginfo** をインストールした後に、**perf record** コマンドを再実行する必要はありません。単に **perf report** を再実行してください。

関連情報

- デバッグ情報を使用したデバッグの有効化

19.5. デバッグおよびソースのリポジトリーの有効化

Red Hat Enterprise Linux の標準インストールでは、デバッグリポジトリーおよびソースリポジトリーが有効になっていません。このリポジトリーには、システムコンポーネントのデバッグとパフォーマンスの測定に必要な情報が含まれます。

手順

- ソースおよびデバッグの情報パッケージチャンネルを有効にします。

```
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms
```

$(uname -i) の部分は、システムのアーキテクチャーで一致する値に自動的に置き換えられます。

アーキテクチャー名	値
64 ビット Intel および AMD	x86_64
64 ビット ARM	aarch64
IBM POWER	ppc64le
64 ビット IBM Z	s390x

19.6. GDB を使用したアプリケーションまたはライブラリーの DEBUGINFO パッケージの取得

デバッグ情報は、コードをデバッグするために必要です。パッケージからインストールされるコードの場合、GNU デバッガー (GDB) は足りないデバッグ情報を自動的に認識し、パッケージ名を解決し、パッケージの取得方法に関する具体的なアドバイスを提供します。

前提条件

- デバッグするアプリケーションまたはライブラリーがシステムにインストールされている。
- GDB と **debuginfo-install** ツールがシステムにインストールされている。詳細は アプリケーションをデバッグするための設定 を参照してください。
- **debuginfo** および **debugsource** パッケージを提供するリポジトリーを設定し、システムで有効にしている。詳細は、デバッグおよびソースリポジトリーの有効化 を参照してください。

手順

1. デバッグするアプリケーションまたはライブラリーに割り当てられた GDB を起動します。GDB は、足りないデバッグ情報を自動的に認識し、実行するコマンドを提案します。

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. GDB を終了します。**q** と入力して、**Enter** で確認します。

```
(gdb) q
```

3. GDB が提案するコマンドを実行して、必要な **debuginfo** パッケージをインストールします。

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

dnf パッケージ管理ツールは、変更の概要を提供し、確認を求め、確認後に必要なファイルをすべてダウンロードしてインストールします。

4. GDB が **debuginfo** パッケージを提案できない場合は、手動でのアプリケーションまたはライブラリーの debuginfo パッケージの取得 で説明されている手順に従います。

関連情報

- Red Hat ナレッジベースソリューション How can I download or install debuginfo packages for RHEL systems?

第20章 PERF STAT を使用したプロセス実行中のイベントのカウント

perf stat コマンドを使用すると、プロセスの実行中にハードウェアおよびソフトウェアのイベントをカウントできます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

20.1. PERF STAT の目的

perf stat コマンドは指定されたコマンドを実行し、コマンドの実行中にハードウェアおよびソフトウェアのイベントの発生回数を維持し、これらのカウントの統計を生成します。イベントを指定しないと、**perf stat** は共通のハードウェアおよびソフトウェアのイベントセットをカウントします。

20.2. PERF STAT を使用したイベントのカウント

perf stat を使用すると、コマンドの実行中に発生したハードウェアおよびソフトウェアのイベントをカウントし、これらのカウントの統計を生成できます。デフォルトでは、**perf stat** はスレッドごとのモードで動作します。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

- イベントをカウントします。
  - root アクセスなしで **perf stat** コマンドを実行すると、ユーザー空間で発生したイベントのみをカウントします。

```
$ perf stat ls
```

例20.1 perf stat の出力が root アクセスなしで実行

```
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
```

以前の例で分かるように、**perf stat** を root アクセスなしで実行すると、イベント名の後に **:u** が付けられ、これらのイベントがユーザー空間でのみカウントされていることが分かります。

- ユーザー空間およびカーネルスペースの両方のイベントをカウントするには、**perf stat** の実行時に root アクセスが必要になります。

```
# perf stat ls
```

例20.2 root アクセスで実行された perf stat の出力

```
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
```

- デフォルトでは、**perf stat** はスレッドごとのモードで動作します。CPU 全体のイベントカウントに変更するには、**-a** オプションを **perf stat** に渡します。CPU 全体のイベントをカウントするには、root アクセスが必要です。

```
# perf stat -a ls
```

関連情報

- man ページの **perf-stat(1)**

20.3. PERF STAT 出力の解釈

perf stat は指定されたコマンドを実行し、コマンドの実行中にイベントの発生をカウントし、これらのカウントの統計を 3 列で表示します。

1. 指定されたイベントでカウントされた発生数
2. カウントされたイベントの名前。

3. 関連するメトリックが利用可能な場合、右側のコラムのハッシュ記号 (**#**) の後に比率またはパーセンテージが表示されます。
   たとえば、デフォルトモードで実行している場合、**perf stat** はサイクルと命令の両方をカウントします。したがって、右端のコラムのサイクルごとの命令を計算して表示します。デフォルトでは両方のイベントがカウントされるため、分岐ミスに関して同様の動作がすべてのブランチのパーセントとして表示されます。

20.4. 実行中のプロセスに PERF STAT を割り当てる

perf stat を実行中のプロセスに割り当てることができます。これにより、コマンドの実行中に、指定したプロセスでのみ発生するイベントをカウントするように **perf stat** に指示します。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

- **perf stat** を実行中のプロセスに割り当てます。

  ```
  $ perf stat -p ID1,ID2 sleep seconds
  ```

  前の例では、ID が **ID1** and **ID2** のプロセス内のイベントを、**sleep** コマンドで指定した **seconds** の秒数だけカウントします。

関連情報

- man ページの **perf-stat(1)**

第21章 PERF によるパフォーマンスプロファイルの記録および分析

perf ツールを使用すると、パフォーマンスデータを記録し、後で分析することができます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

21.1. PERF RECORD の目的

perf record コマンドはパフォーマンスデータのサンプルを収集し、**perf.data** ファイルに保存して、他の **perf** コマンドで読み込み、視覚化できるようにします。**perf.data** は現在のディレクトリーに生成され、後で別のマシンからアクセスできます。

perf record を記録するコマンドを指定しないと、**Ctrl+C** を押して手動でプロセスを停止するまで記録されます。**-p** オプションに続いてプロセス ID を 1 つ以上渡すと、**perf record** を特定のプロセスに割り当てることができます。root アクセスなしで **perf record** レコードを実行できますが、実行するとユーザー領域のパフォーマンスデータのサンプルのみとなります。デフォルトモードでは、**perf record** レコードは CPU サイクルをサンプルイベントとして使用し、継承モードが有効な状態でスレッドごとのモードで動作します。

21.2. ROOT アクセスなしのパフォーマンスプロファイルの記録

root アクセスなしで **perf record** を使用すると、ユーザー空間のみのパフォーマンスデータのサンプリングおよび記録を行うことができます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

- パフォーマンスデータのサンプルと記録:

  ```
  $ perf record command
  ```

  command を、サンプルデータを作成するコマンドに置き換えます。コマンドを指定しないと、**Ctrl+C** を押して手動で停止するまで **perf record** がデータのサンプリングを行います。

関連情報

- man ページの **perf-record(1)**

21.3. ROOT アクセスによるパフォーマンスプロファイルの記録

root アクセスで **perf record** を使用して、ユーザー空間とカーネル空間の両方でパフォーマンスデータをサンプリングおよび記録できます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- root アクセスがある。

手順

- パフォーマンスデータのサンプルと記録:

```
# perf record command
```

command を、サンプルデータを作成するコマンドに置き換えます。コマンドを指定しないと、**Ctrl**+**C** を押して手動で停止するまで **perf record** がデータのサンプリングを行います。

関連情報

- man ページの **perf-record(1)**

21.4. CPU ごとのモードでのパフォーマンスプロファイルの記録

CPU ごとのモードで **perf record** を使用すると、監視対象の CPU のすべてのスレッドにわたって、ユーザー空間とカーネル空間の両方で同時にパフォーマンスデータをサンプリングして記録できます。デフォルトでは、CPU ごとのモードはすべてのオンライン CPU を監視します。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

- パフォーマンスデータのサンプルと記録:

```
# perf record -a command
```

command を、サンプルデータを作成するコマンドに置き換えます。コマンドを指定しないと、**Ctrl**+**C** を押して手動で停止するまで **perf record** がデータのサンプリングを行います。

関連情報

- man ページの **perf-record(1)**

21.5. PERF レコードで呼び出し先のデータを取得する

perf record ツールを設定して、どの関数がパフォーマンスプロファイル内の他の関数を呼び出しているかを記録することができます。これは、複数のプロセスが同じ関数を呼び出す場合にボトルネックを特定するのに役立ちます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

- **--call-graph** オプションを使用して、パフォーマンスデータのサンプルと記録を行います。

```
$ perf record --call-graph method command
```

  - **command** を、サンプルデータを作成するコマンドに置き換えます。コマンドを指定しないと、**Ctrl+C** を押して手動で停止するまで **perf record** がデータのサンプリングを行います。
  - method を、以下のアンワインドメソッドのいずれかに置き換えます。

    fp

    フレームポインターメソッドを使用します。GCC オプション **--fomit-frame-pointer** でビルドされたバイナリーの場合など、コンパイラーの最適化により、スタックをアンワインドできない可能性があります。

    dwarf

    DWARF 呼び出し情報を使用してスタックのアンワインドを行います。

    lbr

    Intel プロセッサーで最後のブランチレコードハードウェアを使用します。

関連情報

- man ページの **perf-record(1)**

21.6. PERF レポートを使用した PERF.DATA の分析

perf report を使用して **perf.data** ファイルを表示し、分析できます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- 現行ディレクトリーに **perf.data** ファイルがある。
- **perf.data** ファイルが root アクセスで作成された場合は、root アクセスで **perf report** を実行する必要もあります。

手順

- 詳細な分析のために **perf.data** ファイルの内容を表示します。

```
# perf report
```

このコマンドは、以下のような出力を表示します。

```
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
```

関連情報

- man ページの **perf-report(1)**

21.7. PERF REPORT 出力の解釈

perf report コマンドを実行して表示されるテーブルは、データを複数のコラムに分類します。

Overhead 列
: その特定の関数で収集されたサンプル全体の割合を示します。

Command 列
: サンプルが収集されたプロセスを通知します。

Shared Object 列
: サンプルの送信元である ELF イメージの名前を表示します (サンプルがカーネルからのものである場合に [kernel.kallsyms] という名前が使用されます)。

Symbol 列
: 関数名またはシンボルを表示します。

デフォルトモードでは、関数は、オーバーヘッドの最も高いものが最初に表示される順に降順でソートされます。

21.8. 別のデバイスで読み取り可能な PERF.DATA ファイルの生成

perf ツールを使用してパフォーマンスデータを **perf.data** ファイルに記録し、異なるデバイスで分析することができます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- カーネル **debuginfo** パッケージがインストールされている。詳細は GDB を使用したアプリケーションまたはライブラリーの debuginfo パッケージの取得 を参照してください。

手順

1. さらに調査する予定のパフォーマンスデータを取得します。

```
# perf record -a --call-graph fp sleep seconds
```

この例では、**sleep** コマンドの使用によって指定される **秒** 数でシステム全体の **perf.data** を生成します。また、フレームポインターの方法を使用して呼び出しグラフデータを取得します。

2. 記録されたデータのデバッグシンボルを含むアーカイブファイルを生成します。

```
# perf archive
```

検証手順

- アーカイブファイルが現在のアクティブなディレクトリーで生成されたことを確認します。

```
# ls perf.data*
```

出力には、**perf.data** で始まる現在のディレクトリーのすべてのファイルが表示されます。アーカイブファイルの名前は以下のいずれかになります。

```
perf.data.tar.gz
```

または、以下を実行します。

```
perf.data.tar.bz2
```

関連情報

- perf によるパフォーマンスプロファイルの記録および分析
- perf レコードで呼び出し先のデータを取得する

21.9. 別のデバイスで作成された PERF.DATA ファイルの分析

perf ツールを使用して、別のデバイスで生成された **perf.data** ファイルを分析することができます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- 使用中の現在のデバイスに、別のデバイスで生成された **perf.data** ファイルと関連アーカイブファイルが存在する。

手順

1. **perf.data** ファイルとアーカイブファイルの両方を現在のアクティブなディレクトリーにコピーします。
2. アーカイブファイルを ~/**.debug** にデプロイメントします。

```
# mkdir -p ~/.debug
# tar xf perf.data.tar.bz2 -C ~/.debug
```

注記

アーカイブファイルの名前は **perf.data.tar.gz** でも構いません。

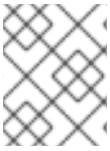
3. **perf.data** ファイルを開いて詳細な分析を行います。

```
# perf report
```

21.10. PERF が一部の関数名を RAW 関数アドレスとして表示する理由

カーネル関数の場合は、**perf** が **/proc/kallsyms** ファイルからの情報を使用して、サンプルをそれぞれの関数名またはシンボルにマッピングします。ただし、ユーザー空間で実行される関数については、バイナリーがストライピングされるので、raw 機能のアドレスが表示される可能性があります。

実行ファイルの **debuginfo** パッケージがインストールされているか、実行ファイルがローカルで開発したアプリケーションである場合は、アプリケーションがデバッグ情報 (GCC の **-g** オプション) を有効にしてコンパイルされ、このような状況で関数名またはシンボルが表示される必要があります。



注記

実行ファイルに関連付けられた **debuginfo** をインストールした後に、**perf record** コマンドを再実行する必要はありません。単に **perf report** を再実行してください。

関連情報

- デバッグ情報を使用したデバッグの有効化

21.11. デバッグおよびソースのリポジトリーの有効化

Red Hat Enterprise Linux の標準インストールでは、デバッグリポジトリーおよびソースリポジトリーが有効になっていません。このリポジトリーには、システムコンポーネントのデバッグとパフォーマンスの測定に必要な情報が含まれます。

手順

- ソースおよびデバッグの情報パッケージチャンネルを有効にします。

```
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms
```

$(uname -i) の部分は、システムのアーキテクチャーで一致する値に自動的に置き換えられます。

アーキテクチャー名	値

アーキテクチャー名	値
64 ビット Intel および AMD	x86_64
64 ビット ARM	aarch64
IBM POWER	ppc64le
64 ビット IBM Z	s390x

21.12. GDB を使用したアプリケーションまたはライブラリーの DEBUGINFO パッケージの取得

デバッグ情報は、コードをデバッグするために必要です。パッケージからインストールされるコードの場合、GNU デバッガー (GDB) は足りないデバッグ情報を自動的に認識し、パッケージ名を解決し、パッケージの取得方法に関する具体的なアドバイスを提供します。

前提条件

- デバッグするアプリケーションまたはライブラリーがシステムにインストールされている。
- GDB と **debuginfo-install** ツールがシステムにインストールされている。詳細は アプリケーションをデバッグするための設定 を参照してください。
- **debuginfo** および **debugsource** パッケージを提供するリポジトリーを設定し、システムで有効にしている。詳細は、デバッグおよびソースリポジトリーの有効化 を参照してください。

手順

1. デバッグするアプリケーションまたはライブラリーに割り当てられた GDB を起動します。GDB は、足りないデバッグ情報を自動的に認識し、実行するコマンドを提案します。

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. GDB を終了します。**q** と入力して、**Enter** で確認します。

```
(gdb) q
```

3. GDB が提案するコマンドを実行して、必要な **debuginfo** パッケージをインストールします。

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

dnf パッケージ管理ツールは、変更の概要を提供し、確認を求め、確認後に必要なファイルをすべてダウンロードしてインストールします。

4. GDB が **debuginfo** パッケージを提案できない場合は、手動でのアプリケーションまたはライブラリーの debuginfo パッケージの取得 で説明されている手順に従います。

関連情報

- Red Hat ナレッジベースソリューション How can I download or install debuginfo packages for RHEL systems?

第22章 PERF を使用したビジーな CPU の調査

システムでパフォーマンスの問題を調査する際には、**perf** ツールを使用して最もビジー状態の CPU を特定し、監視することで、作業に集中することができます。

22.1. PERF STAT でカウントされた CPU イベントの表示

perf stat を使用すると、CPU カウントアグリゲーションを無効にすることで、どの CPU イベントがカウントされたかを表示できます。この機能を使用するには、**-a** フラグを使用してシステム全体のモードでイベントをカウントする必要があります。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

- CPU カウントアグリゲーションが無効になっているイベントをカウントします。

```
# perf stat -a -A sleep seconds
```

この例では、**CPU0** 以降の各 CPU に対して **sleep** コマンドを使用し、一定時間 (**秒** 単位) に記録された一般的なハードウェアおよびソフトウェアイベントのデフォルトセット数が表示されます。そのため、cycle などのイベントを指定すると便利です。

```
# perf stat -a -A -e cycles sleep seconds
```

22.2. PERF レポートを使用して実行した CPU サンプルの表示

perf record コマンドはパフォーマンスデータをサンプルし、このデータを **perf.data** ファイルに保存します。このファイルは **perf report** コマンドで読み取ることができます。**perf record** コマンドは、どの CPU サンプルが発生したかを常に記録します。**perf report** を設定して、この情報を表示することができます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- 現行ディレクトリーに **perf record** で **perf.data** ファイルが作成されている。**perf.data** ファイルが root アクセスで作成された場合は、root アクセスで **perf report** を実行する必要もあります。

手順

- CPU でソートしながら、詳細な分析のために **perf.data** ファイルの内容を表示します。

```
# perf report --sort cpu
```

  - CPU およびコマンドでソートすると、CPU 時間が費やされている場所に関する詳細情報を表示できます。

```
# perf report --sort cpu,comm
```

この例では、すべての監視 CPU からのコマンドを、オーバーヘッド使用量の降順で合計オーバーヘッドでリスト表示し、コマンドが実行された CPU を特定します。

関連情報

- perf によるパフォーマンスプロファイルの記録および分析

22.3. PERF TOP を使用したプロファイリング中の特定の CPU の表示

perf top を設定して、システムのリアルタイムプロファイリング中に特定の CPU および相対使用率を表示できます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

- CPU でソートしながら **perf top** インターフェイスを起動します。

```
# perf top --sort cpu
```

この例では、CPU とその各オーバーヘッドを、オーバーヘッド使用量の降順にリアルタイムでリスト表示します。

  - CPU およびコマンドでソートして、CPU 時間が費やされている場所の詳細を確認できます。

```
# perf top --sort cpu,comm
```

この例では、オーバーヘッド使用量の降順で合計オーバーヘッドでコマンドをリスト表示し、そのコマンドがリアルタイムに実行された CPU を特定します。

22.4. PERF レコードと PERF レポートを使用した特定 CPU の監視

perf record は、対象の特定の CPU のみのサンプルを設定し、詳細な分析のために **perf report** で生成された **perf.data** ファイルを分析できます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

1. **perf.data** ファイルを生成して、特定の CPU のパフォーマンスデータをサンプルし、記録します。
   - CPU のコンマ区切りリストを使用します。

```
# perf record -C 0,1 sleep seconds
```

上記の例は、**sleep** コマンドの使用によって決定される **秒** 数で CPU 0 と 1 にデータをサンプルし、記録します。

- さまざまな CPU を使用:

```
# perf record -C 0-2 sleep seconds
```

上記の例は、**sleep** コマンドの使用によって決定される **秒** 数で、CPU 0 から 2 までのすべての CPU にデータをサンプルし、記録します。

2. 詳細な分析のために **perf.data** ファイルの内容を表示します。

```
# perf report
```

この例では、**perf.data** の内容を表示します。複数の CPU を監視しており、どの CPU データがサンプルされたかを把握する場合は、perf report を使用した CPU サンプルの表示 を参照してください。

第23章 PERF でアプリケーションパフォーマンスの監視

perf ツールを使用して、アプリケーションのパフォーマンスを監視および分析できます。

23.1. 実行中のプロセスに PERF レコードを割り当てる

実行中のプロセスに **perf レコード** をアタッチできます。これにより、**perf record** が、指定されたプロセスでパフォーマンスデータのサンプルデータと記録のみを行うように指示されます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

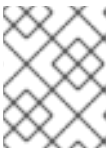
手順

- 実行中のプロセスに **perf レコード** をアタッチします。

```
$ perf record -p ID1,ID2 sleep seconds
```

上記の例では、**sleep** コマンドを使用して、プロセス ID の **ID1** と **ID2** のプロセスのパフォーマンスデータを **秒** 数でサンプルし、記録します。**perf** を設定して、イベントを特定のスレッドに記録することもできます。

```
$ perf record -t ID1,ID2 sleep seconds
```



注記

-t フラグを使用し、スレッド ID をログに記録する場合、 **perf** はデフォルトで継承を無効にします。**--inherit** オプションを追加して継承を有効にできます。

23.2. PERF レコードで呼び出し先のデータを取得する

perf record ツールを設定して、どの関数がパフォーマンスプロファイル内の他の関数を呼び出しているかを記録することができます。これは、複数のプロセスが同じ関数を呼び出す場合にボトルネックを特定するのに役立ちます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

- **--call-graph** オプションを使用して、パフォーマンスデータのサンプルと記録を行います。

```
$ perf record --call-graph method command
```

  - **command** を、サンプルデータを作成するコマンドに置き換えます。コマンドを指定しないと、**Ctrl+C** を押して手動で停止するまで **perf record** がデータのサンプリングを行います。

- **method** を、以下のアンワインドメソッドのいずれかに置き換えます。

  fp

  フレームポインターメソッドを使用します。GCC オプション **--fomit-frame-pointer** でビルドされたバイナリーの場合など、コンパイラーの最適化により、スタックをアンワインドできない可能性があります。

  dwarf

  DWARF 呼び出し情報を使用してスタックのアンワインドを行います。

  lbr

  Intel プロセッサーで最後のブランチレコードハードウェアを使用します。

関連情報

- man ページの **perf-record(1)**

23.3. PERF レポートを使用した PERF.DATA の分析

perf report を使用して **perf.data** ファイルを表示し、分析できます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- 現行ディレクトリーに **perf.data** ファイルがある。
- **perf.data** ファイルが root アクセスで作成された場合は、root アクセスで **perf report** を実行する必要もあります。

手順

- 詳細な分析のために **perf.data** ファイルの内容を表示します。

```
# perf report
```

このコマンドは、以下のような出力を表示します。

```
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
```

関連情報

- man ページの **perf-report(1)**

第24章 PERF を使用した UPROBE の作成

24.1. PERF を使用した関数レベルでのプローブの作成

perf ツールを使用すると、プロセスまたはアプリケーション内の任意の点に動的なトレースポイントを作成できます。その後、このトレースポイントを **perf stat** や **perf record** などの他の **perf** ツールと併用すると、プロセスやアプリケーションの動作をよりよく理解できるようになります。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

1. プロセスまたはアプリケーション内の対象の場所で、監視対象のプロセスまたはアプリケーションに uprobe を作成します。

```
# 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
```

関連情報

- man ページの **perf-probe**
- perf によるパフォーマンスプロファイルの記録および分析
- perf stat を使用したプロセス実行中のイベントのカウント

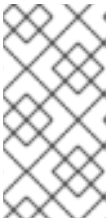
24.2. PERF を使用した関数内の行でのアップローブの作成

その後、このトレースポイントを **perf stat** や **perf record** などの他の **perf** ツールと併用すると、プロセスやアプリケーションの動作をよりよく理解できるようになります。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- 実行ファイルのデバッグシンボルを取得している。

```
# objdump -t ./your_executable | head
```



注記

これを行うには、実行ファイルの **debuginfo** パッケージをインストールする必要があります。または、実行ファイルがローカルで開発したアプリケーションの場合は、デバッグ情報 (GCC の **-g** オプション) を使用してアプリケーションをコンパイルする必要があります。

手順

1. プローブを配置できる関数行を表示します。

```
$ perf probe -x ./your_executable -L main
```

このコマンドの出力は、以下のようになります。

```
<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. 目的の関数行の uprobe を作成します。

```
# perf probe -x ./my_executable main:8
Added new event:
      probe_my_executable:main_L8   (on main:8 in /home/user/my_executable)

    You can now use it in all perf tools, such as:

        perf record -e probe_my_executable:main_L8 -aR sleep 1
```

24.3. UPROBE に記録されたデータのスクリプト出力を実行する

uprobe を使用して収集したデータを分析する一般的な方法は、**perf script** コマンドを使用して **perf.data** ファイルを読み込み、記録されたワークロードの詳細なトレースを表示することです。

perf スクリプトの出力例では、以下のようになります。

- **my_prog**と呼ばれるプログラムの関数 **isprime()** に uprobe が追加されます。
- **a** は、uprobe に追加された関数引数です。または、**a** を、uprobe を追加するコードスコープで表示される任意の変数にすることもできます。

```
# 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
```

第25章 PERF MEM によるメモリーアクセスのプロファイリング

perf mem コマンドを使用して、システム上でメモリーアクセスのサンプリングを行うことができます。

25.1. PERF MEM の目的

perf ツールの **mem** サブコマンドは、メモリーアクセスのサンプリング (読み込みおよ格納) を可能にします。**perf mem** コマンドは、メモリーのレイテンシーに関する情報、メモリーアクセスの種類、キャッシュヒットおよびミスを引き起こした機能、データシンボルの記録、これらのヒットおよびミスが発生するメモリーの場所に関する情報を提供します。

25.2. PERF MEM によるメモリーアクセスのサンプリング

この手順では、**perf mem** コマンドを使用して、システム上でメモリーアクセスのサンプリングを行う方法を説明します。このコマンドは、**perf record** および **perf report** と同じオプションと、**mem** サブコマンドに制限される一部のオプションをすべて取ります。記録されたデータは、後で分析するために、現在のディレクトリーの **perf.data** ファイルに保存されます。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

1. メモリーアクセスの例:

```
# perf mem record -a sleep seconds
```

この例では、**sleep** コマンドで指示されるように、**秒単位**の期間にわたる、すべての CPU でのメモリーアクセスを例に挙げています。**sleep** コマンドは、メモリーアクセスデータをサンプルしたいコマンドに置き換えることができます。デフォルトでは、**perf mem** は、メモリーロードおよびストアの両方をサンプルします。**-t** オプションを使用して、**perf mem** と **record** 間に load または store のいずれかを指定します。ロードすると、メモリー階層レベルに関する情報、TLB メモリーアクセス、バススヌーピング、およびメモリーロックがキャプチャーされます。

2. 分析用に **perf.data** ファイルを開きます。

```
# perf mem report
```

上記のコマンド例を使用すると、以下のような出力になります。

```
Available samples
35k cpu/mem-loads,ldlat=30/P
54k cpu/mem-stores/P
```

cpu/mem-loads,ldlat=30/P の行は、メモリーロードで収集されるデータを示し、**cpu/mem-stores/P** の行はメモリーストアで収集されるデータを示します。対象のカテゴリーを強調表示し、**Enter** を押してデータを表示します。

```
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
```

データを表示するときに、結果をソートして、対象の異なる側面を調査できます。たとえば、サンプリング期間中に発生したメモリーアクセスの種類ごとに、主な原因となるオーバーヘッドの降順で、メモリー負荷でデータを分類するには、以下を行います。

```
# perf mem -t load report --sort=mem
```

たとえば、以下のような出力になります。

```
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
```

関連情報

- man ページの **perf-mem(1)**

25.3. PERF NEN 出力の解釈

修飾子を指定せずに **perf mem report** コマンドを実行すると表示される表では、データを複数のコラムに分類します。

Overhead 列

その特定の機能で収集された全体のサンプルのパーセンテージを示します。

Samples 列

その行でアカウントを指定したサンプル数を表示します。

Local Weight の列

プロセッサーのコアサイクルでアクセスレイテンシーを表示します。

Memory Access の列

発生したメモリーアクセスのタイプを表示します。

Symbol 列

関数名またはシンボルを表示します。

Shared Object 列

サンプルの送信元である ELF イメージの名前を表示します (サンプルがカーネルからのものである場合に [kernel.kallsyms] という名前が使用されます)。

Data Symbol の列

行がターゲットとしていたメモリーの場所のアドレスを表示します。

重要

多くの場合、アクセスされるメモリーまたはスタックメモリーの動的割り当てにより、Data Symbol の列は raw アドレスを表示します。

Snoop の列

バストランザクションを表示します。

TLB アクセスの列

TLB メモリーアクセスを表示します。

Locked の列

関数がメモリーがロックされたか否かを示します。

デフォルトモードでは、関数は、オーバーヘッドの最も高いものが最初に表示される順に降順でソートされます。

第26章 偽共有の検出

偽共有は、対称型マルチプロセッシング (SMP) システムのプロセッサーコアが、プロセッサー間で共有されていない他のデータアイテムにアクセスするために、他のプロセッサーによって使用されている同じキャッシュラインのデータアイテムを変更すると発生します。

この初期修正では、キャッシュラインを使用する他のプロセッサーがコピーを無効にし、変更されたデータアイテムの更新バージョンをプロセッサーが必要としないにもかかわらず、または必然的にアクセスできる場合でも、更新されたコピーを要求する必要があります。

perf c2c コマンドを使用して、偽共有を検出できます。

26.1. PERF C2C の目的

perf ツールの **c2c** サブコマンドは、Shared Data Cache-to-Cache (C2C) 分析を有効にします。 **perf c2c** コマンドを使用して、キャッシュライン競合を検査し、true と false の両方の共有を検出できます。

キャッシュラインの競合は、対称型マルチプロセッシング (SMP) システムのプロセッサーコアが、他のプロセッサーによって使用されている同じキャッシュラインにあるデータオブジェクトを修正すると発生します。このキャッシュラインを使用する他のプロセッサーはすべて、コピーを無効にして更新されたものを要求します。これにより、パフォーマンスが低下する可能性があります。

perf c2c コマンドは、以下の情報を提供します。

- 競合が検出されたキャッシュライン
- データの読み取りおよび書き込みのプロセス
- 競合の原因となった命令
- 競合に関連する NUMA (Non-Uniform Memory Access) ノード

26.2. PERF C2C でキャッシュライン競合の検出

perf c2c コマンドを使用して、システム内のキャッシュライン競合を検出します。

perf c2c コマンドは、**perf record** と同じオプションと、**c2c** サブコマンドに排他的なオプションに対応します。記録されたデータは、後で分析するために、現在のディレクトリーの **perf.data** ファイルに保存されます。

前提条件

- **perf** ユーザー空間ツールがインストールされている。詳細は perf のインストール を参照してください。

手順

- **perf c2c** を使用してキャッシュラインの競合を検出します。

```
# perf c2c record -a sleep seconds
```

この例では、**sleep** コマンドによって指示される期間 (**seconds**) に対し、すべての CPU でキャッシュライン競合データをサンプルおよび記録します。**sleep** コマンドは、キャッシュライン競合データを収集するコマンドに置き換えることができます。

関連情報

- **perf-c2c(1)** の man ページ

26.3. PERF C2C レコードで記録された PERF.DATA ファイルの可視化

この手順では、**perf c2c** コマンドを使用して記録された **perf.data** ファイルを視覚化する方法を説明します。

前提条件

- **perf** ユーザー空間ツールがインストールされている。詳細は、perf のインストール を参照してください。
- **perf c2c** コマンドを使用して記録された **perf.data** ファイルは、現在のディレクトリーで利用できます。詳細は、perf c2c でキャッシュライン競合の検出 を参照してください。

手順

1. **perf.data** ファイルを開いて詳細な分析を行います。

```
# perf c2c report --stdio
```

このコマンドは、**perf.data** ファイルを端末内の複数のグラフに可視化します。

```
=================================================
            Trace Event Information
=================================================
  Total records                     :     329219
  Locked Load/Store Operations      :      14654
  Load Operations                   :      69679
  Loads - uncacheable               :          0
  Loads - IO                        :          0
  Loads - Miss                      :       3972
  Loads - no mapping                :          0
  Load Fill Buffer Hit              :      11958
  Load L1D hit                      :      17235
  Load L2D hit                      :         21
  Load LLC hit                      :      14219
  Load Local HITM                   :       3402
  Load Remote HITM                  :      12757
  Load Remote HIT                   :       5295
  Load Local DRAM                   :        976
  Load Remote DRAM                  :       3246
  Load MESI State Exclusive         :       4222
  Load MESI State Shared            :          0
  Load LLC Misses                   :      22274
  LLC Misses to Local DRAM          :        4.4%
  LLC Misses to Remote DRAM         :       14.6%
  LLC Misses to Remote cache (HIT)  :       23.8%
  LLC Misses to Remote cache (HITM) :       57.3%
  Store Operations                  :     259539
  Store - uncacheable               :          0
  Store - no mapping                :         11
  Store L1D Hit                     :     256696
```

```
 Store L1D Miss                  :       2832
 No Page Map Rejects             :       2376
 Unable to parse data source     :          1

=================================================
   Global Shared Cache Line Event Information
=================================================
 Total Shared Cache Lines        :         55
 Load HITs on shared lines       :      55454
 Fill Buffer Hits on shared lines :      10635
 L1D hits on shared lines        :      16415
 L2D hits on shared lines        :          0
 LLC hits on shared lines        :       8501
 Locked Access on shared lines   :      14351
 Store HITs on shared lines      :     109953
 Store L1D hits on shared lines  :     109449
 Total Merged records            :     126112

=================================================
               c2c details
=================================================
 Events                          : cpu/mem-loads,ldlat=30/P
                                 : cpu/mem-stores/P
 Cachelines sort on              : Remote HITMs
 Cacheline data groupping        : offset,pid,iaddr

=================================================
    Shared Data Cache Line Table
=================================================
#
#                           Total      Rmt  ----- LLC Load Hitm -----  ---- Store Reference ----  --- Load
Dram ----      LLC    Total  ----- Core Load Hit -----  -- LLC Load Hit --
# Index           Cacheline  records     Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss
Lcl       Rmt  Ld Miss    Loads       FB       L1       L2       Llc       Rmt
# .....  ...................  .......  .......  .......  .......  .......  .......  .......  .......  ........  ........  .......  .......  .......
.......  .......  ........  ........
#
     0           0x602180   149904   77.09%    12103     2269     9834   109504   109036      468
727      2657    13747    40400     5355    16154        0     2875       529
     1           0x602100    12128   22.20%     3951     1119     2832        0        0        0       65
200     3749    12128     5096      108        0     2056       652
     2  0xffff883ffb6a7e80      260    0.09%       15        3       12      161      161        0        1
1       15       99       25       50        0        6        1
     3  0xffffffff81aec000      157    0.07%        9        0        9        1        0        1        0        7
20      156       50       59        0       27        4
     4  0xffffffff81e3f540      179    0.06%        9        1        8      117       97       20        0       10
25       62       11        1        0       24        7

=================================================
     Shared Cache Line Distribution Pareto
=================================================
#
#        ----- HITM -----  -- Store Refs --        Data address                               ---------- cycles --
--------       cpu                                Shared
#   Num      Rmt      Lcl   L1 Hit  L1 Miss              Offset      Pid        Code address  rmt hitm  lcl
hitm      load       cnt              Symbol              Object                Source:Line  Node{cpu list}
```

```
# ..... ....... ....... ....... ....... ..... ....... ..... ....... ....... ....... ....... .....
..... ..... ..... ....
#
 ---------------------------------------------------------
    0    9834    2269   109036     468          0x602180
 ---------------------------------------------------------
       65.51%   55.88%   75.20%    0.00%             0x0    14604          0x400b4f     27161
26039     26017        9  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:144   0{0-1,4}  1{24-25,120}  2{48,54}  3{169}
   0.41%    0.35%    0.00%    0.00%             0x0    14604          0x400b56     18088
12601     26671        9  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:145   0{0-1,4}  1{24-25,120}  2{48,54}  3{169}
   0.00%    0.00%   24.80%  100.00%             0x0    14604          0x400b61        0        0
0        9  [.] read_write_func  no_false_sharing.exe  false_sharing_example.c:145   0{0-1,4}
1{24-25,120}  2{48,54}  3{169}
   7.50%    9.92%    0.00%    0.00%            0x20    14604          0x400ba7     2470
1729      1897        2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:154   1{122}  2{144}
  17.61%   20.89%    0.00%    0.00%            0x28    14604          0x400bc1     2294
1575      1649        2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:158   2{53}  3{170}
   8.97%   12.96%    0.00%    0.00%            0x30    14604          0x400bdb     2325
1897      1828        2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:162   0{96}  3{171}

 ---------------------------------------------------------
    1    2832     1119       0       0          0x602100
 ---------------------------------------------------------
  29.13%   36.19%    0.00%    0.00%            0x20    14604          0x400bb3     1964
1230      1788        2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:155   1{122}  2{144}
  43.68%   34.41%    0.00%    0.00%            0x28    14604          0x400bcd     2274
1566      1793        2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:159   2{53}  3{170}
  27.19%   29.40%    0.00%    0.00%            0x30    14604          0x400be7     2045
1247      2011        2  [.] read_write_func  no_false_sharing.exe
false_sharing_example.c:163   0{96}  3{171}
```

26.4. PERF C2C 出力の解釈

perf c2c report --stdio コマンドを実行して表示される視覚化は、データを複数のテーブルに分類します。

Trace Events Information

この表は、**perf c2c record** コマンドで収集された負荷およびストアサンプルの大まかな概要を示しています。

Global Shared Cache Line Event Information

この表は、共有キャッシュラインに関する統計を提供します。

c2c Details

この表は、サンプルされたイベントと、**perf c2c report** データを視覚化で編成する方法についての情報を提供します。

Shared Data Cache Line Table

この表は、デフォルトでキャッシュラインごとに検出されるリモート Hitm の量によって、偽共有が検出され、降順に並び替えられるホットテストキャッシュラインの 1 行の概要を提供します。

Shared Cache Line Distribution Pareto

以下の表には、競合が発生している各キャッシュラインに関するさまざまな情報が記載されています。

- キャッシュラインは **NUM** 列で番号 **0** から始まる番号です。
- 各キャッシュラインの仮想アドレスは **Data address Offset** の列に含まれます。また、その後に異なるアクセスが発生したキャッシュラインにオフセットが続きます。
- **Pid** 列にはプロセス ID が含まれます。
- **Code Address** 列には、インストラクションポインターコードアドレスが含まれます。
- **cycles** ラベル下の列には、平均負荷のレイテンシーが表示されます。
- **cpu cnt** 列には、生成された CPU の各種サンプルの数を表示します (特定の場所でインデックス化したデータを待つさまざまな CPU の数)。
- **Symbol** 列には関数名またはシンボルが表示されます。
- **Shared Object** 列は、サンプルの送信元である ELF イメージの名前を表示します (サンプルがカーネルからの場合は [**kernel.kallsyms**] という名前が使用されます)。
- **Source:Line** 列には、ソースファイルと行番号が表示されます。
- **Node{cpu list}** 列には、各ノードに対して、どの特定の CPU サンプルが生成されたかが表示されます。

26.5. PERF C2C を使用した偽共有の検出

この手順では、**perf c2c** コマンドを使用して偽共有を検出する方法を説明します。

前提条件

- **perf** ユーザー空間ツールがインストールされている。詳細は perf のインストール を参照してください。
- **perf c2c** コマンドを使用して記録された **perf.data** ファイルは、現在のディレクトリーで利用できます。詳細は、perf c2c でキャッシュライン競合の検出 を参照してください。

手順

1. **perf.data** ファイルを開いて詳細な分析を行います。

```
# perf c2c report --stdio
```

これにより、端末で **perf.data** ファイルが開きます。

2. Trace Event Information テーブルで、**LLC Misses to Remote Cache (HITM)**の値が含まれる行を見つけます。

LLC Misses to Remote Cache (HITM)の行の値コラムの割合は、変更したキャッシュラインのNUMA ノード全体で発生していた LLC ミスの割合を表し、偽共有が発生したことを示す主要な指標です。

```
=================================================
          Trace Event Information
=================================================
  Total records                 :     329219
  Locked Load/Store Operations  :      14654
  Load Operations               :      69679
  Loads - uncacheable           :          0
  Loads - IO                    :          0
  Loads - Miss                  :       3972
  Loads - no mapping            :          0
  Load Fill Buffer Hit          :      11958
  Load L1D hit                  :      17235
  Load L2D hit                  :         21
  Load LLC hit                  :      14219
  Load Local HITM               :       3402
  Load Remote HITM              :      12757
  Load Remote HIT               :       5295
  Load Local DRAM               :        976
  Load Remote DRAM              :       3246
  Load MESI State Exclusive     :       4222
  Load MESI State Shared        :          0
  Load LLC Misses               :      22274
  LLC Misses to Local DRAM      :       4.4%
  LLC Misses to Remote DRAM     :      14.6%
  LLC Misses to Remote cache (HIT) :      23.8%
  LLC Misses to Remote cache (HITM) : 57.3%
  Store Operations              :     259539
  Store - uncacheable           :          0
  Store - no mapping            :         11
  Store L1D Hit                 :     256696
  Store L1D Miss                :       2832
  No Page Map Rejects           :       2376
  Unable to parse data source   :          1
```

3. **Shared Data Cache Line Table**の **LLC Load Hitm** フィールドの **Rmt** 列を確認します。

```
  =================================================
           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
```

この表は、キャッシュ行ごとに検出されるリモート **Hitm** の量によって降順で並び替えられます。**LLC Load Hitm** セクションの **Rmt** 列の数値が大きい場合は、偽共有を示しており、偽共有アクティビティーをデバッグするには、それが発生したキャッシュラインをさらに検査する必要があります。

第27章 FLAMEGRAPHS の使用

システム管理者は、**flamegraphs** を使用して、**perf** ツールで記録されたシステムパフォーマンスデータの視覚化を作成できます。ソフトウェア開発者は、**flamegraphs** を使用して、**perf** ツールで記録されたアプリケーションパフォーマンスデータの視覚化を作成できます。

スタックトレースのサンプリングは、**perf** ツールを使用して CPU パフォーマンスをプロファイリングするための一般的な方法です。ただし、**perf** を使用したプロファイリングスタックトレースの結果は、極めて詳細にわたるので、分析の工数がかなりかかる可能性があります。**flamegraphs** は、**perf** で記録されたデータから作成された視覚化で、より早く、簡単にホットコードパスを特定できるようにします。

27.1. FLAMEGRAPHS のインストール

flamegraphs の使用を開始するには、必要なパッケージをインストールします。

手順

- **flamegraphs** パッケージをインストールします。

```
# yum install js-d3-flame-graph
```

27.2. システム全体でのフレームグラフの作成

この手順では、**flamegraphs** を使用して、システム全体で記録されたパフォーマンスデータを視覚化する方法を説明します。

前提条件

- **flamegraphs** が、flamegraphs のインストール の説明どおりにインストールされている。
- **perf** ツールが perf のインストール の説明どおりにインストールされている。

手順

- データを記録し、視覚化を作成します。

```
# perf script flamegraph -a -F 99 sleep 60
```

このコマンドは、**sleep** コマンドを使用して調整されるように、60 秒にわたりシステム全体でパフォーマンスデータをサンプルおよび記録し、現在のアクティブなディレクトリーに **flamegraph.html** として保存される視覚化を構築します。このコマンドは、デフォルトで呼び出しグラフデータをサンプルし、**perf** ツールと同じ引数を取ります。この例では、以下のようになります。

-a
システム全体でデータを記録するように調整します。

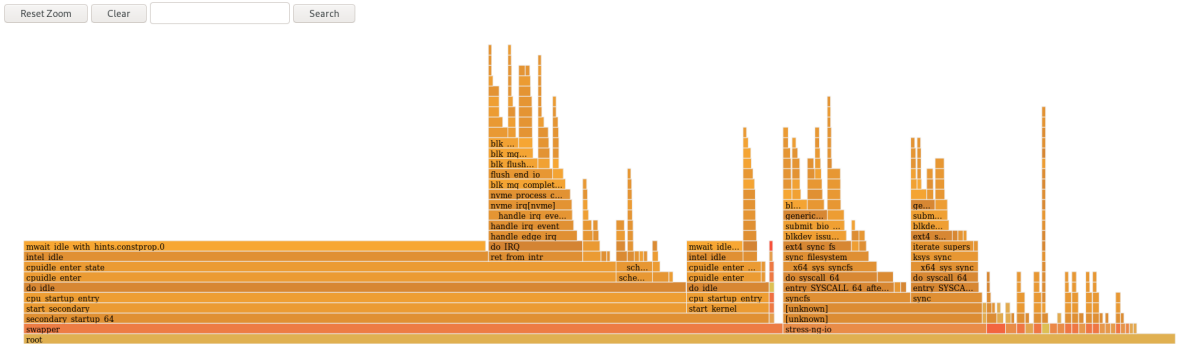
-F
1秒あたりのサンプリング頻度を設定します。

検証手順

- 分析するには、生成された視覚化を表示します。

```
# xdg-open flamegraph.html
```

このコマンドにより、デフォルトのブラウザーで視覚化が開きます。



27.3. 特定プロセスにおけるフレームグラフの作成

flamegraphs を使用して、特定の実行中のプロセスで記録されたパフォーマンスデータを視覚化できます。

前提条件

- **flamegraphs** が、flamegraphs のインストール の説明どおりにインストールされている。
- **perf** ツールが perf のインストール の説明どおりにインストールされている。

手順

- データを記録し、視覚化を作成します。

```
# perf script flamegraph -a -F 99 -p ID1,ID2 sleep 60
```

このコマンドは、**sleep** コマンドの使用で規定されているようにプロセス ID が **ID1** および **ID2** のプロセスのパフォーマンスデータを 60 秒間にわたりサンプルして記録します。次に、**flamegraph.html** として現在のアクティブディレクトリーに保存される視覚化を構築します。このコマンドは、デフォルトで呼び出しグラフデータをサンプルし、**perf** ツールと同じ引数を取ります。この例では、以下のようになります。

-a
システム全体でデータを記録するように調整します。

-F
1秒あたりのサンプリング頻度を設定します。

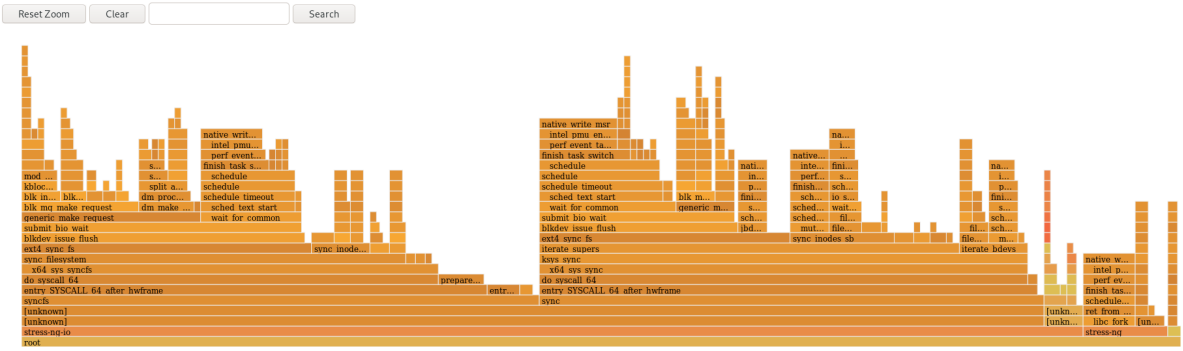
-p
特定のプロセス ID をシュミュレートし、データをサンプリングして記録します。

検証手順

- 分析するには、生成された視覚化を表示します。

```
# xdg-open flamegraph.html
```

このコマンドにより、デフォルトのブラウザーで視覚化が開きます。



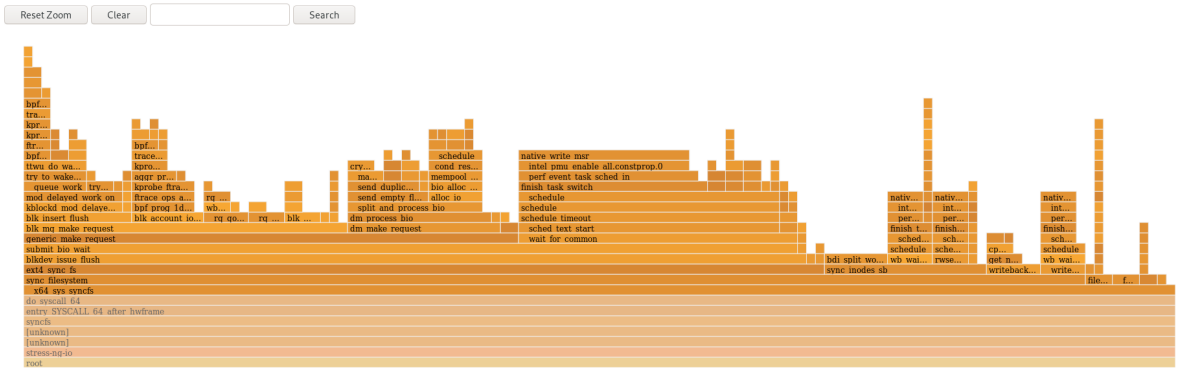
27.4. FLAMEGRAPHS の解釈

フレームグラフの各ボックスは、スタック内の異なる関数を示しています。スタックの深さは、x 軸で示されています。CPU でサンプルされた実際の関数は、最も上のボックスで、その他下のものは、その上位となります。X 軸は、サンプルの呼び出しグラフデータの近接性を表示します。

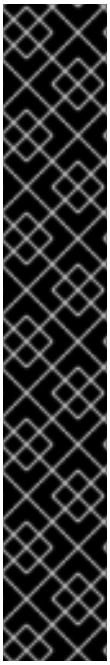
特定の行のスタックの子は、x 軸に沿って降順でそれぞれの関数から取得されたサンプルの数に基づいて表示されます。x 軸は時間の経過を表すものではありません。ボックスが広いほど、データがサンプリングされていたときに、CPU 上または CPU 上の一部での頻度が高いことを意味します。

手順

- 以前表示されていない可能性のある関数の名前を確認するには、フレームグラフ内のボックスをクリックしてその特定の場所のスタックに拡大します。

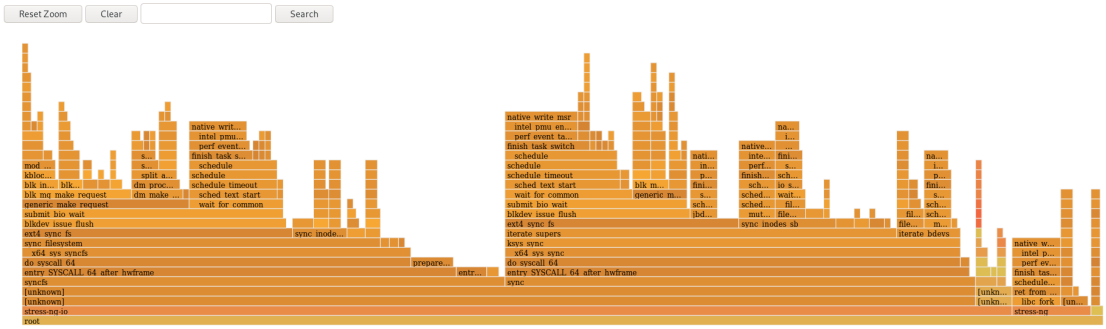


- フレームグラフのデフォルトビューに戻るには、**Reset Zoom** をクリックします。



重要

ユーザー空間関数を表すボックスには、関数のバイナリーが取り除かれているため、**flamegraphs** で **Unknown** とラベルが付けられる場合があります。実行可能ファイルの **debuginfo** パッケージがインストールされているか、実行可能ファイルがローカルで開発したアプリケーションである場合は、アプリケーションはデバッグ情報と共にコンパイルされる必要があります。GCC で **-g** オプションを使用して、このような状況で関数名またはシンボルを表示します。



関連情報

- perf が一部の関数名を raw 関数アドレスとして表示する理由
- デバッグ情報を使用したデバッグの有効化

第28章 PERF CIRCULAR バッファーを使用したパフォーマンスのボトルネックの監視

システムで実行されている特定のプロセスまたはアプリケーションの一部のパフォーマンスのボトルネックを監視するために、**perf** ツールを使用してデータのイベント固有のスナップショットを取得する循環バッファーを作成できます。このような場合には、**perf** は、指定されたイベントが検出されると、後で分析するために **perf.data** ファイルにデータのみを書き込みます。

28.1. PERF を使用した循環バッファーおよびイベント固有のスナップショット

perf を使用してプロセスまたはアプリケーションでパフォーマンスの問題を調査する場合、特定の対象イベントが発生する前の数時間のデータを記録することは、簡単ではない、または適切ではない場合があります。このような場合は、**perf record** を使用して、特定のイベントの後にスナップショットを取得するカスタムの循環バッファーを作成できます。

--overwrite オプションを使用すると、**perf record** は上書き可能な循環バッファーにすべてのデータを保存します。バッファーがいっぱいになると、**perf record** が最も古いレコードを自動的に上書きするため、**perf.data** ファイルに書き込まれることはありません。

--overwrite および **--switch-output-event** オプションを併用すると、**--switch-output-event** トリガーイベントを検出するまで、データを継続的に記録およびダンプする循環バッファーが設定されます。トリガーイベントは、ユーザーが関心のある何かが発生したので、循環バッファー内のデータを **perf.data** ファイルに書き込むように **perf record** に通知します。これにより、関心のある特定のデータが収集されると同時に、**perf.data** ファイルに不要なデータを書き込まないことで、実行中の **perf** プロセスのオーバーヘッドが削減されます。

28.2. PERF 循環バッファーを使用したパフォーマンスのボトルネックを監視するための特定のデータの収集

perf ツールを使用すると、関心のあるデータのみを収集するために指定したイベントによってトリガーされる循環バッファーを作成できます。イベント固有のデータを収集する循環バッファーを作成するには、**perf** に **--overwrite** および **--switch-output-event** オプションを使用します。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- プロセスまたはアプリケーション内の関心のある場所に、監視したいプロセスまたはアプリケーションに uprobe を配置している。

```
# 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
```

手順

- トリガーイベントとして uprobe を使用して循環バッファーを作成します。

```
# 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> ]
```

この例では、実行可能ファイルを開始し、**-e** オプションの後に指定された CPU サイクルを、**perf** が **--switch-output-event** オプションの後に指定されたトリガーイベントである uprobe を検出するまで収集します。この時点で、**perf** は循環バッファーにあるすべてのデータのスナップショットを取得し、タイムスタンプで識別される一意の **perf.data** ファイルに保存します。この例では、合計 2 つのスナップショットが生成され、最後の **perf.data** ファイルは **Ctrl**+**c** を押すことによって強制されました。

第29章 PERF を停止または再起動せずに、実行中の PERF コレクターからトレースポイントを追加および削除する

コントロールパイプインターフェイスを使用して、実行中の **perf** コレクターで異なるトレースポイントを有効化および無効化することで、**perf** を停止または再起動せずに、収集するデータを動的に調整できます。これにより、プロセスの停止または再起動中に記録されたはずのパフォーマンスデータが失われることはありません。

29.1. PERF を停止または再起動せずに、実行中の PERF コレクターにトレースポイントを追加する

コントロールパイプインターフェイスを使用して実行中の **perf** コレクターにトレースポイントを追加し、**perf** を停止してパフォーマンスデータを損失することなく録画中のデータを調整します。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。

手順

1. 制御パイプインターフェイスを設定します。

```
# mkfifo control ack perf.pipe
```

2. コントロールファイル設定と、有効にするイベントで **perf record** を実行します。

```
# perf record --control=fifo:control,ack -D -1 --no-buffering -e 'sched:*' -o - > perf.pipe
```

この例では、**-e** オプションの後に **'sched:*'** を宣言すると、スケジューラーイベントで **perf record** が開始されます。

3. 2 つ目の端末で、制御パイプの読み取り側を起動します。

```
# cat perf.pipe | perf --no-pager script -i -
```

コントロールパイプの読み取り側を起動すると、最初の端末で以下のメッセージがトリガーされます。

```
Events disabled
```

4. 3 つ目の端末で、制御ファイルを使用してトレースポイントを有効にします。

```
# echo 'enable sched:sched_process_fork' > control
```

このコマンドは **perf** をトリガーし、宣言されたイベントについて制御ファイル内の現在のイベントリストをスキャンします。イベントが存在する場合は、トレースポイントが有効になり、次のメッセージが最初の端末に表示されます。

```
event sched:sched_process_fork enabled
```

トレースポイントを有効にすると、2 つ目の端末は、トレースポイントを検出した **perf** の出力を表示します。

```
bash 33349 [034] 149587.674295: sched:sched_process_fork: comm=bash pid=33349
child_comm=bash child_pid=34056
```

29.2. PERF を停止または再起動せずに、実行中の PERF コレクターからトレースポイントを削除する

制御パイプインターフェイスを使用して、実行中の **perf** コレクターからトレースポイントを削除し、**perf** を停止してパフォーマンスデータを失うことなく収集するデータの範囲を減らします。

前提条件

- perf のインストール で説明されているように、**perf** ユーザー領域ツールがインストールされている。
- 制御パイプインターフェイスを介して、トレースポイントを実行中の **perf** コレクターに追加している。詳細は、perf を停止または再起動せずに、実行中の perf コレクターにトレースポイントを追加する を参照してください。

手順

- トレースポイントを削除します。

```
# echo 'disable sched:sched_process_fork' > control
```

注記

この例では、以前にスケジューラーイベントをコントロールファイルに読み込み、トレースポイント **sched:sched_process_fork** を有効にしていることを前提としています。

このコマンドは **perf** をトリガーし、宣言されたイベントについて制御ファイル内の現在のイベントリストをスキャンします。イベントが存在する場合は、トレースポイントが無効になり、制御パイプの設定に使用する端末に以下のメッセージが表示されます。

```
event sched:sched_process_fork disabled
```

第30章 NUMASTAT を使用したメモリー割り当てのプロファイリング

numastat ツールを使用すると、システムのメモリー割り当てに関する統計を表示できます。

numastat ツールは、各 NUMA ノードのデータを個別に表示します。この情報を使用して、システムのメモリーパフォーマンスや、システムのさまざまなメモリーポリシーの効果を調査できます。

30.1. デフォルトの NUMASTAT 統計

デフォルトでは、**numastat** ツールは、各 NUMA ノードの以下のカテゴリーに対する統計を表示します。

numa_hit

このノードに正常に割り当てられていたページ数。

numa_miss

対象のノードのメモリーが少ないために、このノードに割り当てたページ数。それぞれの **numa_miss** イベントには、別のノードで対応する **numa_foreign** イベントがあります。

numa_foreign

代わりに別のノードに割り当てられたこのノードについて最初に意図されたページ数です。それぞれの **numa_foreign** イベントには、対応する **numa_miss** イベントが別のノードにあります。

interleave_hit

このノードに正常に割り当てられるインターリーブポリシーページの数。

local_node

このノードのプロセスで、このノードで正常に割り当てられるページ数。

other_node

別のノードのプロセスでこのノードに割り当てられるページ数。

注記

高い **numa_hit** の値と低い **numa_miss** の値 (互いに相対的) は、最適なパフォーマンスを示します。

30.2. NUMASTAT を使用したメモリー割り当ての表示

numastat ツールを使用してシステムのメモリー割り当てを表示できます。

前提条件

- **numactl** パッケージをインストールする。

```
# yum install numactl
```

手順

- システムのメモリー割り当てを表示する。

```
$ 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
```

関連情報

- **numastat(8)** の man ページ

第31章 CPU 使用率を最適化するためのオペレーティングシステムの設定

ワークロード全体で CPU 使用率を最適化するように、オペレーティングシステムを設定できます。

31.1. プロセッサーの問題を監視および診断するためのツール

以下は、プロセッサー関連のパフォーマンス問題を監視および診断するために Red Hat Enterprise Linux 8 で利用可能なツールです。

- **turbostat** ツールは、指定した間隔でカウンターの結果を出力し、過剰な電力使用量、ディープスリープ状態に入れない、システム管理割り込み (SMI) が不必要に作成されるなど、サーバーでの予期しない動作を特定するのに役立ちます。
- **numactl** ユーティリティーはプロセッサーとメモリー親和性を管理する数多くのオプションを提供します。**numactl** パッケージには、カーネルがサポートする NUMA ポリシーに簡単なプログラミングインターフェイスを提供する **libnuma** ライブラリーが含まれており、**numactl** アプリケーションよりも詳細なチューニングに使用できます。
- **numastat** ツールは、オペレーティングシステムおよびそのプロセスについての NUMA ノードごとのメモリー統計を表示し、プロセスのメモリーがシステム全体に分散されているか、特定のノードで集中化されているかを表示します。このツールは、**numactl** パッケージで提供されます。
- **numad** は NUMA アフィニティーの自動管理デーモンです。NUMA リソースの割り当てと管理を動的に改善するために、システム内の NUMA トポロジーとリソースの使用状況を監視します。
- **/proc/interrupts** ファイルには割り込み要求 (IRQ) 番号、システムの各プロセッサーによって処理される同様の割り込み要求の数、送信される割り込みのタイプ、およびリスト表示される割り込み要求に応答するデバイスのコンマ区切りのリストが表示されます。
- **pqos** ユーティリティーは **intel-cmt-cat** パッケージで利用できます。最新の Intel プロセッサーで CPU キャッシュとメモリー帯域幅を監視します。以下を監視します。
  - サイクルごとの命令 (IPC)。
  - 最終レベルのキャッシュ MISSES の数。
  - LLC で特定の CPU で実行されるプログラムのサイズ (キロバイト単位)。
  - ローカルメモリーへの帯域幅 (MBL)。
  - リモートメモリー (MBR) への帯域幅。
- **X86_energy_perf_policy** ツールを使用すると、パフォーマンスと電力消費効率の相対的な重要性を定義できます。この情報は、パフォーマンスと電力消費効率の間でトレードオフするオプションを選択すると、この機能をサポートするプロセッサーに影響を与えるために使用できます。
- **taskset** ツールは、**util-linux** パッケージで提供されます。これにより、管理者は実行中のプロセスのプロセッサー親和性を取得および設定したり、指定されたプロセッサー親和性でプロセスを起動したりできます。

関連情報

- **turbostat(8)**、**numactl(8)**、**numastat(8)**、**numa(7)**、**numad(8)**、**pqos(8)**、**x86_energy_perf_policy(8)**、および **taskset(1)** の man ページ

31.2. システムトポロジーの種類

現代のコンピューティングでは、ほとんどの最近のシステムに複数のプロセッサーがあるため、CPU の意図は誤解を招くものです。システムのトポロジーは、これらのプロセッサー同士が、他のシステムリソースに接続する方法です。これにより、システムおよびアプリケーションのパフォーマンスに影響を及ぼし、システムのチューニングの考慮事項が影響を受ける可能性があります。

現代のコンピューティングで使用されるトポロジーの主なタイプを以下に示します。

SMP (symmetric Multi-Processor) トポロジー

SMP トポロジーにより、すべてのプロセッサーが同時にメモリーにアクセスできるようになります。ただし、共有および同等のメモリーアクセスは、本質的にすべての CPU からのメモリーアクセスをシリアライズするため、SMP システムのスケーリング制約が一般的に許容できないものとして表示されます。このため、最近のサーバーシステムはすべて NUMA マシンです。

NUMA (Non-Uniform Memory Access) の固定 (ピニング)

NUMA トポロジーは、SMP トポロジーよりも最近開発されました。NUMA システムでは、複数のプロセッサーが 1 つのソケット上で物理的にグループ化されます。各ソケットには、そのメモリーへのローカルアクセスを持つメモリーとプロセッサーの専用領域があります。これらは、すべてノードと呼ばれます。同じノード上のプロセッサーは、そのノードのメモリーバンクに高速でアクセスでき、ノード上にないメモリーバンクへの低速アクセスを提供します。
そのため、ローカル以外のメモリーにアクセスするとパフォーマンスが低下します。したがって、NUMA トポロジーを使用するシステム上のパフォーマンスに敏感なアプリケーションは、アプリケーションを実行するプロセッサーと同じノードにあるメモリーにアクセスする必要があり、可能な限りリモートメモリーにアクセスしないようにしてください。

パフォーマンスに敏感するマルチスレッドアプリケーションは、特定のプロセッサーではなく特定の NUMA ノードで実行されるように設定することで、メリットが得られます。これが適切なかどうかは、システムやアプリケーションの要件によって異なります。複数のアプリケーションスレッドが同じキャッシュされたデータにアクセスする場合、同じプロセッサーでこれらのスレッドを実行するように設定することが適切な場合があります。ただし、異なるデータにアクセスし、キャッシュする複数のスレッドが同じプロセッサーで実行される場合、各スレッドは、以前のスレッドによってアクセスされたキャッシュデータをエビクトする可能性があります。これは、各スレッドがキャッシュを失い、メモリーからデータをフェッチし、これをキャッシュで置き換えていることを意味します。**perf** ツールを使用して、過剰な数のキャッシュミスをチェックします。

31.2.1. システムトポロジーの表示

システムのトポロジーを理解するのに便利なコマンドは複数あります。この手順では、システムトポロジーを確認する方法を説明します。

手順

- システムトポロジーの概要を表示するには、以下のコマンドを実行します。

```
$ 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
[...]
```

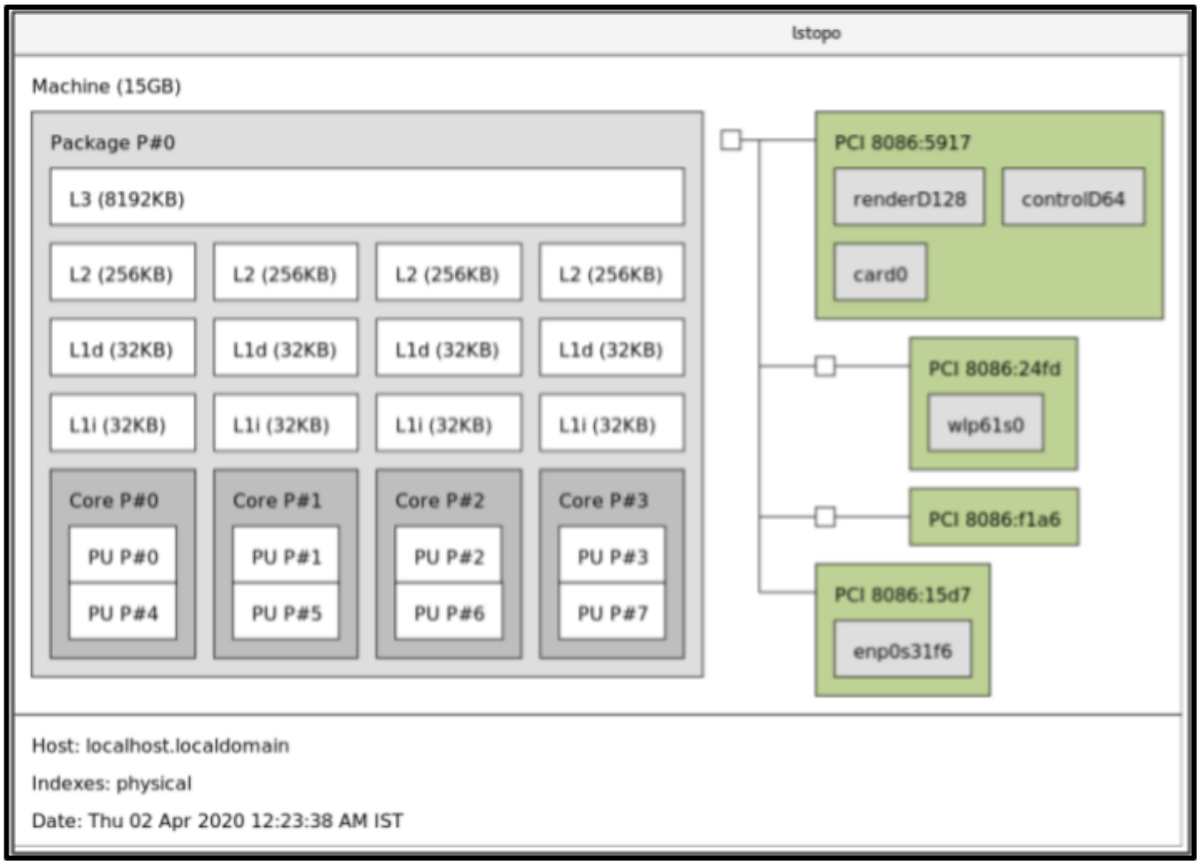
- CPU 数、スレッド数、コア数、ソケット数、NUMA ノード数などの CPU アーキテクチャーに関する情報を収集するには、以下を実行します。

```
$ 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
```

- システムのグラフィカル表現を表示するには、以下のコマンドを実行します。

```
# yum install hwloc-gui
# lstopo
```

図31.1 lstopo の出力



- 詳細なテキスト出力を表示するには、次のコマンドを実行します。

```
# yum install hwloc
# lstopo-no-graphics
Machine (15GB)
  Package L#0 + L3 L#0 (8192KB)
    L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
      PU L#0 (P#0)
      PU L#1 (P#4)
     HostBridge L#0
    PCI 8086:5917
      GPU L#0 "renderD128"
      GPU L#1 "controlD64"
      GPU L#2 "card0"
    PCIBridge
      PCI 8086:24fd
        Net L#3 "wlp61s0"
    PCIBridge
      PCI 8086:f1a6
    PCI 8086:15d7
      Net L#4 "enp0s31f6"
```

関連情報

- **numactl(8)**、**lscpu(1)**、および **lstopo(1)** の man ページ

31.3. カーネルティック時間の設定

デフォルトでは、Red Hat Enterprise Linux 8 はティックレスカーネルを使用します。これは、電力使用量を低減し、新しいプロセッサーがディープスリープ状態を利用できるようにするためにアイドル状態の CPU を中断しません。

Red Hat Enterprise Linux 8 には動的ティックレスオプションもあります。これは、高パフォーマンスコンピューティングやリアルタイムのコンピューティングなどのレイテンシーに制約のあるワークロードに役立ちます。デフォルトでは、動的ティックレスオプションは無効になっています。Red Hat は、**cpu-partitioning TuneD** プロファイルを使用して、**isolated_cores** で指定されたコアの動的な tickless オプションを有効にすることを推奨します。

この手順では、動的なティックレス動作を永続的に有効にする方法を説明します。

手順

1. 特定のコアで動的なティックレス動作を有効にするには、**nohz_full** パラメーターを使用して、カーネルコマンドラインでこれらのコアを指定します。16 コアシステムでは、**nohz_full=1-15** カーネルオプションを有効にします。

```
# grubby --update-kernel=ALL --args="nohz_full=1-15"
```

   これにより、コア **1** から **15** までの動的なティックレス動作が有効になり、すべての時間管理が未指定のコアのみに移動します (コア **0**)。

2. システムが起動したら、**rcu** スレッドをレイテンシーを区別しないコア (この場合は core **0**) に手動で移動します。

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

3. オプション: カーネルコマンドラインで **isolcpus** パラメーターを使用して、特定のコアをユーザー空間タスクから分離します。

4. オプション: カーネルの **write-back bdi-flush** スレッドの CPU 親和性をハウスキーピングコアに設定します。

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

検証手順

- システムを再起動したら、**dynticks** が有効になっていることを確認します。

```
# journalctl -xe | grep dynticks
Mar 15 18:34:54 rhel-server kernel: NO_HZ: Full dynticks CPUs: 1-15.
```

- 動的ティックレス設定が正しく機能していることを確認します。

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
```

  このコマンドは、CPU 1 に 3 秒間スリープするように指示しながら、CPU 1 のティックを測定します。

- デフォルトのカーネルタイマーの設定では、通常の CPU で 3100 ティックが表示されます。

```
# 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
```

- 動的ティックレスカーネルを設定すると、代わりに 4 ティックが表示されるはずです。

```
# 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
```

関連情報

- **perf(1)** および **cpuset(7)** の man ページ
- All about nohz_full kernel parameter in RHEL 7 (Red Hat ナレッジベースの記事)
- How to verify list of "isolated" and "nohz_full" CPU information from sysfs?Red Hat ナレッジベースの記事

31.4. 割り込み要求の概要

割り込み要求または IRQ は、ハードウェアの一部からプロセッサーに直ちに送信されるシグナルです。システム内の各デバイスには、固有の割り込みを送信できる IRQ 番号が割り当てられます。割り込みが有効になっていると、割り込み要求を受信するプロセッサーは割り込み要求に対応するために現在のアプリケーションスレッドの実行を即時に一時停止します。

割り込みは通常の動作を停止するため、割り込み率が高くなると、システムのパフォーマンスが大幅に低下する可能性があります。割り込みの親和性を設定するか、優先度の低い割り込みをバッチ (複数の割り込みをまとめる) に送信することで、割り込みにかかる時間を低減することができます。

割り込み要求には関連するアフィニティープロパティー **smp_affinity** があり、割り込み要求を処理するプロセッサーを定義します。アプリケーションのパフォーマンスを向上させるには、割り込みの親和性とプロセスの親和性を同じプロセッサーまたは同じコアにあるプロセッサーに割り当てます。これにより、指定された割り込みとアプリケーションスレッドがキャッシュラインを共有できるようになります。

割り込みステアリングに対応するシステムでは、割り込み要求の **smp_affinity** プロパティーを変更するとハードウェアが設定され、カーネルを介入することなくハードウェアレベルで特定のプロセッサーに割り込みを処理させる決定が行われるようになります。

31.4.1. 割り込みの手動分散

BIOS が NUMA トポロジーをエクスポートする場合、**irqbalance** サービスは、サービスを要求するハードウェアに対してローカルとなるノードで割り込み要求を自動的に処理できます。

手順

1. 設定する割り込み要求に対応するデバイスを確認します。

2. プラットフォームのハードウェア仕様を見つけます。システムのチップセットが割り込みの分散に対応しているかどうかを確認します。

   a. その場合には、以下の手順に従って割り込み配信を設定できます。また、チップセットが割り込みの分散に使用するアルゴリズムを確認してください。BIOS によっては割り込み配信を設定するオプションがあります。

   b. そうでない場合は、チップセットは常にすべての割り込みを単一の静的 CPU にルーティングします。使用される CPU を設定することはできません。

3. システムでどの Advanced Programmable Interrupt Controller (APIC) モードが使用されているかを確認します。

```
$ journalctl --dmesg | grep APIC
```

   ここでは、以下のようになります。

   - システムが **flat** 以外のモードを使用している場合は、**APIC ルーティングの物理フラットへの設定** と同様の行が表示されます。
   - このようなメッセージが表示されない場合は、システムが **flat** モードを使用します。システムで **x2apic** モードを使用している場合は、**bootloader** 設定のカーネルコマンドラインに **nox2apic** オプションを追加して無効にできます。

     物理以外のフラットモード (**flat**) のみが、複数の CPU への割り込みの分散をサポートします。このモードは、CPU が最大 **8** のシステムでのみ利用できます。

4. **smp_affinity マスク** を計算します。**smp_affinity mask** の計算方法については、smp_affinity マスクの設定 を参照してください。

関連情報

- **journalctl(1)** および **taskset(1)** の man ページ

31.4.2. smp_affinity マスクの設定

smp_affinity の値は、システム内のすべてのプロセッサーを表す 16 進数のビットマスクとして保存されます。各ビットは異なる CPU を設定します。最も大きなビットは CPU 0 です。

マスクのデフォルト値は **f** で、割り込み要求をシステム内のどのプロセッサーでも処理できることを意味します。この値を 1 に設定すると、プロセッサー 0 のみが割り込みを処理できます。

手順

1. バイナリーでは、割り込みを処理する CPU に 1 の値を使用します。たとえば、割り込みを処理する CPU 0 と CPU 7 を設定するには、バイナリーコードに **0000000010000001** を使用します。

表31.1 CPU のバイナリービット

CPU	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
バイナリー	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

2. バイナリーコードを 16 進数に変換します。
   たとえば、Python を使用してバイナリーコードを変換するには、次のコマンドを実行します。

```
>>> hex(int('0000000010000001', 2))

'0x81'
```

   プロセッサーが 32 個を超えるシステムでは、32 ビットグループごとに **smp_affinity** 値を区切る必要があります。たとえば、64 プロセッサーシステムの最初の 32 プロセッサーのみが割り込み要求を処理できるようにするには、**0xffffffff,00000000** を使用します。

3. 特定の割り込み要求の割り込み親和性の値は、関連付けられた **/proc/irq/irq_number/smp_affinity** ファイルに保存されます。このファイルで **smp_affinity** マスクを設定します。

```
# echo mask > /proc/irq/irq_number/smp_affinity
```

関連情報

- **journalctl(1)**、**irqbalance(1)**、および **taskset(1)** の man ページ

第32章 スケジューリングポリシーの調整

Red Hat Enterprise Linux では、プロセス実行の最小単位はスレッドと呼ばれます。システムスケジューラーは、スレッドを実行するプロセッサーと、スレッドの実行期間を決定します。ただし、スケジューラーの主な懸念はシステムをビジーに維持することであるため、アプリケーションのパフォーマンスに対してスレッドを最適にスケジュールしない場合があります。

たとえば、NUMA システムのアプリケーションがノード A で実行され、ノード B のプロセッサーが利用可能になるとします。プロセッサーをノード B でビジー状態に維持するために、スケジューラーはアプリケーションのスレッドのいずれかをノード B に移動します。ただし、アプリケーションスレッドは依然としてノード A のメモリーにアクセスする必要があります。ただし、スレッドがノード B で実行され、ノード A のメモリーがスレッドに対してローカルにならないため、このメモリーへのアクセスには時間がかかります。そのため、スレッドがノード B での実行を終了するには、ノード A のプロセッサーが利用可能になるまで待機してから、ローカルメモリーアクセスで元のノードでスレッドを実行するよりも時間がかかる場合があります。

32.1. スケジューリングポリシーの分類

パフォーマンス重視のアプリケーションには、多くの場合、スレッドが実行される場所を決定する手段や管理者の恩恵を受けることができます。Linux スケジューラーは、スレッドの実行場所と実行期間を決定する複数のスケジューリングポリシーを実装します。

以下は、スケジューリングポリシーの主なカテゴリーです。

Normal policies

通常スレッドは、通常の優先度のタスクに使用されます。

Realtime policies

リアルタイムポリシーは、中断なしで完了する必要のある時間的制約のあるタスクに使用されます。リアルタイムスレッドは、タイムスライスの対象ではありません。つまり、スレッドは、ブロック、終了、デプロイメント、または優先度の高いスレッドによってプリエンプションされるまで実行されます。
最も優先度の低いリアルタイムスレッドは、通常のポリシーを持つスレッドの前にスケジュールされます。詳しくは、Static priority scheduling with SCHED_FIFO および Round robin priority scheduling with SCHED_RR をご覧ください。

関連情報

- **sched(7)**、**sched_setaffinity(2)**、**sched_getaffinity(2)**、**sched_setscheduler(2)**、および **sched_getscheduler(2)** の man ページ

32.2. SCHED_FIFO を使用した静的優先度スケジューリング

SCHED_FIFO は静的優先度スケジューリングとも呼ばれ、各スレッドに固定の優先度を定義するリアルタイムポリシーです。このポリシーにより、管理者はイベントの応答時間を改善し、レイテンシーを短縮できます。このポリシーは、時間的制約のあるタスクについて長期間実行しないようにすることが推奨されます。

SCHED_FIFO が使用されている場合、スケジューラーは **SCHED_FIFO** の全スレッドのリストを優先度順にスキャンし、実行準備ができているスレッドを最も優先度の高いものとしてスケジュールします。**SCHED_FIFO** スレッドの優先度レベルは、**1** から **99** までの任意の整数にすることができます。ここで、**99** は最も高い優先度として処理されます。Red Hat は、レイテンシーの問題を特定する場合にのみ、数値を減らし、優先度を増加させることを推奨します。

警告

リアルタイムスレッドはタイムスライスの影響を受けないため、Red Hat は優先度を 99 に設定しないことを推奨します。これにより、プロセスは移行およびウォッチドッグスレッドと同じ優先レベルを維持します。スレッドが演算ループに入り、これらのスレッドがブロックされると、実行できなくなります。単一のプロセッサーを持つシステムは、この状況では最終的にハングします。

管理者は、**SCHED_FIFO** 帯域幅を制限し、リアルタイムのアプリケーションプログラマーがプロセッサーを単調にするリアルタイムのタスクを開始できないようにすることができます。

以下は、このポリシーで使用されるパラメーターの一部です。

/proc/sys/kernel/sched_rt_period_us

このパラメーターは、プロセッサー帯域幅の 100 パーセントとみなされる期間 (マイクロ秒単位) を定義します。デフォルト値は **1000000 µs** (**1 秒**) です。

/proc/sys/kernel/sched_rt_runtime_us

このパラメーターは、リアルタイムスレッドを実行する時間 (マイクロ秒単位) を定義します。デフォルト値は **950000 µs** (**0.95 秒**) です。

32.3. SCHED_RR を使用したラウンドロビン優先度スケジューリング

SCHED_RR は、**SCHED_FIFO** のラウンドロビン型です。このポリシーは、複数のスレッドを同じ優先レベルで実行する必要がある場合に役に立ちます。

SCHED_FIFO と同様に、**SCHED_RR** は、各スレッドに固定の優先度を定義するリアルタイムポリシーです。スケジューラーは、すべての SCHED_RR スレッドのリストを優先度順にスキャンし、実行準備ができているスレッドを最も優先度の高いものとしてスケジュールします。ただし、**SCHED_FIFO** とは異なり、優先順位が同じスレッドは、特定のタイムスライス内のラウンドロビンスタイルでスケジュールされます。

このタイムスライスの値は、**/proc/sys/kernel/sched_rr_timeslice_ms** ファイルの **sched_rr_timeslice_ms** カーネルパラメーターでミリ秒単位で設定できます。最小値は **1 millisecond** です。

32.4. SCHED_OTHER を使用した通常のスケジューリング

SCHED_OTHER は、Red Hat Enterprise Linux 8 のデフォルトスケジューリングポリシーです。このポリシーは Completely Fair Scheduler (CFS) を使用して、このポリシーでスケジュールされているすべてのスレッドへの公平プロセッサーアクセスを許可します。このポリシーは、スレッドが多数ある場合や、データスループットの優先度が優先される場合に最も便利です。これは、時間とともにスレッドをより効率的にスケジュールできるためです。

このポリシーが使用されると、スケジューラーは各プロセススレッドの niceness 値に基づいて動的な優先順位リストを作成します。管理者はプロセスの niceness 値を変更できますが、スケジューラーの動的優先順位のリストを直接変更することはできません。

32.5. スケジューラーポリシーの設定

chrt コマンドラインツールを使用してスケジューラーポリシーおよび優先順位を確認し、調整します。これは、必要なプロパティーで新規プロセスを開始したり、実行中のプロセスのプロパティーを変更したりできます。また、ランタイム時にポリシーを設定するのにも使用できます。

手順

1. アクティブなプロセスのプロセス ID (PID) を表示します。

```
# ps
```

   ps コマンドで **--pid** または **-p** オプションを使用して、特定の PID の詳細を表示します。

2. 特定のプロセスのスケジューリングポリシー、PID、および優先順位を確認します。

```
# 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
```

   ここで、468 と 476 はプロセスの PID です。

3. プロセスのスケジューリングポリシーを設定します。

   a. たとえば、PID 1000 のプロセスを、優先度が 50 の **SCHED_FIFO** に設定するには、以下を実行します。

```
# chrt -f -p 50 1000
```

   b. たとえば、PID 1000 のプロセスを、優先度が 0 の **SCHED_OTHER** に設定するには、以下を実行します。

```
# chrt -o -p 0 1000
```

   c. たとえば、PID 1000 のプロセスを、優先度が 10 の **SCHED_RR** に設定するには、以下を実行します。

```
# chrt -r -p 10 1000
```

   d. 特定のポリシーおよび優先度で新規アプリケーションを開始するには、アプリケーションの名前を指定します。

```
# chrt -f 36 /bin/my-app
```

関連情報

- **chrt(1)** の man ページ
- Policy Options for the chrt command
- ブートプロセス中のサービス優先度の変更

32.6. CHRT コマンドのポリシーオプション

chrt コマンドでは、プロセスのスケジューリングポリシーを表示および設定することができます。

次の表は、プロセスのスケジューリングポリシーを設定するために使用できる、適切なポリシーオプションについて説明しています。

表32.1 chrt コマンドのポリシーオプション

短いオプション	Long オプション	説明
-f	**--fifo**	スケジュールを **SCHED_FIFO** に設定
-o	**--other**	スケジュールを **SCHED_OTHER** に設定
-r	**--rr**	スケジュールを **SCHED_RR** に設定します。

32.7. ブートプロセス中のサービス優先度の変更

systemd サービスを使用すると、システムの起動プロセス中に起動したサービスに対して、リアルタイムの優先度を設定できます。**ユニット設定ディレクティブ** は、ブートプロセス中にサービスの優先度を変更するために使用されます。

ブートプロセスの優先度の変更は、service セクションの以下のディレクティブを使用して行われます。

CPUSchedulingPolicy=

実行したプロセスの CPU スケジューリングポリシーを設定します。これは、**他** のポリシー、**fifo** ポリシー、および **rr** ポリシーを設定するために使用されます。

CPUSchedulingPriority=

実行したプロセスの CPU スケジューリングの優先度を設定します。利用可能な優先度の範囲は、選択した CPU スケジューリングポリシーによって異なります。リアルタイムスケジューリングポリシーでは、**1** (最も低い優先度) から **99** (最も高い優先度) までの整数を使用できます。

以下の手順では、ブートプロセス中に **mcelog** サービスを使用してサービスの優先度を変更する方法を説明します。

前提条件

1. TuneD パッケージをインストールします。

```
# yum install tuned
```

2. TuneD サービスを有効にして起動している。

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

手順

1. 実行中のスレッドのスケジュールの優先度を表示します。

```
# 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. 補助の **mcelog** サービス設定ディレクトリーファイルを作成し、このファイルにポリシー名と優先度を挿入します。

```
# cat << EOF > /etc/systemd/system/mcelog.service.d/priority.conf

[Service]
CPUSchedulingPolicy=fifo
CPUSchedulingPriority=20
EOF
```

3. **systemd** スクリプト設定を再読み込みします。

```
# systemctl daemon-reload
```

4. **mcelog** サービスを再起動します。

```
# systemctl restart mcelog
```

検証手順

- **systemd** 問題で設定した **mcelog** 優先度を表示します。

```
# tuna -t mcelog -P
thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
826     FIFO    20  0,1,2,3         13            0          mcelog
```

関連情報

- **systemd(1)** および **tuna(8)** の man ページ
- 優先度範囲の説明

32.8. 優先順位マップ

優先度はグループで定義され、特定のカーネル機能専用のグループもあります。リアルタイムスケジューリングポリシーでは、**1** (最も低い優先度) から **99** (最も高い優先度) までの整数を使用できます。

次の表は、プロセスのスケジューリングポリシーを設定する際に使用できる優先度の範囲を示しています。

表32.2 優先度範囲の説明

優先度	Threads	説明
1	優先度の低いカーネルスレッド	通常、この優先度は **SCHED_OTHER** よりは優先度の高いタスク用に予約されます。
2 - 49	利用可能	標準的なアプリケーションの優先度に使用される範囲。
50	デフォルトの IRQ 値	
51 - 98	優先度の高いスレッド	この範囲は、定期的に実行されるスレッドと、迅速な応答時間に使用します。CPU にバインドされたスレッドには割り込みが必要になるため、この範囲を使用しないでください。
99	ウォッチドッグおよび移行	最も高い優先順位で実行される必要があるシステムスレッド。

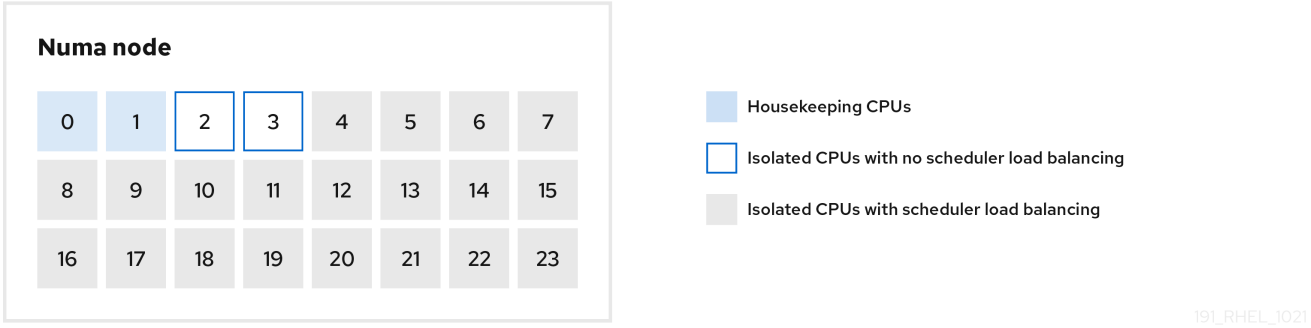
32.9. TUNED CPU-PARTITIONING プロファイル

レイテンシーに敏感なワークロード用に Red Hat Enterprise Linux 8 を調整する場合は、**cpu-partitioning** TuneD プロファイルを使用することが推奨されます。

Red Hat Enterprise Linux 8 以前では、低レイテンシーの Red Hat ドキュメントで、低レイテンシーのチューニングを実現するために必要な低レベルの手順が数多く説明されていました。Red Hat Enterprise Linux 8 では、**cpu-partitioning** TuneD プロファイルを使用することで、低レイテンシーのチューニングをより効率的に実行できます。このプロファイルは、個々の低レイテンシーアプリケーションの要件に従って簡単にカスタマイズできます。

以下の図は、**cpu-partitioning** プロファイルの使用方法を示す例になります。この例では、CPU とノードのレイアウトを使用します。

図32.1 cpu-partitioning の図



/etc/tuned/cpu-partitioning-variables.conf ファイルで cpu-partitioning プロファイルを設定するには、以下の設定オプションを使用します。

負荷分散機能のある分離された CPU

cpu-partitioning の図では、4 から 23 までの番号が付けられたブロックが、デフォルトの分離された CPU です。カーネルスケジューラーのプロセスの負荷分散は、この CPU で有効になります。これは、カーネルスケジューラーの負荷分散を必要とする複数のスレッドを使用した低レイテンシープロセス用に設計されています。
isolated_cores=cpu-list オプションを使用して、**/etc/tuned/cpu-partitioning-variables.conf** ファイルで cpu-partitioning プロファイルを設定できます。このオプションは、カーネルスケジューラーの負荷分散を使用する分離する CPU をリスト表示します。

分離された CPU のリストはコンマ区切りで表示するか、**3-5** のようにハイフンを使用して範囲を指定できます。このオプションは必須です。このリストにない CPU は、自動的にハウスキーピング CPU と見なされます。

負荷分散を行わずに分離した CPU

cpu-partitioning の図では、2 と 3 の番号が付けられたブロックは、追加のカーネルスケジューラープロセスの負荷分散を提供しない分離された CPU です。
/etc/tuned/cpu-partitioning-variables.conf ファイルで cpu-partitioning プロファイルを設定するには、**no_balance_cores=cpu-list** オプションを使用します。このオプションは、カーネルスケジューラーの負荷分散を使用しない CPU を分離するようにリスト表示します。

no_balance_cores オプションの指定は任意ですが、このリストの CPU は、**isolated_cores** リストに記載されている CPU のサブセットである必要があります。

このような CPU を使用するアプリケーションスレッドは、各 CPU に個別にピン留めする必要があります。

ハウスキーピング CPU

cpu-partitioning-variables.conf ファイル内で分離されていない CPU は、自動的にハウスキーピング CPU と見なされます。ハウスキーピング CPU では、すべてのサービス、デーモン、ユーザープロセス、移動可能なカーネルスレッド、割り込みハンドラー、およびカーネルタイマーの実行が許可されます。

関連情報

- **tuned-profiles-cpu-partitioning(7)** man ページ

32.10. 低レイテンシーチューニングへの TUNED の CPU-PARTITIONING プロファイルの使用

この手順では、TuneD の **cpu-partitioning** プロファイルを使用して、低レイテンシーになるようにシステムをチューニングする方法を説明します。これは、cpu-partitioning の図で説明されているように、**cpu-partitioning** と CPU レイアウトを使用できる低レイテンシーのアプリケーションの例を使用します。

この場合のアプリケーションでは、以下を使用します。

- ネットワークからデータを読み込む 1 つの専用リーダースレッドが、CPU 2 に固定されます。
- このネットワークデータを処理する多数のスレッドは、CPU 4-23 に固定されます。

- 処理されたデータをネットワークに書き込む専用のライタースレッドは、CPU 3 に固定されます。

前提条件

- **yum install tuned-profiles-cpu-partitioning** コマンドを root で使用して、**cpu-partitioning** TuneD プロファイルをインストールしている。

手順

1. **/etc/tuned/cpu-partitioning-variables.conf** ファイルを編集し、以下の内容を追加します。

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

2. **cpu-partitioning** TuneD プロファイルを設定します。

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

3. 再起動
   再起動後、システムは、cpu-partitioning の図の分離に従って、低レイテンシーにチューニングされます。このアプリケーションでは、タスクセットを使用して、リーダーおよびライターのスレッドを CPU 2 および 3 に固定し、残りのアプリケーションスレッドを CPU 4-23 に固定できます。

関連情報

- **tuned-profiles-cpu-partitioning(7)** man ページ

32.11. CPU-PARTITIONING TUNED プロファイルのカスタマイズ

TuneD プロファイルを拡張して、追加のチューニング変更を行うことができます。

たとえば、**cpu-partitioning** プロファイルは、**cstate=1** を使用する CPU を設定します。**cpu-partitioning** プロファイルを使用しながら、cstate1 から cstate0 に CPU の cstate を変更するために、以下の手順では **my_profile** という名前の新しい TuneD プロファイルを説明しています。このプロファイルは、**cpu-partitioning** プロファイルを継承した後、C state-0 を設定します。

手順

1. **/etc/tuned/my_profile** ディレクトリーを作成します。

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

2. このディレクトリーに **tuned.conf** ファイルを作成し、次の内容を追加します。

```
# 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. 新しいプロファイルを使用します。

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

注記

この共有例では、再起動は必要ありません。ただし、**my_profile** プロファイルの変更を有効にするために再起動が必要な場合は、マシンを再起動します。

関連情報

- **tuned-profiles-cpu-partitioning(7)** man ページ

第33章 I/O およびファイルシステムパフォーマンスに影響を与える要因

ストレージおよびファイルシステムパフォーマンスに適した設定は、ストレージの目的より大きく左右されます。

I/O およびファイルシステムのパフォーマンスは、以下のいずれかの要因により影響を受ける可能性があります。

- データの書き込みまたは読み取りパターン
- 順次または無作為
- バッファーまたはダイレクト IO
- 基礎となるジオメトリーとのデータ調整
- ブロックサイズ
- ファイルシステムのサイズ
- ジャーナルサイズおよび場所
- アクセス時間の記録
- データの信頼性確保
- 事前にフェッチするデータ
- ディスク領域の事前割り当て
- ファイルの断片化
- リソースの競合

33.1. I/O およびファイルシステムの問題を監視および診断するツール

Red Hat Enterprise Linux 8 では、システムパフォーマンスを監視し、I/O、ファイルシステム、その設定に関連するパフォーマンス問題を診断するために、以下のツールを利用できます。

- **vmstat** ツールは、システム全体のプロセス、メモリー、ページング、ブロック I/O、割り込み、および CPU アクティビティーに関する報告を行います。管理者はこのツールを使用することで、パフォーマンスの問題が I/O サブシステムによるものかを判断しやすくなります。**vmstat** を使用した分析で、I/O サブシステムが原因でパフォーマンスが低下していることがわかった場合に、管理者は **iostat** ツールを使用して原因となる I/O デバイスを判別できます。

- **iostat** は、システムでの I/O デバイスの負荷を報告します。このツールは **sysstat** パッケージで提供されます。

- **blktrace** は、I/O サブシステムでの時間の使用について詳細にわたる情報を提供します。同梱のユーティリティーである **blkparse** は、**blktrace** からのロー出力を読み取り、**blktrace** が記録した入出力操作を人間が判読できるようにまとめます。

- **btt** は **blktrace** 出力を分析し、I/O スタックのエリアごとにデータが費やした時間を表示するので、I/O サブシステムのボトルネックを見つけやすくなります。このユーティリティー

は、**blktrace** パッケージの一部として提供されます。**blktrace** メカニズムで追跡され、**btt** が分析する重要なイベントには、以下のようなものがあります。

  - I/O イベント (**Q**) のキュー
  - ドライバーイベント (**D**) への I/O のディスパッチ
  - I/O イベントの完了 (**C**)

- **iowatcher** は **blktrace** 出力を使用して、I/O を経時的にグラフ化できます。このツールは、ディスク I/O の論理ブロックアドレス (LBA)、1 秒あたりのスループット (メガバイト単位)、シーク数および I/O 操作に重点を置いています。これを使用することで、デバイスの演算回数の上限に到達するタイミングを判断しやすくなります。

- BPF コンパイラーコレクション (BCC) は、**eBPF** (extended Berkeley Packet Filter) プログラムの作成を容易にするライブラリーです。**eBPF** プログラムは、ディスク I/O、TCP 接続、プロセス作成などのイベントでトリガーされます。BCC ツールは、**/usr/share/bcc/tools/** ディレクトリーにインストールされます。以下の **bcc-tools** は、パフォーマンスの分析に役立ちます。

  - **biolatency** は、ブロックデバイス I/O(ディスク I/O) のレイテンシーをヒストグラムにまとめています。これにより、デバイスのキャッシュヒット、キャッシュミス、レイテンシーアウトライナーの 2 つのモードなど、分散を調査できます。
  - **biosnoop** は、基本的なブロック I/O 追跡ツールで、各 I/O イベントの表示、プロセス ID および I/O レイテンシーの発行を行います。このツールを使用して、ディスク I/O パフォーマンスの問題を調査できます。
  - **biotop** は、カーネルのブロック I/O 操作に使用します。
  - **Filelife** ツールは、 **stat()** システムコールを追跡します。
  - **fileslower** は、読み取りと書き込みが遅い同期ファイルを追跡します。
  - **filetop** は、プロセスによるファイルの読み取りと書き込みを表示します。
  - **ext4slower**、**nfsslower** および **xfsslower** は、特定のしきい値よりも操作速度の遅いファイルを表示するツールです (デフォルト値 **10ms**)。
    詳細は、BPF コンパイラーコレクションでシステムパフォーマンスの分析 を参照してください。

- **bpftace** は、パフォーマンスの問題の分析に使用される **eBPF** のトレース言語です。また、システムの監査用に BCC のような追跡ユーティリティーが含まれており、I/O のパフォーマンスの問題を調査するのに役立ちます。

- 以下の **SystemTap** スクリプトは、ストレージまたはファイルシステムのパフォーマンスの問題の診断に役立ちます。

  - **disktop.stp**: 5 秒ごとにディスクの読み取りまたは書き込みのステータスを確認し、その期間の上位 10 エントリーを出力します。
  - **iotime.stp**: 読み取り、書き込み操作に使用した時間、読み取りおよび書き込みバイト数を出力します。
  - **traceio.stp**: 確認された累積 I/O トラフィックに基づいて上位 10 の実行可能ファイルを秒単位で出力します。

- **traceio2.stp**: 指定したデバイスに読み取りおよび書き込みが行われると、実行可能ファイル名とプロセス ID を出力します。
- **inodewatch.stp**: 指定したメジャー/マイナーデバイスで、指定の inode に対して読み取りまたは書き込みが行われるたびに、実行可能ファイル名とプロセス ID を出力します。
- **inodewatch2.stp**: 指定したメジャー/マイナーデバイスの指定の inode で属性が変更されるたびに、実行可能ファイル名とプロセス ID、属性を出力します。

関連情報

- **vmstat(8)**、**iostat(1)**、**blktrace(8)**、**blkparse(1)**、**btt(1)**、**bpftrace** および **iowatcher(1)** の man ページ
- BPF コンパイラーコレクションでシステムパフォーマンスの分析

33.2. ファイルシステムのフォーマットに利用可能なチューニングオプション

一部のファイルシステム設定は一旦決定すると、デバイスのフォーマット後に変更できません。

以下は、ストレージデバイスをフォーマットする前に利用可能なオプションです。

Size (サイズ)

ワークロードに適したサイズのファイルシステムを作成します。ファイルシステムが小さいと、ファイルシステムのチェックにかかる時間とメモリーが少なくて済みます。ただし、ファイルシステムが小さすぎると、断片化が多くなり、パフォーマンスが低下します。

ブロックサイズ

ブロックは、ファイルシステムの作業単位です。ブロックサイズで、単一のブロックに保存可能なデータ量が決まるので、1度に書き込みまたは読み取りされる最小データ量を指します。
デフォルトのブロックサイズは、ほとんどのユースケースに適しています。ただし、ブロックサイズや複数のブロックのサイズが、一般的に一度に読み取る/書き込むデータ量と同じか、若干多い場合には、ファイルシステムのパフォーマンスは向上し、データの保存をより効率的に行います。
ファイルが小さい場合は、引き続きブロック全体を使用します。ファイルは複数のブロックに分散できますが、ランタイム時のオーバーヘッドが増える可能性があります。

また、ファイルシステムによっては、特定のブロック数が上限となっており、ファイルシステムの最大数が制限されます。**mkfs** コマンドでデバイスをフォーマットする時に、ファイルシステムのオプションの一部としてブロックサイズを指定します。ブロックサイズを指定するパラメーターは、ファイルシステムによって異なります。

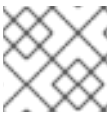
ジオメトリー

ファイルシステムジオメトリーは、ファイルシステム全体でのデータの分散に関係があります。システムで RAID などのストライプ化ストレージを使用する場合は、デバイスのフォーマット時にデータおよびメタデータを下層にあるストレージジオメトリーに合わせて調整することで、パフォーマンスを向上させることができます。
多くのデバイスは推奨のジオメトリーをエクスポートし、デバイスが特定のファイルシステムでフォーマットされるとそのジオメトリーが自動的に設定されます。デバイスでこのような推奨ジオメトリーをエクスポートしない場合や、推奨の設定を変更する場合には、**mkfs** コマンドでデバイスのフォーマット時にジオメトリーを指定する必要があります。

ファイルシステムのジオメトリーを指定するパラメーターは、ファイルシステムによって異なります。

外部ジャーナル

ジャーナリングファイルシステムは、操作を実行する前に、ジャーナルファイルに書き込み操作中に加えられる変更を文書化します。これにより、システムクラッシュや電源異常の発生時にストレージデバイスが破損する可能性が低下し、復旧プロセスが高速化されます。



注記

Red Hat では、外部ジャーナルオプションの使用は推奨していません。

メタデータ集約型のワークロードでは、ジャーナルへの更新頻度が非常に多くなります。ジャーナルが大きいと、より多くのメモリーを使用しますが、書き込み操作の頻度は低減します。さらに、プライマリーストレージと同じか、それ以上の速度の専用ストレージにジャーナルを配置することで、メタデータ集約型のワークロードでデバイスのシーク時間を短縮できます。

警告

外部ジャーナルが信頼できることを確認します。外部ジャーナルデバイスが損失すると、ファイルシステムが破損します。外部ジャーナルは、フォーマット時に作成し、マウント時にジャーナルデバイスを指定する必要があります。

関連情報

- man ページの **mkfs(8)** および **mount(8)**
- 利用可能なファイルシステムの概要

33.3. ファイルシステムのマウントに利用可能なチューニングオプション

以下は、ほとんどのファイルシステムで利用可能なオプションで、デバイスのマウント時に指定できます。

Access Time

ファイルが読み込まれるたびに、ファイルのメタデータはアクセス時点で更新されます (**atime**)。この際、追加の書き込み I/O が行われます。ほとんどのファイルシステムの **atime** のデフォルト設定は **relatime** です。
ただし、このメタデータの更新に時間がかかる場合で正確なアクセス時間データが必要ない場合には、**noatime** マウントオプションを指定してファイルシステムをマウントしてください。この設定で、ファイルの読み取り時にメタデータへの更新が無効になります。また、**nodiratime** 動作も有効にし、ディレクトリーの読み取り時にメタデータへの更新を無効にします。

注記

noatime mount オプションを使用して **atime** の更新を無効にすると、バックアッププログラムなどに依存するアプリケーションが破損する可能性があります。

read-ahead

Read-ahead 動作では、すぐに必要となる可能性の高いデータを事前にフェッチし、ページキャッ

シュ (ディスク上にある場合よりも早くデータを取得可能) に読み込むことでファイルのアクセス時間を短縮します。read-ahead 値が大きいほど、さらに事前にシステムのデータがフェッチされます。
Red Hat Enterprise Linux は、ファイルシステムについて検出した内容に基づいて、適切な read-ahead 値の設定を試みます。ただし、正確な検出が常に可能であるとは限りません。たとえば、ストレージアレイが単一の LUN としてシステムに表示した場合に、システムはその単一の LUN を検出するので、アレイに適した read-ahead 値は設定されません。

連続 I/O を大量にストリーミングするワークロードは、read-ahead 値を高くすると効果がある場合が多いです。Red Hat Enterprise Linux で提供されるストレージ関連の tuned プロファイルは、LVM ストライプ化と同様に read-ahead 値を増やしますが、このような調整は、ワークロードすべてで常に十分であるというわけではありません。

関連情報

- **mount(8)**、**xfs(5)**、および **ext4(5)** の man ページ

33.4. 未使用ブロックの破棄の種類

ファイルシステムで未使用のブロックを破棄することは、ソリッドステートディスク (SSD) およびシンプロビジョニングストレージのいずれの場合でも推奨のプラクティスです。

以下は、未使用のブロックを破棄する 2 つの方法です。

バッチ破棄

fstrim コマンドに、このタイプの破棄が含まれています。ファイルシステム内にある未使用のブロックで、管理者が指定した基準に一致するものをすべて破棄します。Red Hat Enterprise Linux 8 は、XFS および ext4 でフォーマットされおり、物理的な破棄操作に対応するデバイスでのバッチ破棄をサポートします。

オンライン破棄

このタイプの破棄操作は、discard オプションを指定してマウント時に設定します。この操作は、ユーザーの介入なしにリアルタイムで実行されます。ただし、未使用から空き状態に移行しているブロックのみを破棄します。Red Hat Enterprise Linux 8 では XFS および ext4 フォーマットのデバイスでオンライン破棄をサポートしています。
Red Hat は、パフォーマンスを維持するためにオンライン破棄が必要な場合や、システムのワークロードでバッチ破棄を実行できない場合を除き、バッチ破棄を推奨します。

事前割り当てでは、領域にデータを書き込むことなく、ファイルに割り当て済みとしてディスク領域をマークします。これは、データの断片化や、読み取りのパフォーマンスの低下を抑える場合に役立ちます。Red Hat Enterprise Linux 8 は、XFS、ext4、および GFS2 ファイルシステムでの領域の事前割り当てに対応します。アプリケーションは、**fallocate(2) glibc** 呼び出しを使用して、事前割り当てした領域の利点を活用することもできます。

関連情報

- **mount(8)** および **fallocate(2)** の man ページ

33.5. ソリッドステートディスク (SSD) の調整に関する考慮事項

ソリッドステートディスク (SSD) は、回転磁気プラッターではなく、NAND フラッシュチップを使用して永続データを保存します。SSD は、完全な論理ブロックアドレス範囲のデータにアクセスする時間を一定に保ち、回転プラッターのように、測定できるレベルでシーク数を犠牲にすることがありませ

ん。ギガバイト単位のストレージ領域としてはより高価で、ストレージ密度が少なくなっていますが、HDD に比べ、レイテンシーが低く、スループットが高くなっています。

SSD の使用済みのブロックが、ディスク容量を占有してくると、通常パフォーマンスは低下します。パフォーマンスの低下レベルはベンダーによって異なりますが、このような状況では、すべてのデバイスでパフォーマンスが低下します。破棄動作を有効にすると、この低下を軽減できます。詳細は、未使用ブロックの破棄の種類 を参照してください。

デフォルトの I/O スケジューラーおよび仮想メモリーオプションは、SSD での使用に適しています。設定時には、SSD のパフォーマンスに影響を与える可能性がある以下の要素を考慮してください。

I/O スケジューラー

I/O スケジューラーはどれでも、ほとんどの SSD で適切に動作することが想定されます。ただし、他のストレージタイプと同様に、特定のワークロードに対する最適な設定を決定するためにベンチマーク評価を行うことを推奨します。SSD を使用する場合、I/O スケジューラーの変更は特定のワークロードのベンチマーク評価を行う場合に限ることを推奨しています。I/O スケジューラー間の切り替え方法は、**/usr/share/doc/kernel-version/Documentation/block/switching-sched.txt** ファイルを参照してください。
単一キュー HBA の場合は、デフォルトの I/O スケジューラーは **deadline** です。複数のキュー HBA の場合は、デフォルトの I/O スケジューラーは **none** です。I/O スケジューラーの設定方法は、ディスクスケジューラーの設定 を参照してください。

仮想メモリー

I/O スケジューラーと同様に、仮想メモリー (VM) サブシステムには特別なチューニングは必要ありません。SSD の I/O が高速であるという性質をもとに、**vm_dirty_background_ratio** と **vm_dirty_ratio** 設定の値を下げ、書き出しのアクティビティーが増えても通常は、ディスクの他の操作のレイテンシーに悪影響はありません。ただし、このチューニングで全体的な I/O が生み出されるので、ワークロード固有のテストがない場合には通常推奨していません。

スワップ

SSD はスワップデバイスとしても使用でき、ページアウトおよびページインのパフォーマンスが向上する可能性が高くなります。

33.6. 汎用ブロックデバイスのチューニングパラメーター

ここにリストされている一般的なチューニングパラメーターは、**/sys/block/sdX/queue/** ディレクトリーにあります。

以下のチューニングパラメーターは、I/O スケジューラーチューニングとは別のパラメーターで、全 I/O スケジューラーに適用されます。

add_random

一部の I/O イベントは、**/dev/random** のエントロピープールに貢献します。これらの貢献のオーバーヘッドが測定できるレベルになった場合には、このパラメーターを **0** に設定してください。

iostats

デフォルトでは、**iostats** は有効で、デフォルト値は **1** です。**iostats** 値を **0** に設定すると、デバイスの I/O 統計の収集が無効になり、I/O パスでのオーバーヘッドが少し削除されます。また、**iostats** を **0** に設定すると、特定の NVMe ソリッドステートストレージデバイスなど、非常に高性能なデバイスのパフォーマンスが若干向上します。特定のストレージモデルで無効にするようにベンダーからの指示がない限り、**iostats** は有効のままにしておくことを推奨します。
iostats を無効にすると、デバイスの I/O 統計が **/proc/diskstats** ファイルからなくなります。I/O 情報は、**/sys/diskstats** ファイルの内容をソースとしており、**sar** や **iostats** などの I/O ツールの監視に使用します。したがって、デバイスの **iostats** パラメーターを無効にすると、デバイスは I/O 監視ツールの出力に表示されなくなります。

max_sectors_kb

I/O 要求の最大サイズを KB 単位で指定します。デフォルト値は **512** KB です。このパラメーターの最小値は、ストレージデバイスの論理ブロックサイズで決まります。このパラメーターの最大値は、**max_hw_sectors_kb** の値で決まります。
Red Hat は、**max_sectors_kb** を常に最適な I/O サイズと内部消去ブロックサイズの倍数にするように推奨しています。0 が指定されているか、ストレージデバイスによる指定がない場合には、どちらかのパラメーターの **logical_block_size** の値を使用します。

nomerges

要求をマージは、ほとんどのワークロードで有用です。ただし、デバッグの目的では、マージを無効にすると便利です。デフォルトでは、**nomerges** パラメーターは **0** に設定されており、マージが有効です。単純な 1 回だけのマージを無効にするには **nomerges** を **1** に設定します。すべてのタイプのマージを無効にするには、**nomerges** を **2** に設定します。

nr_requests

キューに配置可能な最大 I/O 数です。現在の I/O スケジューラーが **none** の場合は、この数値を減らすことができますが、それ以外の場合は数字の増減が可能です。

optimal_io_size

このパラメーターで最適な I/O サイズを報告するストレージデバイスもあります。この値が報告される場合は、できるだけ報告された最適な I/O サイズに合わせその倍数の I/O をアプリケーションで発行させることを推奨しています。

read_ahead_kb

連続読み込み操作中にオペレーティングシステムが読み取ることができる最大キロバイト数を定義します。その結果、必要な情報は、次の連続読み込みのカーネルページキャッシュにすでに存在するので、読み取り I/O のパフォーマンスが向上します。
read_ahead_kb の値を大きくすると、デバイスマッパーは効果を得られます。開始点として、各デバイスに対して **128** KB の値に指定すると適切です。ディスクの **read_ahead_kb** の値を、要求キューのディスクの **max_sectors_kb** まで増やすと、大型のファイルの連続読み込みが行われるアプリケーション環境でパフォーマンスが向上する可能性があります。

rotational

一部のソリッドステートディスクは、ソリッドステートのステータスを正しく公開せず、従来の回転ディスクとしてマウントされます。スケジューラーで不要なシーク時間短縮ロジックを無効にするには、**rotational** の値を **0** に手動で設定します。

rq_affinity

rq_affinity のデフォルト値は **1** です。これは、発行した CPU コアと同じ CPU グループにある CPU コア 1 つで I/O 操作を完了します。I/O 要求を発行したプロセッサーでのみ I/O 操作を完了するには、**rq_affinity** を **2** に設定します。上記の 2 つの機能を無効にするには、**0** に設定します。

scheduler

特定のストレージデバイスにスケジューラーまたはスケジューラーの優先度を設定するには、**/sys/block/devname/queue/scheduler** ファイルを編集します。ここで、devname は設定するデバイスの名前に置き換えます。

第34章 ネットワークパフォーマンスのチューニング

ネットワーク設定のチューニングは、考慮すべき要素が多数ある複雑なプロセスです。たとえば、これには、CPU からメモリーへのアーキテクチャー、CPU コアの量などが含まれます。Red Hat Enterprise Linux は、ほとんどのシナリオに最適化されたデフォルト設定を使用します。ただし、場合によっては、ネットワーク設定をチューニングして、スループットや遅延を増やしたり、パケットドロップなどの問題を解決したりする必要があります。

34.1. ネットワークアダプター設定のチューニング

40 Gbps 以上の高速ネットワークでは、ネットワークアダプター関連のカーネル設定の特定のデフォルト値がパケットドロップやパフォーマンス低下の原因となる可能性があります。これらの設定をチューニングすると、このような問題を防ぐことができます。

34.1.1. nmcli を使用して、高いパケットドロップ率を減らすためにリングバッファーサイズを増やす

パケットドロップ率が原因でアプリケーションがデータの損失、タイムアウト、またはその他の問題を報告する場合は、イーサネットデバイスのリングバッファーのサイズを増やします。

受信リングバッファーは、デバイスドライバーとネットワークインターフェイスコントローラー (NIC) の間で共有されます。カードは、送信 (TX) および受信 (RX) リングバッファーを割り当てます。名前が示すように、リングバッファーは循環バッファーであり、オーバーフローによって既存のデータが上書きされます。NIC からカーネルにデータを移動するには、ハードウェア割り込みと、SoftIRQ とも呼ばれるソフトウェア割り込みの 2 つの方法があります。

カーネルは RX リングバッファーを使用して、デバイスドライバーが着信パケットを処理できるようになるまで着信パケットを格納します。デバイスドライバーは、通常は SoftIRQ を使用して RX リングをドレインします。これにより、着信パケットは **sk_buff** または **skb** と呼ばれるカーネルデータ構造に配置され、カーネルを経由して関連するソケットを所有するアプリケーションまでの移動を開始します。

カーネルは TX リングバッファーを使用して、ネットワークに送信する必要がある発信パケットを保持します。これらのリングバッファーはスタックの一番下にあり、パケットドロップが発生する重要なポイントであり、ネットワークパフォーマンスに悪影響を及ぼします。

手順

1. インターフェイスのパケットドロップ統計を表示します。

```
# ethtool -S enp1s0
    ...
    rx_queue_0_drops: 97326
    rx_queue_1_drops: 63783
    ...
```

   コマンドの出力は、ネットワークカードとドライバーに依存することに注意してください。

   discard または **drop** カウンターの値が高い場合は、カーネルがパケットを処理できるよりも速く、使用可能なバッファーがいっぱいになることを示します。リングバッファーを増やすと、このような損失を回避できます。

2. 最大リングバッファーサイズを表示します。

```
# ethtool -g enp1s0
 Ring parameters for enp1s0:
 Pre-set maximums:
 RX:             4096
 RX Mini:        0
 RX Jumbo:       16320
 TX:             4096
 Current hardware settings:
 RX:             255
 RX Mini:        0
 RX Jumbo:       0
 TX:             255
```

Pre-set maximums セクションの値が **Current hardware settings** セクションよりも高い場合は、次の手順で設定を変更できます。

3. このインターフェイスを使用する NetworkManager 接続プロファイルを特定します。

```
# nmcli connection show
NAME                UUID                                  TYPE      DEVICE
Example-Connection  a5eb6490-cc20-3668-81f8-0314a27f3f75  ethernet  enp1s0
```

4. 接続プロファイルを更新し、リングバッファーを増やします。

   - RX リングバッファーを増やすには、次のように入力します。

```
# nmcli connection modify Example-Connection ethtool.ring-rx 4096
```

   - TX リングバッファーを増やすには、次のように入力します。

```
# nmcli connection modify Example-Connection ethtool.ring-tx 4096
```

5. NetworkManager 接続をリロードします。

```
# nmcli connection up Example-Connection
```

重要

NIC が使用するドライバーによっては、リングバッファーを変更すると、ネットワーク接続が短時間中断されることがあります。

関連情報

- ifconfig および ip コマンドがパケットドロップを報告する
- 0.05% のパケットドロップ率について心配する必要がありますか?
- **ethtool(8)** man ページ

34.1.2. パケットドロップを回避するためにネットワークデバイスのバックログキューをチューニングする

ネットワークカードがパケットを受信し、カーネルプロトコルスタックがこれらのパケットを処理する前に、カーネルはこれらのパケットをバックログキューに保存します。カーネルは、CPU コアごとに個別のキューを維持します。

コアのバックログキューがいっぱいの場合、カーネルは、**netif_receive_skb()** カーネル関数がこのキューに割り当てるそれ以降の受信パケットをすべてドロップします。サーバーに速度が 10 Gbps 以上のネットワークアダプターまたは複数の 1 Gbps アダプターが含まれている場合は、バックログキューのサイズをチューニングしてこの問題を回避します。

前提条件

- 速度が 10 Gbps 以上、または複数の 1 Gbps ネットワークアダプター

手順

1. バックログキューのチューニングが必要かどうかを判断し、**/proc/net/softnet_stat** ファイル内のカウンターを表示します。

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
221951548  0      0  0  0  0  0  0  0  0  0  0
192058677  18862  0  0  0  0  0  0  0  0  0  1
455324886  0      0  0  0  0  0  0  0  0  0  2
...
```

   この **awk** コマンドは、**/proc/net/softnet_stat** の値を 16 進形式から 10 進形式に変換し、表形式で表示します。各行は、コア 0 から始まる CPU コアを表します。

   関連する列は次のとおりです。

   - 最初の列: 受信フレームの総数
   - 2 番目の列: バックログキューがいっぱいであるためにドロップされたフレームの数
   - 最後の列: CPU コア番号

2. **/proc/net/softnet_stat** ファイルの 2 番目の列の値が時間の経過とともに増加する場合は、バックログキューのサイズを増やします。

   a. 現在のバックログキューのサイズを表示します。

```
# sysctl net.core.netdev_max_backlog
net.core.netdev_max_backlog = 1000
```

   b. 次の内容を含む **/etc/sysctl.d/10-netdev_max_backlog.conf** ファイルを作成します。

```
net.core.netdev_max_backlog = 2000
```

   net.core.netdev_max_backlog パラメーターを現在の値の 2 倍に設定します。

   c. **/etc/sysctl.d/10-netdev_max_backlog.conf** ファイルから設定をロードします。

```
# sysctl -p /etc/sysctl.d/10-netdev_max_backlog.conf
```

検証

- **/proc/net/softnet_stat** ファイルの 2 番目の列を監視します。

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
```

それでも値が増加する場合は、**net.core.netdev_max_backlog** 値を再度 2 倍にします。パケットドロップカウンターが増加しなくなるまで、このプロセスを繰り返します。

34.1.3. NIC の送信キューの長さを増やして送信エラーの数を減らす

カーネルは、パケットを送信する前に送信キューにパケットを格納します。デフォルトの長さ (1000 パケット) は、通常、10 Gbps には十分です。また、多くの場合、40 Gbps ネットワークにも十分です。ただし、より高速なネットワークの場合、またはアダプターで送信エラーが増加する場合は、キューの長さを増やしてください。

手順

1. 現在の送信キューの長さを表示します。

```
# ip -s link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 1000
...
```

この例では、**enp1s0** インターフェイスの送信キューの長さ (**qlen**) は **1000** です。

2. ネットワークインターフェイスのソフトウェア送信キューのドロップされたパケットカウンターを監視します。

```
# tc -s qdisc show dev enp1s0
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5ms interval
100ms memory_limit 32Mb ecn drop_batch 64
 Sent 16889923 bytes 426862765 pkt (dropped 191980, overlimits 0 requeues 2)
...
```

3. 送信エラー数が多い、または増加している場合は、送信キューの長さをより長く設定します。

   a. このインターフェイスを使用する NetworkManager 接続プロファイルを特定します。

```
# nmcli connection show
NAME               UUID                                  TYPE      DEVICE
Example-Connection  a5eb6490-cc20-3668-81f8-0314a27f3f75  ethernet  enp1s0
```

   b. 次の内容で **/etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up** NetworkManager ディスパッチャースクリプトを作成します。

```
#!/bin/bash
# Set TX queue length on enp1s0 to 2000

if [ "$1" == "enp1s0" ] && [ "$2" == "up" ] ; then
    ip link set dev enp1s0 txqueuelen 2000
fi
```

c. **/etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up** ファイルに実行可能ビットを設定します。

```
# chmod +x /etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up
```

d. 変更を適用します。

```
# nmcli connection up Example-Connection
```

検証

1. 送信キューの長さを表示します。

```
# ip -s link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 2000
...
```

2. ドロップされたパケットカウンターを監視します。

```
# tc -s qdisc show dev enp1s0
```

dropped カウンターがまだ増加する場合は、送信キューの長さを再度 2 倍にします。カウンターが増加しなくなるまで、このプロセスを繰り返します。

34.2. IRQ バランシングのチューニング

マルチコアホストでは、Red Hat Enterprise Linux が割り込みキュー (IRQ) のバランスをとり、CPU コア全体に割り込みを分散するようにすることで、パフォーマンスを向上させることができます。

34.2.1. 割り込みと割り込みハンドラー

ネットワークインターフェイスコントローラー (NIC) が受信データを受信すると、Direct Memory Access (DMA) を使用してそのデータをカーネルバッファーにコピーします。次に、NIC はハード割り込みをトリガーして、このデータについてカーネルに通知します。これらの割り込みは、すでに別のタスクに割り込んでおり、ハンドラー自体は割り込むことができないため、最小限の作業を行う割り込みハンドラーによって処理されます。ハード割り込みは、特にカーネルロックを使用する場合、CPU 使用率の点でコストがかかる可能性があります。

その後、ハード割り込みハンドラーは、パケット受信の大部分をソフトウェア割り込み要求 (SoftIRQ) プロセスに任せます。カーネルは、これらのプロセスをより公平にスケジュールできます。

例34.1 ハードウェア割り込みの表示

カーネルは、割り込みカウンターを **/proc/interrupts** ファイルに保存します。**enp1s0** などの特定の NIC のカウンターを表示するには、次のように入力します。

```
# egrep "CPU|enp1s0" /proc/interrupts
       CPU0    CPU1    CPU2    CPU3    CPU4   CPU5
105:  141606       0       0       0       0      0  IR-PCI-MSI-edge      enp1s0-rx-0
106:       0  141091       0       0       0      0  IR-PCI-MSI-edge      enp1s0-rx-1
```

```
107:    2    0   163785    0    0    0  IR-PCI-MSI-edge     enp1s0-rx-2
108:    3    0        0  194370    0    0  IR-PCI-MSI-edge     enp1s0-rx-3
109:    0    0        0    0    0    0  IR-PCI-MSI-edge     enp1s0-tx
```

各キューには、最初の列に割り込みベクトルが割り当てられています。カーネルは、システムの起動時、またはユーザーが NIC ドライバーモジュールをロードしたときに、これらのベクトルを初期化します。各受信 (**RX**) キューと送信 (**TX**) キューには、どの NIC またはキューから割り込みが発生しているかを割り込みハンドラーに通知する固有のベクトルが割り当てられます。列は、各 CPU コアの受信割り込みの数を表します。

34.2.2. ソフトウェア割り込み要求

ソフトウェア割り込み要求 (SoftIRQ) は、ネットワークアダプターの受信リングバッファーをクリアします。カーネルは、他のタスクが割り込みされない時間に SoftIRQ ルーチンが実行されるようにスケジュールします。Red Hat Enterprise Linux では、**ksoftirqd/cpu-number** という名前のプロセスがこれらのルーチンを実行し、ドライバー固有のコード関数を呼び出します。

各 CPU コアの SoftIRQ カウンターを監視するには、次のように入力します。

```
# watch -n1 'egrep "CPU|NET_RX|NET_TX" /proc/softirqs'
                CPU0       CPU1   CPU2       CPU3 CPU4    CPU5       CPU6  CPU7
    NET_TX:    49672      52610  28175      97288      12633   19843      18746     220689
    NET_RX:       96       1615       789        46        31    1735       1315     470798
```

このコマンドは出力を動的に更新します。**Ctrl**+**C** を押して出力に割り込みます。

34.2.3. NAPI ポーリング

New API (NAPI) は、受信ネットワークパケットの効率を向上させるためのデバイスドライバーパケット処理フレームワークの拡張機能です。ハード割り込みは、通常、カーネル空間からユーザー空間へのコンテキストの切り替えを引き起こし、またその逆のコンテキストの切り替えも引き起こし、それ自体を中断することができないため、コストがかかります。割り込み結合を行っても、割り込みハンドラーは CPU コアを完全に独占します。NAPI を使用すると、ドライバーは、パケットを受信するたびにカーネルによってハード割り込みされるのではなく、ポーリングモードを使用できます。

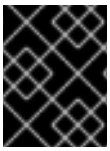
通常の操作では、カーネルは最初のハード割り込みを発行し、続いて NAPI ルーチンを使用してネットワークカードをポーリングするソフト割り込み要求 (SoftIRQ) ハンドラーを発行します。SoftIRQ が CPU コアを独占しないようにするために、ポーリングルーチンには、SoftIRQ が消費できる CPU 時間を決定するバジェットがあります。SoftIRQ ポーリングルーチンが完了すると、カーネルはルーチンを終了し、後で再度実行してネットワークカードからパケットを受信するプロセスを繰り返すようにスケジュールします。

34.2.4. irqbalance サービス

Non-Uniform Memory Access (NUMA) アーキテクチャーを備えたシステムと備えていないシステムの両方で、**irqbalance** サービスは、システム条件に基づいて CPU コア間で効果的に割り込みのバランスをとります。**irqbalance** サービスはバックグラウンドで実行され、10 秒ごとに CPU 負荷を監視します。CPU の負荷が高すぎる場合、このサービスは割り込みを他の CPU コアに移動します。その結果、システムのパフォーマンスが向上し、負荷がより効率的に処理されます。

irqbalance が実行されていない場合、通常は CPU コア 0 がほとんどの割り込みを処理します。中程度の負荷でも、この CPU コアはシステム内のすべてのハードウェアのワークロードを処理しようとしてビジーになる可能性があります。その結果、割り込みまたは割り込みベースの作業ができなかったり、

遅延したりする可能性があります。これにより、ネットワークやストレージのパフォーマンスの低下、パケットロス、その他の問題が発生する可能性があります。



重要

irqbalance を無効にすると、ネットワークのスループットに悪影響を及ぼす可能性があります。

CPU コアが 1 つしかないシステムでは、**irqbalance** サービスは何のメリットも提供せず、自動的に終了します。

デフォルトでは、**irqbalance** サービスは有効になっており、Red Hat Enterprise Linux 上で実行されています。サービスを無効にした場合に再度有効にするには、次のように入力します。

```
# systemctl enable --now irqbalance
```

関連情報

- ソリューション記事 Do we need irqbalance?
- ソリューション記事 How should I configure network interface IRQ channels?

34.2.5. CPU 上で SoftIRQ を実行できる時間の増加

SoftIRQ の実行時間が十分でない場合、受信データの速度が、バッファーを十分な速さでドレインするカーネルの能力を超える可能性があります。その結果、ネットワークインターフェイスコントローラー (NIC) のバッファーがオーバーフローし、パケットが失われます。

softirqd プロセスが 1 回の NAPI ポーリングサイクルでインターフェイスからすべてのパケットを取得できなかった場合、それは SoftIRQ に十分な CPU 時間がないことを示しています。これは、10 Gbps 以上の高速 NIC を備えたホストに当てはまる可能性があります。**net.core.netdev_budget** および **net.core.netdev_budget_usecs** カーネルパラメーターの値を増やすと、**softirqd** がポーリングサイクルで処理できる時間とパケット数を制御できます。

手順

1. **net.core.netdev_budget** パラメーターのチューニングが必要かどうかを判断するには、**/proc/net/softnet_stat** ファイル内のカウンターを表示します。

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
221951548  0  0      0  0  0  0  0  0  0  0  0  0
192058677  0  20380  0  0  0  0  0  0  0  0  0  1
455324886  0  0      0  0  0  0  0  0  0  0  0  2
...
```

この **awk** コマンドは、**/proc/net/softnet_stat** の値を 16 進形式から 10 進形式に変換し、表形式で表示します。各行は、コア 0 から始まる CPU コアを表します。

関連する列は次のとおりです。

- 最初の列: 受信フレームの総数

- 3 番目の列: 1 回の NAPI ポーリングサイクルでインターフェイスからすべてのパケットを取得できなかった **softirqd** プロセスの回数。
- 最後の列: CPU コア番号

2. **/proc/net/softnet_stat** ファイルの 3 番目の列のカウンターが、時間の経過とともに増加する場合は、システムをチューニングします。

   a. **net.core.netdev_budget_usecs** および **net.core.netdev_budget** パラメーターの現在の値を表示します。

   ```
   # sysctl net.core.netdev_budget_usecs net.core.netdev_budget
   net.core.netdev_budget_usecs = 2000
   net.core.netdev_budget = 300
   ```

   これらの設定を使用すると、**softirqd** プロセスは、1 回のポーリングサイクルで、NIC から最大 300 個のメッセージを処理するのに最大 2000 マイクロ秒あります。ポーリングは、どの条件が最初に満たされたかに基づいて終了します。

   b. 次の内容を含む **/etc/sysctl.d/10-netdev_budget.conf** ファイルを作成します。

   ```
   net.core.netdev_budget = 600
   net.core.netdev_budget_usecs = 4000
   ```

   パラメーターを現在の値の 2 倍に設定します。

   c. **/etc/sysctl.d/10-netdev_budget.conf** ファイルから設定をロードします。

   ```
   # sysctl -p /etc/sysctl.d/10-netdev_budget.conf
   ```

検証

- **/proc/net/softnet_stat** ファイルの 3 番目の列を監視します。

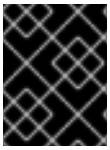
  ```
  # awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
  /proc/net/softnet_stat | column -t
  ```

  それでも値が増加する場合は、**net.core.netdev_budget_usecs** と **net.core.netdev_budget** をより高い値に設定します。カウンターが増加しなくなるまで、このプロセスを繰り返します。

34.3. ネットワーク遅延の改善

CPU 電力管理機能により、時間を考慮する必要があるアプリケーション処理に望ましくない遅延が発生する可能性があります。これらの電源管理機能の一部またはすべてを無効にして、ネットワーク遅延を改善できます。

たとえば、サーバーの負荷が高いときよりもアイドル状態のときの遅延が高い場合は、CPU 電源管理設定が遅延に影響を与える可能性があります。



重要

CPU 電源管理機能を無効にすると、電力消費量が増加し、熱損失が発生する可能性があります。

34.3.1. CPU の電源状態がネットワーク遅延に与える影響

CPU の消費状態 (C-状態) は、コンピューターの消費電力を最適化し、削減します。C-状態には、C0 から始まる番号が付けられます。C0 では、プロセッサーはフルパワーで動作し、実行しています。C1 では、プロセッサーはフルパワーで動作していますが、実行されていません。C-状態の番号が大きいほど、CPU がオフにするコンポーネントの数が多くなります。

CPU コアがアイドル状態になると常に、内蔵の省電力ロジックが介入し、さまざまなプロセッサーコンポーネントをオフにして、コアを現在の C-状態から上位の C-状態に移行しようとします。CPU コアがデータを処理する必要がある場合、Red Hat Enterprise Linux (RHEL) はプロセッサーに割り込みを送信してコアをウェイクアップし、C-状態を C0 に戻します。

ディープ C-状態から C0 に戻るには、プロセッサーのさまざまなコンポーネントの電源を再度オンにするため、時間がかかります。マルチコアシステムでは、多くのコアが同時にアイドル状態になり、より深い C-状態になることもあります。RHEL がそれらを同時にウェイクアップしようとすると、すべてのコアがディープ C-状態から戻る間に、カーネルが大量のプロセッサー間割り込み (IPI) を生成する可能性があります。割り込みの処理中にロックが必要となるため、システムはすべての割り込みの処理中にしばらく停止する可能性があります。これにより、イベントに対するアプリケーションの応答が大幅に遅延する可能性があります。

例34.2 コアごとの C-状態時間の表示

PowerTOP アプリケーションの **Idle Stats** ページには、CPU コアが各 C-状態で費やした時間が表示されます。

```
          Pkg(HW)  |          Core(HW) |          CPU(OS) 0   CPU(OS) 4
                   |                   | C0 active   2.5%       2.2%
                   |                   | POLL        0.0%    0.0 ms  0.0%    0.1 ms
                   |                   | C1          0.1%    0.2 ms  0.0%    0.1 ms
C2 (pc2)   63.7%   |                   |
C3 (pc3)    0.0%   | C3 (cc3)    0.1%   | C3          0.1%    0.1 ms  0.1%    0.1 ms
C6 (pc6)    0.0%   | C6 (cc6)    8.3%   | C6          5.2%    0.6 ms  6.0%    0.6 ms
C7 (pc7)    0.0%   | C7 (cc7)   76.6%   | C7s         0.0%    0.0 ms  0.0%    0.0 ms
C8 (pc8)    0.0%   |                   | C8          6.3%    0.9 ms  5.8%    0.8 ms
C9 (pc9)    0.0%   |                   | C9          0.4%    3.7 ms  2.2%    2.2 ms
C10 (pc10)  0.0%   |                   |
                   |                   | C10        80.8%    3.7 ms 79.4%    4.4 ms
                   |                   | C1E         0.1%    0.1 ms  0.1%    0.1 ms
...
```

関連情報

- PowerTOP を使用した電力消費の管理

34.3.2. EFI ファームウェアの C-状態設定

EFI ファームウェアを備えたほとんどのシステムでは、個々の消費状態 (C-状態) を有効または無効にすることができます。ただし、Red Hat Enterprise Linux (RHEL) では、アイドルドライバーによって、カーネルがファームウェアの設定を使用するかどうかが決定されます。

- **intel_idle**: これは、Intel CPU を搭載したホスト上のデフォルトのドライバーであり、EFI ファームウェアからの C-状態設定を無視します。

- **acpi_idle**: RHEL は、Intel 以外のベンダーの CPU を搭載したホスト上、および **intel_idle** が無効になっている場合に、このドライバーを使用します。デフォルトでは、**acpi_idle** ドライバーは EFI ファームウェアの C-状態設定を使用します。

関連情報

- **kernel-doc** パッケージが提供する **/usr/share/doc/kernel-doc-<version>/Documentation/admin-guide/pm/cpuidle.rst**

34.3.3. カスタム TuneD プロファイルを使用した C-状態の無効化

TuneD サービスは、カーネルの Power Management Quality of Service (**PMQOS**) インターフェイスを使用して、消費状態 (C 状態) のロックを設定します。カーネルアイドルドライバーは、このインターフェイスと通信して C-状態を動的に制限できます。これにより、管理者がカーネルコマンドラインパラメーターを使用して、C-状態の最大値をハードコーディングする必要がなくなります。

前提条件

- **tuned** パッケージがインストールされている。
- **tuned** サービスが有効化され、実行されている。

手順

1. アクティブなプロファイルを表示します。

```
# tuned-adm active
Current active profile: network-latency
```

2. カスタム TuneD プロファイル用のディレクトリーを作成します。

```
# mkdir /etc/tuned/network-latency-custom/
```

3. 次の内容を含む **/etc/tuned/network-latency-custom/tuned.conf** ファイルを作成します。

```
[main]
include=network-latency

[cpu]
force_latency=cstate.id:1|2
```

このカスタムプロファイルは、**network-latency** プロファイルからすべての設定を継承します。**force_latency** TuneD パラメーターは、遅延をマイクロ秒 (µs) 単位で指定します。C-状態のレイテンシーが指定された値よりも高い場合、Red Hat Enterprise Linux のアイドルドライバーは CPU がより高い C-状態に移行するのを防ぎます。**force_latency=cstate.id:1|2** を指定すると、TuneD は最初に **/sys/devices/system/cpu/cpu_<number>_/cpuidle/state_<cstate.id>_/** ディレクトリーが存在するかどうかを確認します。この場合、TuneD はこのディレクトリー内の **latency** ファイルからレイテンシー値を読み取ります。ディレクトリーが存在しない場合は、TuneD はフォールバック値として 2 マイクロ秒を使用します。

4. **network-latency-custom** プロファイルをアクティブ化します。

```
# tuned-adm profile network-latency-custom
```

関連情報

- TuneD を使い始める
- TuneD プロファイルのカスタマイズ

34.3.4. カーネルコマンドラインオプションを使用した C-状態の無効化

processor.max_cstate および **intel_idle.max_cstat** カーネルコマンドラインパラメーターは、CPU コアが使用できる最大消費状態 (C-状態) を設定します。たとえば、パラメーターを **1** に設定すると、CPU は C1 より下の C-状態を要求しなくなります。

この方法を使用して、ホスト上のアプリケーションの遅延が C-状態の影響を受けているかどうかをテストします。特定の状態をハードコーディングしないようにするには、より動的なソリューションの使用を検討してください。Disabling C-states by using a custom TuneD profile を参照してください。

前提条件

- **tuned** サービスが実行されていないか、C-状態設定を更新しないように設定されています。

手順

1. システムが使用するアイドル状態のドライバーを表示します。

```
# cat /sys/devices/system/cpu/cpuidle/current_driver
intel_idle
```

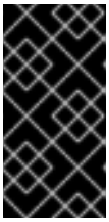
2. ホストが **intel_idle** ドライバーを使用している場合は、**intel_idle.max_cstate** カーネルパラメーターを設定して、CPU コアが使用できる最高の C-状態を定義します。

```
# grubby --update-kernel=ALL --args="intel_idle.max_cstate=0"
```

intel_idle.max_cstate=0 を設定すると、**intel_idle** ドライバーが無効になります。その結果、カーネルは、EFI ファームウェアで設定された C-状態値を使用する **acpi_idle** ドライバーを使用します。このため、これらの C-状態設定をオーバーライドするには、**processor.max_cstate** も設定します。

3. すべてのホストで、CPU ベンダーから独立して、CPU コアが使用できる最高の C-状態を設定します。

```
# grubby --update-kernel=ALL --args="processor.max_cstate=0"
```



重要

intel_idle.max_cstate=0 に加えて **processor.max_cstate=0** を設定すると、**acpi_idle** ドライバーは **processor.max_cstate** の値をオーバーライドして **1** に設定します。その結果、**processor.max_cstate=0 intel_idle.max_cstate=0** の場合、カーネルが使用する最高の C-状態は C0 ではなく C1 になります。

4. 変更を有効にするためにホストを再起動します。

```
# reboot
```

検証

検証

1. 最大の C-状態を表示します。

```
# cat /sys/module/processor/parameters/max_cstate
1
```

2. ホストが **intel_idle** ドライバーを使用している場合は、最大の C-状態を表示します。

```
# cat /sys/module/intel_idle/parameters/max_cstate
0
```

関連情報

- What are CPU "C-states" and how to disable them if needed?
- **kernel-doc** パッケージが提供する **/usr/share/doc/kernel-doc-<version>/Documentation/admin-guide/pm/cpuidle.rst**

34.4. 大量の連続したデータストリームのスループットの向上

IEEE 802.3 標準によれば、仮想ローカルエリアネットワーク (VLAN) タグのないデフォルトのイーサネットフレームの最大サイズは 1518 バイトです。これらの各フレームには 18 バイトのヘッダーが含まれて、ペイロード用に 1500 バイトが残されます。したがって、サーバーがネットワーク経由で送信するデータの 1500 バイトごとに、18 バイト (1.2%) のイーサネットフレームヘッダーがオーバーヘッドとなって送信されます。レイヤー 3 およびレイヤー 4 プロトコルのヘッダーにより、パケットあたりのオーバーヘッドがさらに増加します。

ネットワーク上のホストが、多数の連続したデータストリーム (バックアップサーバーや多数の巨大なファイルをホストするファイルサーバーなど) を頻繁に送信する場合は、オーバーヘッドを節約するためにジャンボフレームの採用を検討してください。ジャンボフレームは、標準のイーサネットペイロードサイズである 1500 バイトよりも大きな最大伝送単位 (MTU) を持つ非標準フレームです。たとえば、最大許容 MTU が 9000 バイトのペイロードでジャンボフレームを設定すると、各フレームのオーバーヘッドは 0.2% に減少します。

ネットワークとサービスによっては、クラスターのストレージバックエンドなど、ネットワークの特定の部分でのみジャンボフレームを有効にすることが有益な場合があります。これにより、パケットの断片化が回避されます。

34.4.1. ジャンボフレームを設定する前の考慮事項

ネットワーク内のハードウェア、アプリケーション、サービスに応じて、ジャンボフレームはさまざまな影響を与える可能性があります。ジャンボフレームを有効にすることがシナリオにメリットをもたらすかどうかを慎重に考慮して決定してください。

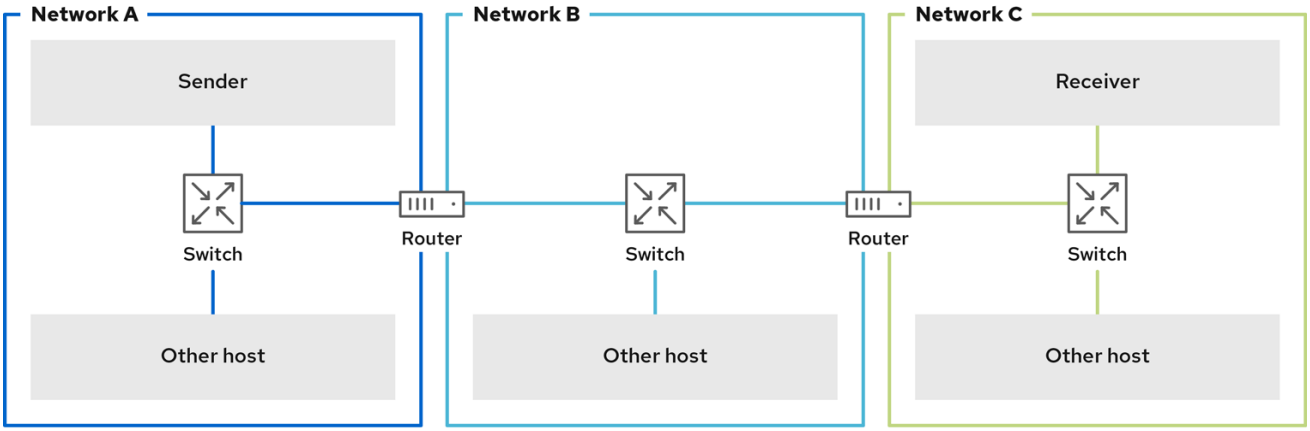
前提条件

伝送パス上のすべてのネットワークデバイスはジャンボフレームをサポートし、同じ最大伝送単位 (MTU) サイズを使用する必要があります。逆の場合は、次の問題に直面する可能性があります。

- ドロップされたパケット
- 断片化されたパケットが原因の遅延が大きい

- 断片化によるパケットロスのリスクの増加。たとえば、ルーターが 1 つの 9000 バイトフレームを 6 つの 1500 バイトフレームに分割し、それらの 1500 バイトフレームのいずれかが失われた場合、フレーム全体は再設定できないため失われます。

次の図では、ネットワーク A のホストがネットワーク C のホストにパケットを送信する場合、3 つのサブネット内のすべてのホストが同じ MTU を使用する必要があります。



ジャンボフレームの利点

- より高いスループット: 各フレームにはより多くのユーザーデータが含まれますが、プロトコルオーバーヘッドは固定されています。
- CPU 使用率の低下: ジャンボフレームにより発生する割り込みが少なくなるため、CPU サイクルが節約されます。

ジャンボフレームの欠点

- 遅延が大きい: フレームが大きいと、後続のパケットが遅延します。
- メモリーバッファーの使用量の増加: フレームが大きくなると、バッファーキューメモリーがより早くいっぱいになる可能性があります。

34.4.2. 既存の NetworkManager 接続プロファイルでの MTU の設定

ネットワークでデフォルトとは異なる最大伝送単位 (MTU) が必要な場合は、対応する NetworkManager 接続プロファイルでこの設定を設定できます。

ジャンボフレームは、1500 - 9000 バイトのペイロードを持つネットワークパケットです。同じブロードキャストドメイン内のすべてのデバイスは、これらのフレームをサポートする必要があります。

前提条件

- ブロードキャストドメイン内のすべてのデバイスが同じ MTU を使用している。
- ネットワークの MTU を把握している。
- 不一致の MTU を使用するネットワークの接続プロファイルがすでに設定されている。

手順

1. オプション: 現在の MTU を表示します。

```
# ip link show
...
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
...
```

2. オプション: NetworkManager 接続プロファイルを表示します。

```
# nmcli connection show
NAME     UUID                                  TYPE      DEVICE
Example  f2f33f29-bb5c-3a07-9069-be72eaec3ecf  ethernet  enp1s0
...
```

3. 不一致の MTU を使用してネットワークへの接続を管理するプロファイルに MTU を設定します。

```
# nmcli connection modify Example mtu 9000
```

4. 接続を再度アクティベートします。

```
# nmcli connection up Example
```

検証

1. MTU 設定を表示します。

```
# ip link show
...
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
...
```

2. 伝送パス上にパケットを断片化するホストがないことを確認します。

- 受信側で、カーネルの IP 再構築統計を表示します。

```
# nstat -az IpReasm*
#kernel
IpReasmTimeout 0 0.0
IpReasmReqds 0 0.0
IpReasmOKs 0 0.0
IpReasmFails 0 0.0
```

  カウンターが **0** を返した場合、パケットは再構築されていません。

- 送信者側で、prohibit-fragmentation-bit を指定して ICMP リクエストを送信します。

```
# ping -c1 -Mdo -s 8972 destination_host
```

コマンドが成功した場合、パケットは断片化されていません。

-s パケットサイズオプションの値を次のように計算します。MTU サイズ - 8 バイト ICMP ヘッダー - 20 バイト IPv4 ヘッダー = パケットサイズ

34.5. 高スループットのための TCP 接続のチューニング

Red Hat Enterprise Linux で TCP 関連の設定をチューニングして、スループットを向上させ、レイテンシーを短縮し、パケットロスなどの問題を防止します。

34.5.1. iperf3 を使用した TCP スループットのテスト

iperf3 ユーティリティーは、サーバーモードとクライアントモードを提供し、2 つのホスト間のネットワークスループットテストを実行します。

注記

アプリケーションのスループットは、アプリケーションが使用するバッファーサイズなどの多くの要因に依存します。したがって、**iperf3** などのテストユーティリティーで測定された結果は、実稼働ワークロード下のサーバー上のアプリケーションの結果とは大幅に異なる可能性があります。

前提条件

- **iperf3** パッケージはクライアントとサーバーの両方にインストールされている。
- どちらかのホスト上の他のサービスによって、テスト結果に大きな影響を与えるネットワークトラフィックが発生することはない。
- 速度が 40 Gbps 以上の接続の場合、ネットワークカードは Accelerated Receive Flow Steering (ARFS) をサポートし、この機能はインターフェイスで有効になっている。

手順

1. オプション: サーバーとクライアントの両方のネットワークインターフェイスコントローラー (NIC) の最大ネットワーク速度を表示します。

```
# ethtool enp1s0 | grep "Speed"
  Speed: 100000Mb/s
```

2. サーバーの場合:

   a. **firewalld** サービスでデフォルトの **iperf3** TCP ポート 5201 を一時的に開きます。

```
# firewall-cmd --add-port=5201/tcp
# firewall-cmd --reload
```

   b. **iperf3** をサーバーモードで起動します。

```
# iperf3 --server
```

   サービスは現在、受信クライアント接続を待機しています。

3. クライアントで以下を実行します。

a. スループットの測定を開始します。

```
# iperf3 --time 60 --zerocopy --client 192.0.2.1
```

- **--time <seconds>**: クライアントが送信を停止する時間を秒単位で定義します。このパラメーターを機能すると予想される値に設定し、後の測定で値を増やします。サーバーが、送信パス上のデバイスよりも速い速度で、またはクライアントが処理できるよりも速い速度でパケットを送信する場合、パケットがドロップされる可能性があります。

- **--zerocopy**: **write()** システムコールを使用する代わりに、ゼロコピーメソッドを有効にします。このオプションは、ゼロコピー対応アプリケーションをシミュレートする場合、または単一ストリームで 40 Gbps 以上に達する場合にのみ必要です。

- **--client <server>**: クライアントモードを有効にし、**iperf3** サーバーを実行するサーバーの IP アドレスまたは名前を設定します。

4. **iperf3** がテストを完了するまで待ちます。サーバーとクライアントの両方で統計が毎秒表示され、最後に概要が表示されます。たとえば、以下はクライアントに表示される概要です。

```
[ ID] Interval           Transfer     Bitrate         Retr
[  5] 0.00-60.00  sec  101 GBytes   14.4 Gbits/sec   0   sender
[  5] 0.00-60.04  sec  101 GBytes   14.4 Gbits/sec       receiver
```

この例では、平均ビットレートは 14.4 Gbps でした。

5. サーバーの場合:

a. **Ctrl**+**C** を押して **iperf3** サーバーを停止します。

b. **firewalld** で TCP ポート 5201 を閉じます。

```
# firewall-cmd --remove-port=5201/tcp
# firewall-cmd --reload
```

関連情報

- **iperf3(1)** man ページ

34.5.2. システム全体の TCP ソケットバッファー設定

ソケットバッファーには、カーネルが受信したデータ、または送信する必要があるデータが一時的に保存されます。

- 読み取りソケットバッファーには、カーネルが受信したがアプリケーションがまだ読み取っていないパケットが保持されます。

- 書き込みソケットバッファーには、アプリケーションがバッファーに書き込んだものの、カーネルがまだ IP スタックとネットワークドライバーに渡していないパケットが保持されます。

TCP パケットが大きすぎてバッファーサイズを超えている場合、またはパケットの送受信速度が速すぎる場合、カーネルはデータがバッファーから削除されるまで、新しい受信 TCP パケットをドロップします。この場合、ソケットバッファーを増やすことでパケットロスを防ぐことができます。

net.ipv4.tcp_rmem (読み取り) と **net.ipv4.tcp_wmem** (書き込み) の両方のソケットバッファーカーネル設定には、次の 3 つの値が含まれます。

```
net.ipv4.tcp_rmem = 4096  131072  6291456
net.ipv4.tcp_wmem = 4096  16384   4194304
```

表示される値はバイト単位であり、Red Hat Enterprise Linux はそれらを次の方法で使用します。

- 最初の値は最小バッファーサイズです。新しいソケットのサイズを小さくすることはできません。
- 2 番目の値はデフォルトのバッファーサイズです。アプリケーションがバッファーサイズを設定しない場合、これがデフォルト値になります。
- 3 番目の値は、自動的にチューニングされるバッファーの最大サイズです。アプリケーションで **setsockopt()** 関数を **SO_SNDBUF** ソケットオプションとともに使用すると、この最大バッファーサイズが無効になります。

net.ipv4.tcp_rmem および **net.ipv4.tcp_wmem** パラメーターは、IPv4 プロトコルと IPv6 プロトコルの両方のソケットサイズを設定することに注意してください。

34.5.3. システム全体の TCP ソケットバッファーの増加

システム全体の TCP ソケットバッファーには、カーネルが受信したデータ、または送信する必要があるデータが一時的に保存されます。**net.ipv4.tcp_rmem** (読み取り) と **net.ipv4.tcp_wmem** (書き込み) の両方のソケットバッファーカーネル設定には、それぞれ最小値、デフォルト値、および最大値の 3 つの設定が含まれています。

重要

バッファーサイズを大きすぎる設定にすると、メモリーが無駄に消費されます。各ソケットはアプリケーションが要求するサイズに設定でき、カーネルはこの値を 2 倍にします。たとえば、アプリケーションが 256 KiB のソケットバッファーサイズを要求し、100 万個のソケットを開く場合、システムは潜在的なソケットバッファースペースとしてのみ最大 512 GB RAM (512 KiB x 100 万) を使用できます。

さらに、最大バッファーサイズの値が大きすぎると、レイテンシーが長くなる可能性があります。

前提条件

- かなりの割合で TCP パケットがドロップされました。

手順

1. 接続の遅延を決定します。たとえば、クライアントからサーバーに ping を実行して、平均ラウンドトリップ時間 (RTT) を測定します。

```
# ping -c 10 server.example.com
...
--- server.example.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9014ms
rtt min/avg/max/mdev = 117.208/117.056/119.333/0.616 ms
```

この例では、レイテンシーは 117 ミリ秒です。

2. 次の式を使用して、チューニングするトラフィックの帯域幅遅延積 (BDP) を計算します。

```
connection speed in bytes * latency in ms = BDP in bytes
```

たとえば、レイテンシーが 117 ミリ秒の 10 Gbps 接続の BDP を計算するには、次のようにします。

```
(10 * 1000 * 1000 * 1000 / 8) * 117 = 10683760 bytes
```

3. **/etc/sysctl.d/10-tcp-socket-buffers.conf** ファイルを作成し、要件に基づいて、最大読み取りバッファーサイズまたは書き込みバッファーサイズ、またはその両方を設定します。

```
net.ipv4.tcp_rmem = 4096 262144 21367520
net.ipv4.tcp_wmem = 4096 24576 21367520
```

値をバイト単位で指定します。環境に最適化された値を特定するには、以下の大体の目安を参考にしてください。

- デフォルトのバッファーサイズ (2 番目の値): この値をわずかに増やすか、最大でも **524288** (512 KiB) に設定します。デフォルトのバッファーサイズが大きすぎると、バッファーの崩壊が発生し、その結果、遅延が急増する可能性があります。
- 最大バッファーサイズ (3 番目の値): 多くの場合、BDP の 2 倍から 3 倍の値で十分です。

4. **/etc/sysctl.d/10-tcp-socket-buffers.conf** ファイルから設定をロードします。

```
# sysctl -p /etc/sysctl.d/10-tcp-socket-buffers.conf
```

5. より大きなソケットバッファーサイズを使用するようにアプリケーションを設定します。**net.ipv4.tcp_rmem** および **net.ipv4.tcp_wmem** パラメーターの 3 番目の値は、アプリケーションの **setsockopt()** 関数が要求できる最大バッファーサイズを定義します。
詳細については、アプリケーションのプログラミング言語のドキュメントを参照してください。アプリケーションの開発者ではない場合は、開発者にお問い合わせください。

6. **net.ipv4.tcp_rmem** または **net.ipv4.tcp_wmem** パラメーターの 2 番目の値を変更した場合は、新しい TCP バッファーサイズを使用するようにアプリケーションを再起動します。
3 番目の値のみを変更した場合は、自動チューニングによってこれらの設定が動的に適用されるため、アプリケーションを再起動する必要はありません。

検証

1. オプション: iperf3 を使用して TCP スループットをテストします。

2. パケットドロップが発生したときに使用したのと同じ方法を使用して、パケットドロップ統計を監視します。
パケットドロップが依然として発生するが、レートが低い場合は、バッファーサイズをさらに増やします。

関連情報

- ソリューション記事 What are the implications of changing socket buffer sizes?
- **tcp(7)** man ページ

- **socket(7)** man ページ

34.5.4. TCP ウィンドウのスケーリング

Red Hat Enterprise Linux でデフォルトで有効になっている TCP ウィンドウスケーリング機能は、スループットを大幅に向上させる TCP プロトコルの拡張機能です。

たとえば、ラウンドトリップ時間 (RTT) が 1.5 ミリ秒の 1 Gbps 接続の場合、次のようになります。

- TCP ウィンドウスケーリングを有効にすると、約 630 Mbps が現実的になります。
- TCP ウィンドウスケーリングを無効にすると、スループットは 380 Mbps に低下します。

TCP が提供する機能の 1 つはフロー制御です。フロー制御を使用すると、送信者は受信者が受信可能な量のデータを送信できますが、それ以上のデータは送信できません。これを達成するために、受信者は、送信者が送信できるデータの量である **window** 値をアドバタイズします。

TCP は当初、最大 64 KiB のウィンドウサイズをサポートしていましたが、帯域幅遅延積 (BDP) が高い場合、送信者が一度に 64 KiB を超えるサイズを送信できないため、この値が制限となります。高速接続では、一度に 64 KiB をはるかに超えるデータを転送できます。たとえば、システム間の遅延が 1 ms の 10 Gbps リンクでは、一度に 1 MiB を超えるデータが転送される可能性があります。ホストが 64 KiB のみを送信し、他のホストがその 64 KiB を受信するまで一時停止する場合、非効率的になります。

このボトルネックを解消するために、TCP ウィンドウスケーリング拡張機能を使用すると、TCP ウィンドウ値を左算術シフトしてウィンドウサイズを 64 KiB を超えて増やすことができます。たとえば、最大ウィンドウ値 **65535** は左に 7 桁シフトし、ウィンドウサイズは 8 MiB 近くになります。これにより、一度により多くのデータを転送できるようになります。

TCP ウィンドウスケーリングは、すべての TCP 接続を開く 3 ウェイ TCP ハンドシェイク中にネゴシエートされます。この機能が動作するには、送信者と受信者の両方が TCP ウィンドウスケーリングをサポートしている必要があります。参加者の一方または両方がハンドシェイクでウィンドウスケーリング機能をアドバタイズしない場合、接続は元の 16 ビット TCP ウィンドウサイズの使用に戻ります。

デフォルトでは、TCP ウィンドウスケーリングは Red Hat Enterprise Linux で有効になっています。

```
# sysctl net.ipv4.tcp_window_scaling
net.ipv4.tcp_window_scaling = 1
```

サーバー上で TCP ウィンドウスケーリングが無効 (**0**) になっている場合は、設定した際と同じ方法で設定を元に戻します。

関連情報

- RFC 1323: TCP Extensions for High Performance
- ランタイム時のカーネルパラメーターの設定

34.5.5. TCP SACK がパケットドロップ率を下げる仕組み

Red Hat Enterprise Linux (RHEL) でデフォルトで有効になっている TCP Selective Acknowledgment (TCP SACK) 機能は、TCP プロトコルの拡張機能であり、TCP 接続の効率を高めます。

TCP 送信では、受信者は受信するパケットごとに ACK パケットを送信者に送信します。たとえば、クライアントは TCP パケット 1 - 10 をサーバーに送信しますが、パケット番号 5 と 6 が失われます。TCP SACK がないと、サーバーはパケット 7 - 10 をドロップし、クライアントは損失点からすべてのパ

ケットを再送信する必要があり、非効率的です。両方のホストで TCP SACK が有効になっている場合、クライアントは失われたパケット 5 と 6 のみを再送信する必要があります。

重要

TCP SACK を無効にするとパフォーマンスが低下し、TCP 接続の受信側でのパケットドロップ率が高くなります。

デフォルトでは、RHEL では TCP SACK が有効になっています。確認するには、以下を実行します。

```
# sysctl net.ipv4.tcp_sack
1
```

サーバー上で TCP SACK が無効 (**0**) になっている場合は、設定した際と同じ方法で設定を元に戻します。

関連情報

- RFC 2018: TCP Selective Acknowledgment Options
- ソリューション記事 Should I be concerned about a 0.05% packet drop rate?
- ランタイム時のカーネルパラメーターの設定

34.6. UDP 接続のチューニング

UDP トラフィックのスループットを向上させるために Red Hat Enterprise Linux のチューニングを開始する前に、現実的に想定することが重要です。UDP は単純なプロトコルです。TCP と比較すると、UDP にはフロー制御、輻輳制御、データ信頼性などの機能が含まれていません。このため、ネットワークインターフェイスコントローラー (NIC) の最大速度に近いスループットレートで、UDP 経由で信頼性の高い通信を実現することが困難になります。

34.6.1. パケットドロップの検出

カーネルがパケットをドロップできるネットワークスタックには複数のレベルがあります。Red Hat Enterprise Linux は、これらのレベルの統計を表示するためのさまざまなユーティリティーを提供します。潜在的な問題を特定するためにこれらを使用してください。

ごくわずかな割合のパケットがドロップされる場合は無視できることに注意してください。ただし、大幅な割合でパケットがドロップされる場合は、チューニング措置を検討してください。

注記

ネットワークスタックが受信トラフィックを処理できない場合、カーネルはネットワークパケットをドロップします。

手順

1. ネットワークインターフェイスコントローラー (NIC) がパケットをドロップするかどうかを特定します。
   a. NIC およびドライバー固有の統計を表示します。

```
# ethtool -S enp1s0
NIC statistics:
     ...
     rx_queue_0_drops: 17657
     ...
```

統計の名前と統計が利用可能かどうかは、NIC とドライバーによって異なります。

b. インターフェイス統計を表示します。

```
# ip -s link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
    RX:   bytes  packets errors dropped  missed   mcast_
    84697611107 56866482      0   10904       0       0
    TX:   bytes  packets errors dropped carrier collsns_
     5540028184  3722234       0       0       0       0
```

RX は、受信パケットの統計を表し、**TX** は送信パケットの統計を表します。

2. 小さすぎるソケットバッファーまたは遅いアプリケーション処理による UDP プロトコル固有のパケットドロップを特定します。

```
# nstat -az UdpSndbufErrors UdpRcvbufErrors
#kernel
UdpSndbufErrors          4    0.0
UdpRcvbufErrors    45716659    0.0
```

出力の 2 番目の列にはカウンターがリストされます。

関連情報

- ソリューション記事 RHEL network interface dropping packets
- ソリューション記事 Should I be concerned about a 0.05% packet drop rate?

34.6.2. iperf3 を使用した UDP スループットのテスト

iperf3 ユーティリティーは、サーバーモードとクライアントモードを提供し、2 つのホスト間のネットワークスループットテストを実行します。

注記

アプリケーションのスループットは、アプリケーションが使用するバッファーサイズなど、多くの要因に依存します。したがって、**iperf3** などのテストユーティリティーで測定された結果は、実稼働ワークロード下のサーバー上のアプリケーションの結果とは大きく異なる可能性があります。

前提条件

- **iperf3** パッケージはクライアントとサーバーの両方にインストールされている。

- 両方のホスト上の他のサービスによって、テスト結果に大きな影響を与えるネットワークトラフィックが発生することはない。
- オプション: サーバーとクライアントの両方で最大 UDP ソケットサイズを増やしている。詳細は、Increasing the system-wide UDP socket buffers を参照してください。

手順

1. オプション: サーバーとクライアントの両方のネットワークインターフェイスコントローラー (NIC) の最大ネットワーク速度を表示します。

```
# ethtool enp1s0 | grep "Speed"
  Speed: 10000Mb/s
```

2. サーバーの場合:
   a. UDP ソケット読み取りバッファーの最大サイズを表示し、値をメモします。

```
# sysctl net.core.rmem_max
net.core.rmem_max = 16777216
```

   表示される値はバイト単位です。

   b. **firewalld** サービスでデフォルトの **iperf3** ポート 5201 を一時的に開きます。

```
# firewall-cmd --add-port=5201/tcp --add-port=5201/udp
# firewall-cmd --reload
```

   iperf3 は、サーバー上の TCP ソケットのみを開くことに注意してください。クライアントが UDP を使用したい場合は、まずこの TCP ポートに接続し、次にサーバーが同じポート番号で UDP ソケットを開き、UDP トラフィックスループットテストを実行します。このため、ローカルファイアウォールで TCP プロトコルと UDP プロトコルの両方に対してポート 5201 を開く必要があります。

   c. **iperf3** をサーバーモードで起動します。

```
# iperf3 --server
```

   サービスは受信クライアント接続を待機するようになります。

3. クライアントで以下を実行します。
   a. クライアントがサーバーへの接続に使用するインターフェイスの最大伝送単位 (MTU) を表示し、その値をメモします。

```
# ip link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
state UP mode DEFAULT group default qlen 1000
...
```

   b. UDP ソケット書き込みバッファーの最大サイズを表示し、値をメモします。

```
# sysctl net.core.wmem_max
net.core.wmem_max = 16777216
```

表示される値はバイト単位です。

c. スループットの測定を開始します。

```
# iperf3 --udp --time 60 --window 16777216 --length 1472 --bitrate 2G --client
192.0.2.1
```

- **--udp**: テストには UDP プロトコルを使用します。
- **--time <seconds>**: クライアントが送信を停止する時間を秒単位で定義します。
- **--window <size>**: UDP ソケットのバッファーサイズを設定します。理想的には、クライアントとサーバーのサイズが同じです。それらが異なる場合は、このパラメーターを小さい方の値に設定します。クライアントでは **net.core.wmem_max**、サーバーでは **net.core.rmem_max** です。
- **--length <size>**: 読み取りおよび書き込みを行うバッファーの長さを設定します。このオプションを断片化されていない最大のペイロードに設定します。理想的な値は次のように計算します。MTU - IP ヘッダー (IPv4 の場合は 20 バイト、IPv6 の場合は 40 バイト) - 8 バイトの UDP ヘッダー。
- **--bitrate <rate>**: ビットレートをビット/秒単位で指定した値に制限します。2 Gbps の場合は **2G** など、単位を指定できます。
  このパラメーターを機能すると予想される値に設定し、後の測定で値を増やします。
  サーバーが、送信パス上のデバイスよりも速い速度で、またはクライアントが処理できるよりも速い速度でパケットを送信する場合、パケットがドロップされる可能性があります。
- **--client <server>**: クライアントモードを有効にし、**iperf3** サーバーを実行するサーバーの IP アドレスまたは名前を設定します。

4. **iperf3** がテストを完了するまで待ちます。サーバーとクライアントの両方で統計が毎秒表示され、最後に概要が表示されます。たとえば、以下はクライアントに表示される概要です。

```
[ ID] Interval       Transfer     Bitrate         Jitter    Lost/Total Datagrams
[ 5] 0.00-60.00 sec  14.0 GBytes   2.00 Gbits/sec  0.000 ms  0/10190216 (0%) sender
[ 5] 0.00-60.04 sec  14.0 GBytes   2.00 Gbits/sec  0.002 ms  0/10190216 (0%) receiver
```

この例では、平均ビットレートは 2 Gbps で、パケットの損失はありませんでした。

5. サーバーの場合:

a. **Ctrl**+**C** を押して **iperf3** サーバーを停止します。

b. **firewalld** のポート 5201 を閉じます。

```
# firewall-cmd --remove-port=5201/tcp --remove-port=5201/udp
# firewall-cmd --reload
```

関連情報

- **iperf3(1)** man ページ

34.6.3. UDP トラフィックスループットに対する MTU サイズの影響

アプリケーションが大きな UDP メッセージサイズを使用する場合、ジャンボフレームを使用するとスループットを改善できます。IEEE 802.3 標準によれば、仮想ローカルエリアネットワーク (VLAN) タグのないデフォルトのイーサネットフレームの最大サイズは 1518 バイトです。これらの各フレームには 18 バイトのヘッダーが含まれて、ペイロード用に 1500 バイトが残されます。したがって、サーバーがネットワーク経由で送信するデータの 1500 バイトごとに、18 バイト (1.2%) がオーバーヘッドになります。

ジャンボフレームは、標準のイーサネットペイロードサイズである 1500 バイトよりも大きな最大伝送単位 (MTU) を持つ非標準フレームです。たとえば、最大許容 MTU が 9000 バイトのペイロードでジャンボフレームを設定すると、各フレームのオーバーヘッドは 0.2% に減少します。

重要

伝送パス上のすべてのネットワークデバイスと関連するブロードキャストドメインは、ジャンボフレームをサポートし、同じ MTU を使用する必要があります。伝送パス上の一貫性のない MTU 設定によるパケットの断片化と再構築により、ネットワークスループットが低下します。

接続の種類によっては、特定の MTU 制限があります。

- イーサネット: MTU は 9000 バイトに制限されます。
- データグラムモードの IP over InfiniBand (IPoIB): MTU は、InfiniBand MTU より 4 バイト小さい値に制限されます。
- インメモリーネットワークは通常、より大きな MTU をサポートします。詳細については、それぞれのドキュメントを参照してください。

34.6.4. UDP トラフィックスループットに対する CPU 速度の影響

バルク転送では、UDP プロトコルは TCP よりも効率が大幅に低くなります。これは、主に UDP でのパケット集約が欠落しているためです。デフォルトでは、Generic Receive Offload (GRO) および Transmit Segmentation Offload (TSO) 機能は有効になっていません。その結果、CPU 周波数によって、高速リンクでのバルク転送の UDP スループットが制限される可能性があります。

たとえば、高い最大伝送単位 (MTU) と大きなソケットバッファーを備えたチューニングされたホストでは、3 GHz CPU が、UDP トラフィックをフルスピードで送受信する 10 GBit NIC のトラフィックを処理できます。ただし、UDP トラフィックを送信する場合は、3 GHz 未満の CPU 速度 100 MHz ごとに約 1 - 2 Gbps の速度低下が予想されます。また、3 GHz の CPU 速度がほぼ 10 Gbps に達する場合、同じ CPU は 40 GBit NIC 上の UDP トラフィックを約 20 - 25 Gbps に制限します。

34.6.5. システム全体の UDP ソケットバッファーの増加

ソケットバッファーには、カーネルが受信したデータ、または送信する必要があるデータが一時的に保存されます。

- 読み取りソケットバッファーには、カーネルが受信したがアプリケーションがまだ読み取っていないパケットが保持されます。
- 書き込みソケットバッファーには、アプリケーションがバッファーに書き込んだものの、カーネルがまだ IP スタックとネットワークドライバーに渡していないパケットが保持されます。

UDP パケットが大きすぎてバッファーサイズを超えている場合、またはパケットの送受信速度が速すぎる場合、カーネルはデータがバッファーから削除されるまで、新しい受信 UDP パケットをドロップします。この場合、ソケットバッファーを増やすことでパケットロスを防ぐことができます。

重要

バッファーサイズを大きすぎる設定にすると、メモリーが無駄に消費されます。各ソケットはアプリケーションが要求するサイズに設定でき、カーネルはこの値を 2 倍にします。たとえば、アプリケーションが 256 KiB のソケットバッファーサイズを要求し、100 万個のソケットを開く場合、システムは潜在的なソケットバッファースペースとしてのみ 512 GB RAM (512 KiB x 100 万) を必要とします。

前提条件

- かなりの割合で UDP パケットがドロップされました。

手順

1. **/etc/sysctl.d/10-udp-socket-buffers.conf** ファイルを作成し、要件に基づいて最大読み取りバッファーサイズまたは書き込みバッファーサイズ、あるいはその両方を設定します。

```
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
```

   値をバイト単位で指定します。この例の値は、バッファーの最大サイズを 16 MiB に設定します。両方のパラメーターのデフォルト値は **212992** バイト (208 KiB) です。

2. **/etc/sysctl.d/10-udp-socket-buffers.conf** ファイルから設定をロードします。

```
# sysctl -p /etc/sysctl.d/10-udp-socket-buffers.conf
```

3. より大きなソケットバッファーサイズを使用するようにアプリケーションを設定します。**net.core.rmem_max** および **net.core.wmem_max** パラメーターは、アプリケーションの **setsockopt()** 関数が要求できる最大バッファーサイズを定義します。**setsockopt()** 関数を使用しないようにアプリケーションを設定すると、カーネルは **rmem_default** および **wmem_default** パラメーターの値を使用することに注意してください。

   詳細については、アプリケーションのプログラミング言語のドキュメントを参照してください。アプリケーションの開発者ではない場合は、開発者にお問い合わせください。

4. 新しい UDP バッファーサイズを使用するには、アプリケーションを再起動します。

検証

- パケットドロップが発生したときに使用したのと同じ方法を使用して、パケットドロップ統計を監視します。
  パケットドロップが依然として発生するが、レートが低い場合は、バッファーサイズをさらに増やします。

関連情報

- ソリューション記事 What are the implications of changing socket buffer sizes?
- **udp(7)** man ページ
- **socket(7)** man ページ

34.7. アプリケーション読み取りソケットバッファーのボトルネックの特定

TCP アプリケーションが読み取りソケットバッファーを十分な頻度でクリアしない場合、パフォーマンスが低下し、パケットが失われる可能性があります。Red Hat Enterprise Linux は、このような問題を特定するためのさまざまなユーティリティーを提供します。

34.7.1. 受信バッファーのコラプシングとプルーニングの特定

受信キュー内のデータが受信バッファーサイズを超えると、TCP スタックはソケットバッファーから不要なメタデータを削除して、スペースを解放しようとします。このステップはコラプシングとして知られています。

コラプシングが追加のトラフィック用に十分なスペースを解放できない場合、カーネルは着信する新しいデータをプルーニングします。これは、カーネルがメモリーからデータを削除し、パケットが失われることを意味します。

操作のコラプシングとプルーニングを回避するには、TCP バッファーのコラプシングとプルーニングがサーバー上で発生するかどうかを監視し、この場合は TCP バッファーをチューニングします。

手順

1. **nstat** ユーティリティーを使用して、**TcpExtTCPRcvCollapsed** カウンターと **TcpExtRcvPruned** カウンターをクエリーします。

```
# nstat -az TcpExtTCPRcvCollapsed TcpExtRcvPruned
#kernel
TcpExtRcvPruned            0        0.0
TcpExtTCPRcvCollapsed      612859   0.0
```

2. しばらく待ってから、**nstat** コマンドを再実行します。

```
# nstat -az TcpExtTCPRcvCollapsed TcpExtRcvPruned
#kernel
TcpExtRcvPruned            0        0.0
TcpExtTCPRcvCollapsed      620358   0.0
```

3. 最初の実行と比較してカウンターの値が増加している場合は、チューニングが必要です。

   - アプリケーションが **setsockopt(SO_RCVBUF)** 呼び出しを使用している場合は、それを削除することを検討してください。この呼び出しでは、アプリケーションは呼び出しで指定された受信バッファーサイズのみを使用し、サイズを自動チューニングするソケットの機能をオフにします。

   - アプリケーションが **setsockopt(SO_RCVBUF)** 呼び出しを使用しない場合は、TCP 読み取りソケットバッファーのデフォルト値と最大値をチューニングします。

4. 受信バックログキューを表示します (**Recv-Q**)。

```
# ss -nti
State   Recv-Q   Send-Q   Local Address:Port   Peer Address:Port   Process
ESTAB   0        0        192.0.2.1:443        192.0.2.125:41574
     :7,7 ... lastrcv:543 ...
ESTAB   78       0        192.0.2.1:443        192.0.2.56:42612
     :7,7 ... lastrcv:658 ...
ESTAB   88       0        192.0.2.1:443        192.0.2.97:40313
     :7,7 ... lastrcv:5764 ...
...
```

5. **ss -nt** コマンドを、各実行の間に数秒の待ち時間を設けて複数回実行します。
出力の **Recv-Q** 列に高い値が 1 件だけリストされている場合、アプリケーションは 2 つの受信操作の間にありました。ただし、**lastrcv** が継続的に増加する一方で、**Recv-Q** の値が一定のままである場合、または **Recv-Q** が時間の経過とともに継続的に増加する場合は、次の問題のいずれかが原因である可能性があります。

   - アプリケーションはソケットバッファーを十分な頻度でチェックしません。この問題の解決方法の詳細は、アプリケーションのベンダーにお問い合わせください。
   - アプリケーションは十分な CPU 時間を取得できません。この問題をさらにデバッグするには、以下を実行します。

     i. アプリケーションが実行されている CPU コアを表示します。

```
# ps -eo pid,tid,psr,pcpu,stat,wchan:20,comm
   PID     TID PSR %CPU STAT WCHAN               COMMAND
...
 44594   44594   5  0.0 Ss   do_select           httpd
 44595   44595   3  0.0 S    skb_wait_for_more_pa httpd
 44596   44596   5  0.0 Sl   pipe_read           httpd
 44597   44597   5  0.0 Sl   pipe_read           httpd
 44602   44602   5  0.0 Sl   pipe_read           httpd
...
```

     PSR 列には、プロセスが現在割り当てられている CPU コアが表示されます。

     ii. 同じコア上で実行されている他のプロセスを特定し、それらを他のコアに割り当てることを検討してください。

関連情報

- システム全体の TCP ソケットバッファーの増加

34.8. 大量のリクエストを受信するアプリケーションのチューニング

Web サーバーなど、大量の受信リクエストを処理するアプリケーションを実行する場合、パフォーマンスを最適化するために Red Hat Enterprise Linux をチューニングすることが必要になる場合があります。

34.8.1. 多数の TCP 接続試行を処理するための TCP リッスンバックログの調整

アプリケーションが **LISTEN** 状態で TCP ソケットを開くと、カーネルは、このソケットが処理できる許可されるクライアント接続の数を制限します。クライアントが、アプリケーションが処理できるよりも多くの接続を確立しようとすると、新しい接続が失われるか、カーネルが SYN Cookie をクライアントに送信します。

システムが通常のワークロード下にあり、正規のクライアントからの接続が多すぎるためにカーネルが SYN Cookie を送信する場合は、Red Hat Enterprise Linux (RHEL) をチューニングしてこれを回避します。

前提条件

- RHEL は、**possible SYN flooding on port <ip_address>:<port_number>** エラーメッセージを Systemd ジャーナルに記録します。

- 多数の接続試行は有効なソースからのものであり、攻撃によって引き起こされたものではありません。

手順

1. チューニングが必要かどうかを確認するには、影響を受けるポートの統計を表示します。

```
# ss -ntl '( sport = :443 )'
State    Recv-Q   Send-Q   Local Address:Port   Peer Address:Port  Process
LISTEN   650      500      192.0.2.1:443        0.0.0.0:*
```

   バックログ (**Recv-Q**) 内の現在の接続数がソケットバックログ (**Send-Q**) より大きい場合、リッスンバックログはまだ十分大きくないため、チューニングが必要です。

2. オプション: 現在の TCP リッスンバックログ制限を表示します。

```
# sysctl net.core.somaxconn
net.core.somaxconn = 4096
```

3. **/etc/sysctl.d/10-socket-backlog-limit.conf** ファイルを作成し、り大きなリッスンバックログ制限を設定します。

```
net.core.somaxconn = 8192
```

   アプリケーションは **net.core.somaxconn** カーネルパラメーターで指定したよりも多くのリッスンバックログを要求することができますが、カーネルはこのパラメーターで設定した番号にアプリケーションを制限することに注意してください。

4. **/etc/sysctl.d/10-socket-backlog-limit.conf** ファイルから設定をロードします。

```
# sysctl -p /etc/sysctl.d/10-socket-backlog-limit.conf
```

5. 新しいリッスンバックログ制限を使用するようにアプリケーションを再設定します。

   - アプリケーションが制限の config オプションを提供している場合は、それを更新します。たとえば、Apache HTTP サーバーには、このサービスのリッスンバックログ制限を設定するための **ListenBacklog** 設定オプションが用意されています。

   - 制限を設定できない場合は、アプリケーションを再コンパイルします。

6. アプリケーションを再起動します。

検証

1. Systemd ジャーナルを監視して、**possible SYN flooding on port <port_number>** エラーメッセージがさらに発生していないかどうかを確認します。

2. バックログ内の現在の接続数を監視し、ソケットバックログと比較します。

```
# ss -ntl '( sport = :443 )'
State    Recv-Q   Send-Q   Local Address:Port   Peer Address:Port  Process
LISTEN   0        500      192.0.2.1:443        0.0.0.0:*
```

   バックログ (**Recv-Q**) 内の現在の接続数がソケットバックログ (**Send-Q**) より大きい場合、リッスンバックログが十分に大きくないため、さらなるチューニングが必要です。

関連情報

- ソリューション記事 kernel: Possible SYN flooding on port #.Sending cookies
- ソリューション記事 Listening TCP server ignores SYN or ACK for new connection handshake
- **listen(2)** man ページ

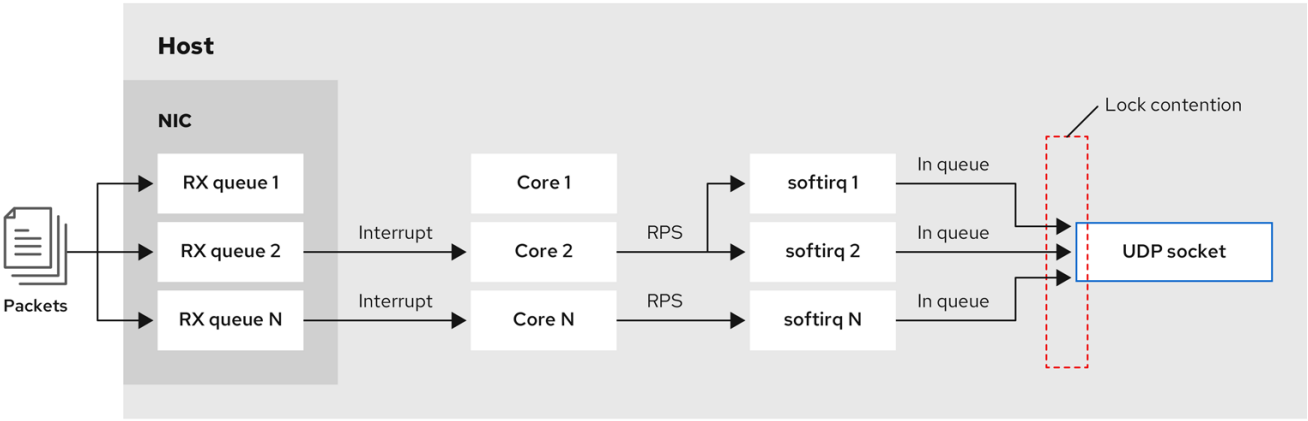
34.9. リッスンキューのロック競合の回避

キューロックの競合により、パケットドロップや CPU 使用率の上昇を引き起こす可能性があり、その結果、レイテンシーが長くなる可能性があります。アプリケーションをチューニングし、送信パケットステアリングを使用することで、受信 (RX) キューと送信 (TX) キューでのキューロックの競合を回避できます。

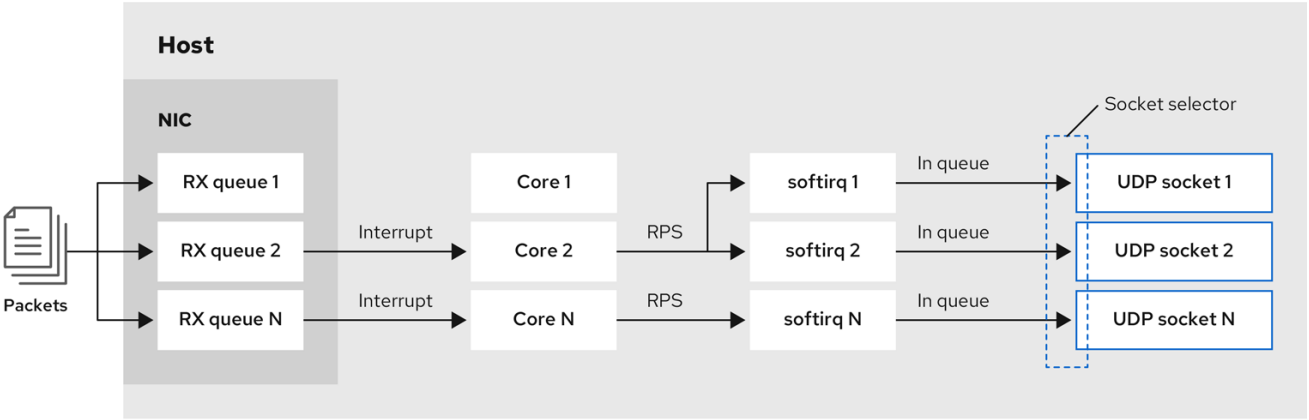
34.9.1. RX キューのロック競合の回避: SO_REUSEPORT および SO_REUSEPORT_BPF ソケットオプション

マルチコアシステムでは、アプリケーションが **SO_REUSEPORT** または **SO_REUSEPORT_BPF** ソケットオプションを使用してポートを開くと、マルチスレッドネットワークサーバーアプリケーションのパフォーマンスを向上することができます。アプリケーションがこれらのソケットオプションのいずれかを使用しない場合、すべてのスレッドは受信トラフィックを受信するために単一のソケットを共有するように強制されます。単一のソケットを使用すると、次のような問題が発生します。

- パケットドロップや CPU 使用率の上昇を引き起こす可能性のある受信バッファーでの重大な競合。
- CPU 使用率の大幅な増加
- パケットドロップの可能性



SO_REUSEPORT または **SO_REUSEPORT_BPF** ソケットオプションを使用すると、1 つのホスト上の複数のソケットを同じポートにバインドできます。



Red Hat Enterprise Linux では、カーネルソースで **SO_REUSEPORT** ソケットオプションを使用する方法のコードサンプルを提供します。コード例にアクセスするには、以下を実行します。

1. **rhel-8-for-x86_64-baseos-debug-rpms** リポジトリーを有効にします。

```
# subscription-manager repos --enable rhel-8-for-x86_64-baseos-debug-rpms
```

2. **kernel-debuginfo-common-x86_64** パッケージをインストールします。

```
# yum install kernel-debuginfo-common-x86_64
```

3. コード例は **/usr/src/debug/kernel-<version>/linux-<version>/tools/testing/selftests/net/reuseport_bpf_cpu.c** ファイルで利用できるようになりました。

関連情報

- **socket(7)** man ページ
- **/usr/src/debug/kernel-<version>/linux-<version>/tools/testing/selftests/net/reuseport_bpf_cpu.c**

34.9.2. TX キューのロック競合の回避: 送信パケットステアリング

複数のキューをサポートするネットワークインターフェイスコントローラー (NIC) を備えたホストでは、送信パケットステアリング (XPS) によって送信ネットワークパケットの処理が複数のキューに分散されます。これにより、複数の CPU が送信ネットワークトラフィックを処理できるようになり、送信キューのロック競合と、その結果として生じるパケットドロップを回避できます。

ixgbe、**i40e**、**mlx5** などの特定のドライバーは、XPS を自動的に設定します。ドライバーがこの機能をサポートしているかどうかを確認するには、NIC ドライバーのドキュメントを参照してください。ドライバーがこの機能をサポートしているかどうかを確認するには、NIC ドライバーのドキュメントを参照してください。ドライバーが XPS 自動チューニングをサポートしていない場合は、CPU コアを送信キューに手動で割り当てることができます。

注記

Red Hat Enterprise Linux には、送信キューを CPU コアに永続的に割り当てるオプションがありません。スクリプトでコマンドを使用し、システムの起動時に実行します。

前提条件

- NIC が複数のキューをサポートする。
- **numactl** パッケージがインストールされている。

手順

1. 使用可能なキューの数を表示します。

```
# ethtool -l enp1s0
Channel parameters for enp1s0:
Pre-set maximums:
RX:  0
TX:  0
Other:  0
Combined: 4
Current hardware settings:
RX:  0
TX:  0
Other:  0
Combined: 1
```

Pre-set maximums セクションにはキューの総数が表示され、**Current hardware settings** には受信キュー、送信キュー、その他のキュー、または結合されたキューに現在割り当てられているキューの数が表示されます。

2. オプション: 特定のチャネルにキューが必要な場合は、それに応じてキューを割り当てます。たとえば、4 つのキューを **Combined** チャネルに割り当てるには、次のように入力します。

```
# ethtool -L enp1s0 combined 4
```

3. NIC がどの Non-Uniform Memory Access (NUMA) ノードに割り当てられているかを表示します。

```
# cat /sys/class/net/enp1s0/device/numa_node
0
```

ファイルが見つからない場合、またはコマンドが **-1** を返す場合は、ホストは NUMA システムではありません。

4. ホストが NUMA システムの場合は、どの CPU がどの NUMA ノードに割り当てられているかを表示します。

```
# lscpu | grep NUMA
NUMA node(s):      2
NUMA node0 CPU(s):  0-3
NUMA node1 CPU(s):  4-7
```

5. 上の例では、NIC には 4 つのキューがあり、NIC は NUMA ノード 0 に割り当てられています。このノードは CPU コア 0 - 3 を使用します。したがって、各送信キューを 0 - 3 の CPU コアの 1 つにマッピングします。

```
# echo 1 > /sys/class/net/enp1s0/queues/tx-0/xps_cpus
# echo 2 > /sys/class/net/enp1s0/queues/tx-1/xps_cpus
```

```
# echo 4 > /sys/class/net/enp1s0/queues/tx-2/xps_cpus
# echo 8 > /sys/class/net/enp1s0/queues/tx-3/xps_cpus
```

CPU コアと送信 (TX) キューの数が同じ場合は、1対1マッピングを使用して、TX キューでのあらゆる種類の競合を回避します。そうしないと、複数の CPU を同じ TX キューにマップすると、異なる CPU での送信操作によって TX キューのロック競合が発生し、送信スループットに悪影響を及ぼします。

CPU のコア番号を含むビットマップをキューに渡す必要があることに注意してください。次のコマンドを使用してビットマップを計算します。

```
# printf %x $((1 << <core_number> ))
```

検証

1. トラフィックを送信するサービスのプロセス ID (PID) を特定します。

```
# pidof <process_name>
12345 98765
```

2. XPS を使用するコアに PID を固定します。

```
# numactl -C 0-3 12345 98765
```

3. プロセスがトラフィックを送信している間、**requeues** カウンターを監視します。

```
# tc -s qdisc
qdisc fq_codel 0: dev enp10s0u1 root refcnt 2 limit 10240p flows 1024 quantum 1514
target 5ms interval 100ms memory_limit 32Mb ecn drop_batch 64
 Sent 125728849 bytes 1067587 pkt (dropped 0, overlimits 0 requeues 30)
 backlog 0b 0p requeues 30
 ...
```

requeues カウンターが大幅な速度で増加しなくなると、TX キューロックの競合は発生しなくなります。

関連情報

- **/usr/share/doc/kernel-doc-_<version>/Documentation/networking/scaling.rst**

34.9.3. UDP トラフィックが多いサーバーでの汎用受信オフロード機能の無効化

高速 UDP バルク転送を使用するアプリケーションは、UDP ソケットで UDP Generic Receive Offload (GRO) を有効にして使用する必要があります。ただし、次の条件が当てはまる場合は、GRO を無効にしてスループットを向上させることができます。

- アプリケーションは GRO をサポートしていないため、機能を追加できません。
- TCP スループットは関係ありません。

警告

GRO を無効にすると、TCP トラフィックの受信スループットが大幅に低下します。したがって、TCP パフォーマンスが関係するホストでは GRO を無効にしないでください。

前提条件

- ホストは主に UDP トラフィックを処理している。
- アプリケーションは GRO を使用していない。
- ホストは、VXLAN などの UDP トンネルプロトコルを使用していない。
- ホストは仮想マシン (VM) やコンテナーを実行していない。

手順

1. オプション: NetworkManager 接続プロファイルを表示します。

```
# nmcli connection show
NAME     UUID                                  TYPE      DEVICE
example  f2f33f29-bb5c-3a07-9069-be72eaec3ecf  ethernet  enp1s0
```

2. 接続プロファイルで GRO サポートを無効にします。

```
# nmcli connection modify example ethtool.feature-gro off
```

3. 接続プロファイルを再度アクティベートします。

```
# nmcli connection up example
```

検証

1. GRO が無効になっていることを確認します。

```
# ethtool -k enp1s0 | grep generic-receive-offload
generic-receive-offload: off
```

2. サーバー上のスループットを監視します。この設定がホスト上の他のアプリケーションにマイナスの影響を与える場合は、NetworkManager プロファイルで GRO を再度有効にします。

関連情報

- Improve UDP performance in RHEL 8.5

34.10. デバイスドライバーと NIC のチューニング

RHEL では、カーネルモジュールは、ネットワークインターフェイスコントローラー (NIC) 用のドライ

バーを提供します。これらのモジュールは、デバイスドライバーと NIC をチューニングおよび最適化するためのパラメーターをサポートしています。たとえば、ドライバーが受信割り込みの生成の遅延をサポートしている場合は、対応するパラメーターの値を減らして、受信記述子の不足を避けることができます。

注記

すべてのモジュールがカスタムパラメーターをサポートしているわけではなく、機能はハードウェア、ドライバーおよびファームウェアのバージョンによって異なります。

34.10.1. カスタム NIC ドライバーのパラメーターの設定

多くのカーネルモジュールは、ドライバーとネットワークインターフェイスコントローラー (NIC) をチューニングするためのパラメーターの設定をサポートしています。ハードウェアやドライバーに応じて設定をカスタマイズできます。

重要

カーネルモジュールにパラメーターを設定すると、RHEL はこれらの設定をこのドライバーを使用するすべてのデバイスに適用します。

前提条件

- ホストに NIC がインストールされている。
- NIC のドライバーを提供するカーネルモジュールは、必要なチューニング機能をサポートしている。
- ローカルでログインしているか、パラメーターを変更するドライバーを使用するネットワークインターフェイスとは異なるネットワークインターフェイスを使用してログインしている。

手順

1. ドライバーを特定します。

```
# ethtool -i enp0s31f6
driver: e1000e
version: ...
firmware-version: ...
...
```

特定の機能には、特定のドライバーとファームウェアのバージョンが必要になる場合があることに注意してください。

2. カーネルモジュールの利用可能なパラメーターを表示します。

```
# modinfo -p e1000e
...
SmartPowerDownEnable:Enable PHY smart power down (array of int)
parm:RxIntDelay:Receive Interrupt Delay (array of int)
```

パラメーターの詳細については、カーネルモジュールのドキュメントを参照してください。RHEL のモジュールについては、**kernel-doc** パッケージで提供される **/usr/share/doc/kernel-doc-<version>/Documentation/networking/device_drivers/** ディレクトリーにあるドキュメ

ントを参照してください。

3. **/etc/modprobe.d/nic-parameters.conf** ファイルを作成し、モジュールのパラメーターを指定します。

```
options <module_name> <parameter1>=<value> <parameter2>=<value>
```

たとえば、ポートの省電力メカニズムを有効にし、受信割り込みの生成を 4 ユニットに設定するには、次のように入力します。

```
options e1000e SmartPowerDownEnable=1 RxIntDelay=4
```

4. モジュールをアンロードします。

```
# modprobe -r e1000e
```

警告

アクティブなネットワークインターフェイスが使用するモジュールをアンロードすると、接続が即座に終了し、サーバーからロックアウトされる可能性があります。

5. モジュールをロードします。

```
# modprobe e1000e
```

6. ネットワーク接続を再アクティブ化します。

```
# nmcli connection up <profile_name>
```

検証

1. カーネルメッセージを表示します。

```
# dmesg
...
[35309.225765] e1000e 0000:00:1f.6: Transmit Interrupt Delay set to 16
[35309.225769] e1000e 0000:00:1f.6: PHY Smart Power Down Enabled
...
```

すべてのモジュールがパラメーター設定をカーネルリングバッファーに記録するわけではないことに注意してください。

2. 特定のカーネルモジュールは、モジュールパラメーターごとに **/sys/module/<driver>/parameters/** ディレクトリーにファイルを作成します。これらの各ファイルには、このパラメーターの現在の値が含まれています。これらのファイルを表示して設定を確認できます。

```
# cat /sys/module/<driver_name>/parameters/<parameter_name>
```

34.11. ネットワークアダプターのオフロード設定

CPU 負荷を軽減するために、特定のネットワークアダプターは、ネットワーク処理負荷をネットワークインターフェイスコントローラー (NIC) に移動するオフロード機能を使用します。たとえば、Encapsulating Security Payload (ESP) オフロードを使用すると、NIC は ESP 操作を実行して、IPsec 接続を高速化し、CPU 負荷を軽減します。

デフォルトでは、Red Hat Enterprise Linux のほとんどのオフロード機能が有効になっています。次の場合にのみ無効にしてください。

- トラブルシューティングの目的でオフロード機能を一時的に無効にします。
- 特定の機能がホストに悪影響を与える場合は、オフロード機能を永続的に無効にします。

ネットワークドライバーでパフォーマンス関連のオフロード機能がデフォルトで有効になっていない場合は、手動で有効にすることができます。

34.11.1. オフロード機能の一時的な設定

オフロード機能によって問題が発生したり、ホストのパフォーマンスが低下したりすると予想される場合は、現在の状態に応じて、オフロード機能を一時的に有効または無効にして、原因の絞り込みを試みることができます。

オフロード機能を一時的に有効または無効にすると、次回の再起動時に以前の値に戻ります。

前提条件

- ネットワークカードがオフロード機能をサポートしている。

手順

1. インターフェイスで利用可能なオフロード機能とその現在の状態を表示します。

```
# ethtool -k enp1s0
...
esp-hw-offload: on
ntuple-filters: off
rx-vlan-filter: off [fixed]
...
```

   出力はハードウェアとそのドライバーの機能によって異なります。**[fixed]** のフラグが付いている機能の状態は変更できないことに注意してください。

2. オフロード機能を一時的に無効にします。

```
# ethtool -K <interface> <feature> [on|off]
```

   - たとえば、**enp10s0u1** インターフェイスで IPsec Encapsulating Security Payload (ESP) オフロードを一時的に無効にするには、次のように入力します。

```
# ethtool -K enp10s0u1 esp-hw-offload off
```

- たとえば、**enp10s0u1** インターフェイスで Receive Flow Steering (aRFS) フィルタリングを一時的に有効にするには、次のように入力します。

```
# ethtool -K enp10s0u1 ntuple-filters on
```

検証

1. オフロード機能の状態を表示します。

```
# ethtool -k enp1s0
...
esp-hw-offload: off
ntuple-filters: on
...
```

2. オフロード機能を変更する前に発生した問題がまだ存在するかどうかをテストします。
   - 特定のオフロード機能を変更した後に問題が解消された場合は、次のようにします。
     i. Red Hat サポート に連絡して問題を報告してください。
     ii. 修正が利用可能になるまで、オフロード機能を永続的に設定すること を検討してください。
   - 特定のオフロード機能を無効にしても問題が解決しない場合は、以下を実行します。
     i. **ethtool -K <interface> <feature> [on|off]** コマンドを使用して、設定を前の状態にリセットします。
     ii. 異なるオフロード機能を有効または無効にして、問題を絞り込みます。

関連情報

- **ethtool(8)** man ページ

34.11.2. オフロード機能の永続的な設定

ホストのパフォーマンスを制限する特定のオフロード機能を特定した場合は、現在の状態に応じて、それを永続的に有効または無効にすることができます。

オフロード機能を永続的に有効または無効にすると、NetworkManager は再起動後もその機能がこの状態のままであることを確認します。

前提条件

- ホスト上のパフォーマンスを制限する特定のオフロード機能を特定している。

手順

1. オフロード機能の状態を変更するネットワークインターフェイスを使用する接続プロファイルを特定します。

```
# nmcli connection show
NAME     UUID                                  TYPE      DEVICE
Example  a5eb6490-cc20-3668-81f8-0314a27f3f75  ethernet  enp1ss0
```

```
...
```

2. オフロード機能の状態を永続的に変更します。

```
# nmcli connection modify <connection_name> <feature> [on|off]
```

- たとえば、**Example** 接続プロファイルで IPsec Encapsulating Security Payload (ESP) オフロードを永続的に無効にするには、次のように入力します。

```
# nmcli connection modify Example ethtool.feature-esp-hw-offload off
```

- たとえば、**Example** 接続プロファイルでアクセラレート Receive Flow Steering (aRFS) フィルタリングを永続的に有効にするには、次のように入力します。

```
# nmcli connection modify Example ethtool.feature-ntuple on
```

3. 接続プロファイルを再度アクティベートします。

```
# nmcli connection up Example
```

検証

- オフロード機能の出力状態を表示します。

```
# ethtool -k enp1s0
...
esp-hw-offload: off
ntuple-filters: on
...
```

関連情報

- **nm-settings-nmcli (5)** man ページ

34.12. 割り込み結合設定のチューニング

割り込み結合は、ネットワークカードによって生成される割り込みの数を減らすためのメカニズムです。一般に、割り込みが少なくなると、ネットワークのレイテンシーと全体的なパフォーマンスが向上します。

割り込み結合設定のチューニングには、以下を制御するパラメーターの調整が含まれます。

- 1 つの割り込みに結合されるパケットの数。
- 割り込みを生成するまでの遅延。

重要

最適な結合設定は、特定のネットワーク条件と使用しているハードウェアによって異なります。したがって、環境とニーズに最適な設定を見つけるには、何度か試すことが必要な場合があります。

34.12.1. 遅延またはスループットの扱いに注意が必要なサービス向けに RHEL を最適化する

結合チューニングの目標は、特定のワークロードに必要な割り込みの数を最小限に抑えることです。高スループットの状況では、高いデータレートを維持しながら、割り込みをできるだけ少なくすることが目標となります。待ち時間が短い状況では、より多くの割り込みを使用してトラフィックを迅速に処理できます。

ネットワークカードの設定を調整して、1 つの割り込みに結合されるパケットの数を増減できます。その結果、トラフィックのスループットまたは遅延を向上させることができます。

手順

1. ボトルネックが発生しているネットワークインターフェイスを特定します。

```
# ethtool -S enp1s0
NIC statistics:
     rx_packets: 1234
     tx_packets: 5678
     rx_bytes: 12345678
     tx_bytes: 87654321
     rx_errors: 0
     tx_errors: 0
     rx_missed: 0
     tx_dropped: 0
     coalesced_pkts: 0
     coalesced_events: 0
     coalesced_aborts: 0
```

   名前に "drop"、"discard"、または "error" を含むパケットカウンターを特定します。これらの特定の統計は、ネットワークインターフェイスカード (NIC) の結合によって発生する可能性がある、NIC のパケットバッファーでの実際のパケットロスを測定します。

2. 前の手順で特定したパケットカウンターの値を監視します。
   これらをネットワークの予想値と比較して、特定のインターフェイスにボトルネックが発生しているかどうかを判断します。ネットワークのボトルネックの一般的な兆候には次のようなものがありますが、これらに限定されません。

   - ネットワークインターフェイス上での多数のエラー
   - 高いパケットロス
   - ネットワークインターフェイスの多用

   注記

   ネットワークのボトルネックを特定する際のその他の重要な要素としては、CPU 使用率、メモリー使用率、ディスク I/O などがあります。

3. 現在の結合設定を表示します。

```
# ethtool enp1s0
Settings for enp1s0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
```

```
                    100baseT/Half 100baseT/Full
                    1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                    100baseT/Half 100baseT/Full
                    1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: Unknown
        Supports Wake-on: g
        Wake-on: g
        Current message level: 0x00000033 (51)
                       drv probe link
        Link detected: yes
```

この出力では、**Speed** フィールドおよび **Duplex** フィールドを監視します。これらのフィールドには、ネットワークインターフェイスの操作に関する情報と、それが期待値で実行されているかどうかが表示されます。

4. 現在の割り込み結合設定を確認します。

```
# ethtool -c enp1s0
Coalesce parameters for enp1s0:
        Adaptive RX: off
        Adaptive TX: off
        RX usecs: 100
        RX frames: 8
        RX usecs irq: 100
        RX frames irq: 8
        TX usecs: 100
        TX frames: 8
        TX usecs irq: 100
        TX frames irq: 8
```

- **usecs** 値は、受信機または送信機が割り込みを生成する前に待機するマイクロ秒数を指します。
- **frames** 値は、受信機または送信機が割り込みを生成する前に待機するフレーム数を指します。
- **irq** 値は、ネットワークインターフェイスがすでに割り込みを処理している場合に、割り込み調整を設定するために使用されます。

注記

すべてのネットワークインターフェイスカードが、出力例のすべての値のレポートと変更をサポートしているわけではありません。

- **Adaptive RX/TX** 値は、割り込み結合設定を動的に調整する適応割り込み結合メカニズムを表します。**Adaptive RX/TX** が有効な場合、NIC ドライバーはパケット条件に基づいて、結合値を自動計算します (アルゴリズムは NIC ドライバーごとに異なります)。

5. 必要に応じて結合設定を変更します。以下に例を示します。

   - **ethtool.coalesce-adaptive-rx** が無効になっている間に、RX パケットの割り込みを生成するまでの遅延を 100 マイクロ秒に設定するように **ethtool.coalesce-rx-usecs** を設定します。

     ```
     # nmcli connection modify enp1s0 ethtool.coalesce-rx-usecs 100
     ```

   - **ethtool.coalesce-rx-usecs** がデフォルト値に設定されている間、**ethtool.coalesce-adaptive-rx** を有効にします。

     ```
     # nmcli connection modify enp1s0 ethtool.coalesce-adaptive-rx on
     ```

     Adaptive-RX 設定を次のように変更します。

     - 低レイテンシー (50us 未満) が気になるユーザーは、**Adaptive-RX** を有効にしないでください。
     - スループットを懸念するユーザーは、おそらく問題なく **Adaptive-RX** を有効にすることができます。適応割り込み結合メカニズムを使用したくない場合は、**ethtool.coalesce-rx-usecs** に 100us や 250us などの大きな値を設定してみることができます。
     - 自分のニーズがわからないユーザーは、問題が発生するまでこの設定を変更しないでください。

6. 接続を再度有効にします。

   ```
   # nmcli connection up enp1s0
   ```

検証手順

- ネットワークパフォーマンスを監視し、ドロップされたパケットを確認します。

  ```
  # ethtool -S enp1s0
  NIC statistics:
      rx_packets: 1234
      tx_packets: 5678
      rx_bytes: 12345678
      tx_bytes: 87654321
      rx_errors: 0
      tx_errors: 0
      rx_missed: 0
      tx_dropped: 0
      coalesced_pkts: 12
      coalesced_events: 34
      coalesced_aborts: 56
  ...
  ```

  rx_errors、**rx_dropped**、**tx_errors**、および **tx_dropped** フィールドの値は 0 またはそれに近い値 (ネットワークトラフィックとシステムリソースに応じて最大数百まで) である必要があり

ます。これらのフィールドの値が高い場合は、ネットワークに問題があることを示します。カウンターには異なる名前を付けることができます。名前に "drop"、"discard"、または "error" を含むパケットカウンターを注意深く監視します。

rx_packets、**tx_packets**、**rx_bytes**、および **tx_bytes** の値は時間の経過とともに増加します。値が増加しない場合は、ネットワークに問題がある可能性があります。パケットカウンターは、NIC ドライバーに応じて異なる名前を持つことができます。

重要

ethtool コマンドの出力は、使用している NIC とドライバーによって異なる場合があります。

極めて低いレイテンシーを重視するユーザーは、監視目的でアプリケーションレベルのメトリクスまたはカーネルパケットタイムスタンプ API を使用できます。

関連情報

- Initial investigation for any performance issue
- What are the kernel parameters available for network tuning?
- How to make NIC ethtool settings persistent (apply automatically at boot)
- Timestamping

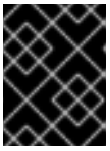
34.13. TCP タイムスタンプの利点

TCP タイムスタンプは、TCP ヘッダー内のオプションの情報であり、TCP プロトコルの拡張機能です。Red Hat Enterprise Linux ではデフォルトで TCP タイムスタンプが有効になっており、カーネルは TCP タイムスタンプを使用して、TCP 接続のラウンドトリップ時間 (RTT) をより適切に推定します。これにより、TCP ウィンドウとバッファーの計算がより正確になります。

さらに、TCP タイムスタンプは、セグメントの寿命と順序を判断し、ラップされたシーケンス番号から保護するための代替方法を提供します。TCP パケットヘッダーは、32 ビットフィールドにシーケンス番号を記録します。10 Gbps 接続では、このフィールドの値は 1.7 秒後にラップされる可能性があります。TCP タイムスタンプがないと、受信側はラップされたシーケンス番号を持つセグメントが新しいセグメントか古い重複かを判断できません。ただし、TCP タイムスタンプを使用すると、受信側はセグメントを受信するか破棄するかを正しく選択できます。したがって、高速ネットワークインターフェイスを備えたシステムでは TCP タイムスタンプを有効にすることが不可欠です。

net.ipv4.tcp_timestamps カーネルパラメーターには、次のいずれかの値を指定できます。

- **0**: TCP タイムスタンプは無効化されています。
- **1**: TCP タイムスタンプは有効化されています (デフォルト)。
- **2**: TCP タイムスタンプは有効ですが、ランダムオフセットはありません。



重要

各接続のランダムなオフセットがなければ、ホストの大体の稼働時間とフィンガープリントを決定し、この情報を攻撃に使用することが可能です。

デフォルトでは、Red Hat Enterprise Linux では TCP タイムスタンプが有効になっており、現在の時刻のみを保存するのではなく、接続ごとにランダムなオフセットを使用します。

```
# sysctl net.ipv4.tcp_timestamps
net.ipv4.tcp_timestamps = 1
```

net.ipv4.tcp_timestamps パラメーターの値がデフォルト (**1**) と異なる場合は、設定したときと同じ方法で設定を元に戻します。

関連情報

- RFC 1323: TCP Extensions for High Performance

34.14. イーサネットネットワークのフロー制御

イーサネットリンクで、ネットワークインターフェイスとスイッチポートの間で継続的にデータ送信が行われると、バッファー容量がいっぱいになる可能性があります。バッファー容量がいっぱいになると、ネットワークの輻輳が発生します。この場合、送信側が受信側の処理能力よりも高いレートでデータを送信すると、パケットロスが発生する可能性があります。リンクの反対側のネットワークインターフェイス (スイッチポート) のデータ処理能力が低いためです。

フロー制御メカニズムは、送信側と受信側の送受信能力がそれぞれ異なるイーサネットリンクを介したデータ送信を管理します。パケットロスを回避するために、イーサネットフロー制御メカニズムはパケット送信を一時的に停止し、スイッチポート側の高い伝送レートを制御します。なお、ルーターがスイッチポートを越えてポーズフレームを転送することはありません。

受信 (RX) バッファーがいっぱいになると、受信側は送信側にポーズフレームを送信します。その後、送信側は、1秒未満の短い期間、データ送信を停止しますが、この一時停止期間中は受信データのバッファリングを続けます。この期間は、受信側がインターフェイスバッファーを空にして、バッファーオーバーフローを防ぐのに十分な時間を提供します。

注記

イーサネットリンクのどちら側も、ポーズフレームを反対側に送信できます。ネットワークインターフェイスの受信バッファーがいっぱいになると、ネットワークインターフェイスはポーズフレームをスイッチポートに送信します。同様に、スイッチポートの受信バッファーがいっぱいになると、スイッチポートはネットワークインターフェイスにポーズフレームを送信します。

デフォルトでは、Red Hat Enterprise Linux のほとんどのネットワークドライバーではポーズフレームのサポートが有効になっています。ネットワークインターフェイスの現在の設定を表示するには、次のように入力します。

```
# ethtool --show-pause enp1s0
Pause parameters for enp1s0:
...
RX:     on
TX:     on
...
```

スイッチのベンダーに問い合わせて、スイッチがポーズフレームをサポートしているかどうかを確認してください。

関連情報

- **ethtool(8)** man ページ
- ネットワークリンクフロー制御とは何ですか? Red Hat Enterprise Linux ではどのように機能しますか?

第35章 メモリーアクセスを最適化するためにオペレーティングシステムの設定

RHEL に含まれているツールを使用して、オペレーティングシステムを設定し、ワークロード全体でメモリーアクセスを最適化できます。

35.1. システムメモリーの問題を監視および診断するツール

以下のツールは、システムパフォーマンスを監視し、システムメモリーに関連するパフォーマンス問題を診断するために、Red Hat Enterprise Linux 8 で利用できます。

- **procps-ng** パッケージが提供する **vmstat** ツールは、システムのプロセス、メモリー、ページング、ブロック I/O、トラップ、ディスク、および CPU アクティビティーのレポートを表示します。これは、マシンが最後にオンされてから、または前回のレポート以降、これらのイベントの平均をインスタンス化するレポートを提供します。

- **valgrind** フレームワークは、ユーザー空間のバイナリーにインストルメンテーションを提供します。**yum install valgrind** コマンドを使用して、このツールをインストールします。これには、以下のようなプログラムパフォーマンスのプロファイリングおよび分析に使用できるツールが多数含まれています。

  - **memcheck** オプションは、デフォルトの **valgrind** ツールです。これは、以下のような多くのメモリーエラーを検出し、報告することが困難となる可能性のあるメモリーエラーについて検出および報告します。

    - 発生すべきでないメモリーアクセス
    - 未定義または初期化されていない値の使用
    - 誤って解放されたヒープメモリー
    - ポインターの重複
    - メモリーリーク

    注記

    memcheck は、このエラーのみを報告でき、エラーを回避することはできません。ただし、**memcheck** は、エラーが発生した場合すぐにエラーメッセージを記録します。

  - **cachegrind** オプションは、システムのキャッシュ階層および分岐予測とのアプリケーションの対話をシミュレートします。アプリケーションの実行期間についての統計を収集し、コンソールにサマリーを出力します。

  - **massif** オプションは、指定されたアプリケーションによって使用されるヒープ容量を測定します。便利な容量やブックキーピングと調整目的で割り当てられた容量を測定します。

関連情報

- man ページの **vmstat(8)** および **valgrind(1)**
- **/usr/share/doc/valgrind-version/valgrind_manual.pdf** ファイル

35.2. システムのメモリーの概要

Linux カーネルは、システムのメモリーリソース (RAM) の使用状況を最大化するために設計されています。このような設計の特徴と、ワークロードのメモリー要件によっては、システムのメモリーの一部がワークロードの変わりにカーネル内で使用されますが、メモリーのサイズは解放されています。この空きメモリーは、特別なシステム割り当てや、その他の優先度のシステムサービス用に予約されています。

システムメモリーの残りの部分はワークロード自体に専用で、以下の 2 つのカテゴリーに分類されます。

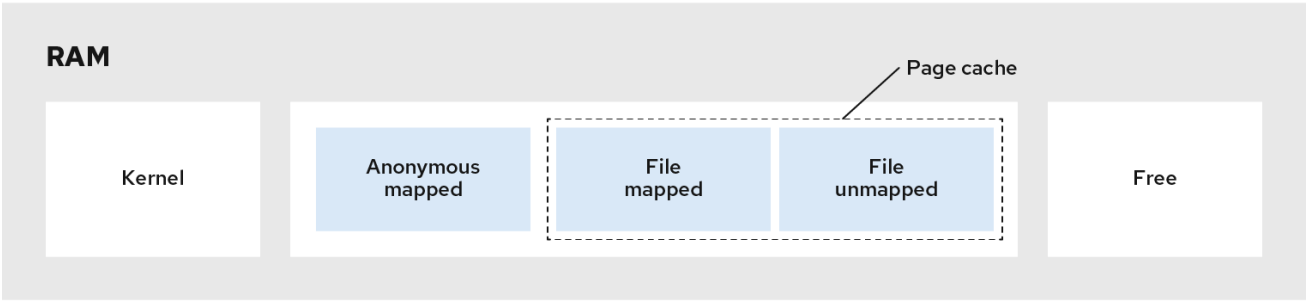
File memory

このカテゴリーに追加されたページは、永続ストレージのファイルの一部を表します。ページキャッシュのこれらのページは、アプリケーションのアドレス空間でマッピングまたはマッピング解除できます。アプリケーションを使用することで、**mmap** システムコールを使用してファイルをアドレス空間にマップしたり、バッファー I/O の読み取りおよび書き込み経由でファイルで操作したりできます。
バッファーされた I/O システムコール、およびページを直接マップするアプリケーションも、マッピングされていないページを再使用できます。その結果、これらのページは、同じページのセットに負荷の高い I/O 操作を再発行しないように、カーネルによりキャッシュに保存されます。これは特に、システムがメモリーインテンシブなタスクを実行していないときが該当します。

Anonymous memory

このカテゴリーのページは、動的に割り当てられたプロセスで使用されているか、永続ストレージのファイルに関連しません。この一連のページは、アプリケーションスタックやヒープ領域など、各タスクのメモリー内制御構造をバックアップします。

図35.1 メモリー使用状況パターン



133_RHEL_0121

35.3. 仮想メモリーパラメーター

仮想メモリーのパラメーターは、**/proc/sys/vm** ディレクトリーにリスト表示されます。

利用可能な仮想メモリーパラメーターを以下に示します。

vm.dirty_ratio

パーセンテージの値です。システムメモリー合計の割合がこの値を超えると、システムは、**pdflush** 操作でディスクへの変更の書き込みを開始します。デフォルト値は **20** % です。

vm.dirty_background_ratio

パーセンテージの値。システムメモリー合計の割合がこの値を超えると、システムはバックグラウンドでディスクへの変更の書き込みを開始します。デフォルト値は **10** % です。

vm.overcommit_memory

大容量メモリーのリクエストを受け入れるか拒否するかを決定する条件を定義します。デフォルト値は **0** です。
デフォルトでは、カーネルは仮想メモリー割り当て要求が現在のメモリー量 (合計 + スワップ) に収まるかどうかをチェックし、大きな要求のみを拒否します。それ以外の場合、仮想メモリーの割り当ては付与され、これはメモリーのオーバーコミットが許可されることを意味します。

overcommit_memory パラメーターの値を設定します。

- このパラメーターを **1** に設定するとカーネルはメモリーのオーバーコミット処理を行いません。これにより、メモリーがオーバーロードする可能性が向上しますが、メモリー集中型タスクのパフォーマンスが向上します。

- このパラメーターを **2** に設定すると、カーネルは、利用可能なスワップ領域の合計と、**overcommit_ratio** で指定される物理 RAM の割合またはそれ以上のメモリーの要求を拒否します。これにより、メモリーのオーバーコミットのリスクが軽減されますが、物理メモリーよりも大きいスワップ領域があるシステムのみに推奨されます。

vm.overcommit_ratio

overcommit_memory が **2** に設定されている場合に考慮される物理 RAM の割合を指定します。デフォルト値は **50** です。

vm.max_map_count

プロセスが使用可能なメモリーマップ領域の最大数を定義します。デフォルト値は **65530** です。アプリケーションに十分なメモリーマップ領域が必要な場合は、この値を増やします。

vm.min_free_kbytes

予約済み空きページプールのサイズを設定します。また、Linux カーネルのページ回収アルゴリズムの動作を管理する **min_page**、**low_page**、**high_page** のしきい値も設定します。また、システム全体で空き状態になる最小キロバイト数も指定します。これにより、各ローメモリーゾーンの特定の値を計算します。それぞれには、サイズに比例して予約済み空きページが多数割り当てられます。
vm.min_free_kbytes パラメーターの値の設定:

- パラメーターの値を増やすと、アプリケーションのワーキングセットが効果的に減少します。そのため、カーネル駆動型のワークロードのみに使用するほうがよい場合があります。この場合、ドライバーバッファーはアトミックコンテキストで割り当てる必要があります。

- パラメーターの値を下げると、システムでメモリーが不足した場合に、カーネルがシステム要求の処理をレンダリングできなくなる可能性があります。

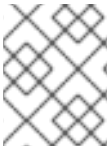
警告

極端な値は、システムのパフォーマンスに悪影響を与えます。**vm.min_free_kbytes** が非常に低い値に設定すると、システムのメモリーを効果的に回収できなくなります。これにより、システムがクラッシュし、サービス割り込みやその他のカーネルサービスに失敗する可能性があります。ただし、**vm.min_free_kbytes** を設定すると、システムの回収アクティビティーが大幅に増大し、誤ったダイレクト回収状態により割り当てレイテンシーが発生します。これにより、システムがメモリー不足の状態に即座に入ります。

vm.min_free_kbytes パラメーターは、**min_pages** というページ回収ウォーターマークも設定します。このウォーターマークは、ページの回収アルゴリズムを管理する他の 2 つのメモリー基準 (**low_pages**、および **high_pages**) を決定する要素として使用されます。

/proc/PID/oom_adj

システムがメモリー不足になり、**panic_on_oom** パラメーターが **0** に設定されている場合は、**oom_killer** 関数は、システムが復旧するまで、プロセスを強制終了し、最も高い **oom_score** を持つプロセスを開始します。
oom_adj パラメーターは、プロセスの **oom_score** を決定します。このパラメーターはプロセス ID ごとに設定されます。**-17** の値は、そのプロセスの **oom_killer** を無効にします。そのほかの有効な値は、**-16** から **15** までになります。



注記

調整したプロセスによって作成されたプロセスは、そのプロセスの **oom_score** を継承します。

vm.swappiness

swappiness 値 (**0** から **200** まで) は、システムが匿名メモリープールまたはページキャッシュメモリープールからメモリーの回収を優先するレベルを制御します。
swappiness パラメーターの値の設定:

- 値を高くすると、ファイルマップ駆動型のワークロードが優先され、RAM のアクティブにアクセスされるプロセスの匿名マッピングメモリーをスワップアウトします。これは、ストレージのファイルのデータに依存するファイルサーバーやストリーミングアプリケーションが、サービスリクエストの I/O レイテンシーを低減させるためにメモリーに駐在するのに便利です。

- 値が小さいほど、ページキャッシュ (ファイルマップされたメモリー) を回収しつつ、匿名のマッピング駆動型ワークロードが優先されます。この設定は、ファイルシステム情報に大きく依存しないアプリケーション、数学的アプリケーションや数値計算アプリケーションなどの動的に割り当てられたメモリーやプライベートメモリーを大幅に使用するアプリケーション、QEMU のような一部のハードウェア仮想化スーパーバイザーにおいて有用です。
vm.swappiness パラメーターのデフォルト値は **60** です。

警告

- **vm.swappiness** を **0** に設定すると、匿名メモリーをディスクにスワップアウトする必要がなくなります。これにより、メモリーまたは I/O 集約型のワークロード下で **oom_killer** 関数によるプロセスの強制終了のリスクが高まります。
- **cgroupsV1** を使用している場合は、**cgroupsV1** 専用の cgroup ごとの swappiness 値が使用されます。そのため、**vm.swappiness** パラメーターでシステム全体のスワップを設定しても、システムのスワップ動作にはほとんど影響がありません。この問題により、予期しない一貫性のないスワップ動作が発生する可能性があります。このような場合は、**vm.force_cgroup_v2_swappiness** パラメーターの使用を検討してください。

  詳細は、Premature swapping with swappiness=0 while there is still plenty of pagecache to be reclaimed KCS ソリューションを参照してください。

force_cgroup_v2_swappiness

この制御は、**cgroupsV1** でのみ使用可能な cgroup ごとの swappiness 値を非推奨にするために使用されます。すべてのシステムおよびユーザープロセスのほとんどは、cgroup 内で実行されます。Cgroup swappiness 値のデフォルトは 60 です。これにより、システムの swappiness 値がシステムのスワップ動作にほとんど影響を与えないという影響が生じる可能性があります。ユーザーが cgroup ごとの swappiness 機能を気にしない場合は、**force_cgroup_v2_swappiness=1** でシステムを設定して、システム全体でより一貫した swappiness の動作を持たせることができます。

関連情報

- man ページの **sysctl(8)**
- メモリー関連のカーネルパラメーターの設定

35.4. ファイルシステムパラメーター

ファイルシステムパラメーターは、**/proc/sys/fs** ディレクトリーにリスト表示されます。利用可能なファイルシステムパラメーターを以下に示します。

aio-max-nr

すべてのアクティブな非同期入出力コンテキストで許可されるイベントの最大数を定義します。デフォルト値は **65536** で、この値を変更しても、カーネルデータ構造の割り当てまたはサイズ変更は行われません。

file-max

システム全体でのファイルハンドルの最大数を決定します。Red Hat Enterprise Linux 8 のデフォルト値は、**8192** またはカーネル起動時に利用可能な空きメモリーページの 10 分の 1 のいずれか高い方になります。
この値を設定すると、利用可能なファイルハンドルがないためにエラーを解決できます。

関連情報

- man ページの **sysctl(8)**

35.5. カーネルパラメーター

カーネルパラメーターのデフォルト値は **/proc/sys/kernel/** ディレクトリーにあります。これらは、カーネルによって提供される設定されたデフォルト値、または **sysctl** を介してユーザーによって指定された値です。

以下は、**msg*** および **shm*** System V IPC (**sysvipc**) システムコールの制限を設定するのに使用されるカーネルパラメーターです。

msgmax

メッセージキューの単一のメッセージに対する最大許容サイズ (バイト単位) を定義します。この値は、キューのサイズ (**msgmnb**) を超えることはできません。**sysctl msgmax** コマンドを使用して、システムの現在の **msgmax** 値を確認します。

msgmnb

単一のメッセージキューの最大サイズをバイト単位で定義します。**sysctl msgmnb** コマンドを使用して、システムの現在の **msgmnb** 値を確認します。

msgmni

メッセージキュー識別子の最大数 (つまりキューの最大数) を定義します。**sysctl msgmni** コマンドを使用して、システムの現在の **msgmni** 値を確認します。

shmall

一度にシステムで使用できる共有メモリー **pages** の合計量を定義します。たとえば、AMD64 アーキテクチャーおよび Intel 64 アーキテクチャーでは、ページは **4096** バイトになります。**sysctl shmall** コマンドを使用して、システムの現在の **shmall** 値を確認します。

shmmax

カーネルが許可する単一の共有メモリーセグメントの最大サイズをバイト単位で定義します。カーネルで、1Gb までの共有メモリーセグメントがサポートされるようになりました。**sysctl shmmax** コマンドを使用して、システムの現在の **shmmax** 値を確認します。

shmmni

システム全体の共有メモリーセグメントの最大数を定義します。いずれのシステムでもデフォルト値は **4096** です。

関連情報

- man ページの **sysvipc(7)** および **sysctl(8)**

35.6. メモリー関連のカーネルパラメーターの設定

パラメーターを一時的に設定すると、システム上でのパラメーターの効果を判断するのに便利です。パラメーター値が望ましい効果を確認すると、後でパラメーターを永続的に設定できます。

この手順では、メモリー関連のカーネルパラメーターを一時的に設定して永続的に設定する方法を説明します。

手順

- メモリー関連のカーネルパラメーターを一時的に設定するには、**/proc** ファイルシステムまたは **sysctl** ツール内の各ファイルを編集します。

たとえば、**vm.overcommit_memory** パラメーターを 1 に一時的に設定します。

```
# echo 1 > /proc/sys/vm/overcommit_memory
# sysctl -w vm.overcommit_memory=1
```

- メモリー関連のカーネルパラメーターを永続的に設定するには、**/etc/sysctl.conf** ファイルを編集し、設定をリロードします。
  たとえば、**vm.overcommit_memory** パラメーターを 1 に永続的に設定するには、以下を実行します。
  - **/etc/sysctl.conf** ファイルに以下の内容を追加します。

    ```
    vm.overcommit_memory=1
    ```

  - **/etc/sysctl.conf** ファイルから **sysctl** 設定を再読み込みします。

    ```
    # sysctl -p
    ```

関連情報

- man ページの **sysctl(8)**
- **proc(5)** の man ページ

関連情報

- Tuning Red Hat Enterprise Linux for IBM DB2

第36章 HUGE PAGE の設定

物理メモリーは、ページと呼ばれる固定サイズのブロックで管理されます。Red Hat Enterprise Linux 8 が対応する x86_64 アーキテクチャーでは、メモリーページのデフォルトサイズは **4 KB** です。このデフォルトのページサイズは、さまざまなワークロードをサポートする Red Hat Enterprise Linux などの一般的なオペレーティングシステムに適しています。

ただし、特定のアプリケーションは、特定のケースで大きなページサイズを使用する利点を得られます。たとえば、数百メガバイトまたは数千ギガバイトの大規模で比較的固定されたデータセットで動作するアプリケーションでは、**4 KB** ページを使用するとパフォーマンスの問題が発生する可能性があります。このようなデータセットには大量の **4 KB** ページが必要になるため、オペレーティングシステムや CPU でオーバーヘッドが発生する可能性があります。

本セクションでは、RHEL 8 で利用可能なヒュージページとその設定方法を説明します。

36.1. 利用可能な HUGE PAGE の機能

Red Hat Enterprise Linux 8 では、大規模なデータセットに対応するアプリケーションに Huge Page を使用し、このようなアプリケーションのパフォーマンスを向上できます。

以下は、RHEL 8 でサポートされる Huge Page メソッドです。

HugeTLB pages

HugeTLB ページも静的なヒュージページと呼ばれます。HugeTLB ページを予約する方法は 2 つあります。

- ブート時: メモリーが大幅に断片化されていないため、成功する可能性が高くなります。ただし、NUMA マシンでは、ページ数は NUMA ノード間で自動的に分割されます。

起動時に HugeTLB ページの動作に影響を与えるパラメーターの詳細は、起動時に HugeTLB ページを確保するためのパラメーター を参照してください。これらのパラメーターを使用して起動時に HugeTLB ページを設定する方法の詳細は、起動時の HugeTLB の設定 を参照してください。

- ランタイム時に: NUMA ノードごとにヒュージページを予約することができます。ランタイム予約がブートプロセスの早い段階で行われると、メモリーの断片化はより低くなります。

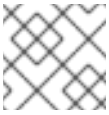
ランタイム時に HugeTLB ページの動作に影響を与えるパラメーターの詳細は、ランタイム時に HugeTLB ページを確保するためのパラメーター を参照し、これらのパラメーターを使用してランタイム時に HugeTLB ページを設定する方法の詳細は、ランタイム時の HugeTLB の設定 を参照してください。

Transparent HugePages (THP)

THP では、カーネルはヒュージページをプロセスに自動的に割り当てるため、静的な Huge Page を手動で予約する必要はありません。以下は、THP における動作の 2 つのモードです。

- **system-wide** で、カーネルはヒュージページを割り当て、プロセスが連続している仮想メモリー領域を使用しているたびに Huge Page をプロセスに割り当てようと試みます。

- **プロセスごと**: ここでは、カーネルは各プロセスのメモリー領域だけに割り当てるため、**madvise**() システムコールを使用して指定できるヒュージページを 1 つのみに割り当てます。



注記

THP 機能は、**2 MB** ページのみに対応します。

起動時に HugeTLB ページの動作に影響を与えるパラメーターの詳細は、透過的な HugePage の有効化 および 透過的な HugePage の無効化 を参照してください。

36.2. 起動時に HUGETLB ページを確保するためのパラメーター

システムの起動時に HugeTLB ページの動作に影響を与える場合は、次のパラメーターを使用します。

これらのパラメーターを使用してブート時に HugeTLB ページを設定する方法の詳細については、ブート時に HugeTLB を設定する を参照してください。

表36.1 起動時の HugeTLB ページの設定に使用されるパラメーター

パラメーター	説明	デフォルト値
hugepages	ブート時にカーネルに設定される永続ヒュージページの数を定義します。 NUMA システムでは、このパラメーターが定義されている Huge Page はノード間で均等に分割されます。 ランタイム時に **/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages** ファイルにあるノードの値を変更することで、起動時に Huge Page を特定のノードに割り当てることができます。	デフォルト値は **0** です。 起動時にこの値を更新するには、**/proc/sys/vm/nr_hugepages** ファイルでこのパラメーターの値を変更します。
hugepagesz	起動時にカーネルに設定される永続ヒュージページのサイズを定義します。	有効な値は **2 MB** および **1 GB** です。デフォルト値は **2 MB** です。
default_hugepagesz	起動時にカーネルに設定される永続 Huge Page のデフォルトのサイズを定義します。	有効な値は **2 MB** および **1 GB** です。デフォルト値は **2 MB** です。

36.3. 起動時の HUGETLB の設定

HugeTLB サブシステムがサポートするページサイズは、アーキテクチャーによって異なります。x86_64 アーキテクチャーは、**2 MB** の Huge Page および **1 GB** のギガンティックページをサポートします。

この手順では、システムの起動時に **1 GB** ページを予約する方法を説明します。

手順

手順

1. **1 GB** ページの HugeTLB プールを作成するには、**default_hugepagesz=1G** および **hugepagesz=1G** カーネルオプションを有効にします。

```
# grubby --update-kernel=ALL --args="default_hugepagesz=1G hugepagesz=1G"
```

2. **/usr/lib/systemd/system/** ディレクトリーに **hugetlb-gigantic-pages.service** という新しいファイルを作成し、以下の内容を追加します。

```
[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
```

3. **/usr/lib/systemd/** ディレクトリーに **hugetlb-reserve-pages.sh** という新しいファイルを作成し、以下の内容を追加します。
以下の内容を追加する場合、**number_of_pages** を、予約する 1GB ページ数に置き換え、**node** を、これらのページを予約するノードの名前に置き換えます。

```
#!/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
```

たとえば、**node0** で 2 つの **1 GB** ページ、**node1** で 1GB のページを予約するには、**number_of_pages** を、**node0** の場合は 2 に置き換え、**node1** の場合は 1 に置き換えます。

```
reserve_pages 2 node0
reserve_pages 1 node1
```

4. 実行可能なスクリプトを作成します。

```
# chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
```

5. 初期のブート予約を有効にします。

```
# systemctl enable hugetlb-gigantic-pages
```

注記

- 任意のタイミングで **nr_hugepages** に書き込みを行うことにより、ランタイム時にさらに **1 GB** ページを予約してみることができます。ただし、メモリーの断片化による障害を防ぐために、起動プロセスの早い段階で **1 GB** のページを予約します。
- 静的 Huge Page を確保することで、システムで利用可能なメモリー量を効果的に減らすことができます。ただし、メモリーの全容量を適切に使用できなくなります。予約された Huge Page の適切なサイズプールは、これを使用するアプリケーションにとって有益になりますが、予約済み Huge Page の過度なサイズや未使用のプールは最終的にシステムパフォーマンス全体に悪影響を及ぼします。予約済みの Huge Page プールを設定する場合は、システムによってメモリーの最大容量を適切に利用できるようになります。

関連情報

- **systemd.service(5)** man ページ
- **/usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt** ファイル

36.4. ランタイム時に HUGETLB ページを確保するためのパラメーター

実行時に HugeTLB ページの動作に影響を与える場合は、以下のパラメーターを使用します。

これらのパラメーターを使用してランタイム時に HugeTLB ページを設定する方法の詳細は、ランタイム時の HugeTLB の設定 を参照してください。

表36.2 ランタイム時に HugeTLB ページを設定するために使用されるパラメーター

パラメーター	説明	ファイル名
nr_hugepages	指定された NUMA ノードに割り当てられる指定したサイズの Huge Page の数を定義します。	**/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages**

パラメーター	説明	ファイル名
nr_overcommit_hugepages	オーバーコミットメモリーを介してシステムで作成され、使用できる追加の Huge Page の最大数を定義します。 このファイルにゼロ以外の値を書き込むと、永続 Huge Page プールが使い切られると、システムはカーネルの通常のページプールから その数の Huge Page を取得することを示しています。これら超過分の Huge Page は使用されなくなるので、これらは解放され、カーネルの通常のページプールに戻ります。	**/proc/sys/vm/nr_overcommit_hugepages**

36.5. ランタイム時の HUGETLB の設定

この手順では、20 2048 kB Huge Page を node2 に追加する方法を説明します。

要件に基づいてページを確保するには、以下を置き換えます。

- 20 (予約する Huge Page 数)
- Huge Page のサイズを含めた 2048kB
- ページを予約するノードのある node2。

手順

1. メモリー統計を表示します。

```
# 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. 指定のサイズの Huge Page 数をノードに追加します。

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

検証手順

- Huge Page の数が追加されていることを確認します。

```
# 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
```

関連情報

- **numastat(8)** の man ページ

36.6. 透過的な HUGEPAGE の有効化

Red Hat Enterprise Linux 8 では、THP がデフォルトで有効になっています。ただし、THP を有効または無効にすることができます。

この手順では、THP を有効にする方法を説明します。

手順

1. THP の現在の状態を確認します。

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. THP を有効にします。

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

3. アプリケーションが、必要以上のメモリーリソースを割り当てないようにするには、システム全体の透過的な Huge Page を無効にし、**madvise** を介して明示的に要求するアプリケーションに対してのみ有効にします。

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```

注記

短期的な割り当てのレイテンシーが低くなると、有効期間の長い割り当てで最適パフォーマンスをすぐに実現するよりも優先度が高くなります。この場合は、THP を有効にしたままでも直接圧縮を無効にできます。

直接圧縮は、Huge Page の割り当て中の同期メモリー圧縮です。直接圧縮を無効にすると、メモリーの保存は保証されませんが、頻繁なページ障害の発生時にレイテンシーが高くなる可能性が減ります。ワークロードが THP から著しく異なる場合に、パフォーマンスが低下する点に注意してください。直接圧縮を無効にします。

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
```

関連情報

- man ページの **madvise(2)**
- 透過的な HugePage の無効化

36.7. 透過的な HUGE PAGE の無効化

Red Hat Enterprise Linux 8 では、THP がデフォルトで有効になっています。ただし、THP を有効または無効にすることができます。

この手順では、THP を無効にする方法を説明します。

手順

1. THP の現在の状態を確認します。

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. THP を無効にします。

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

36.8. 翻訳されたバッファーサイズのページサイズの影響

ページテーブルからアドレスマッピングを読み取るのは、非常に時間がかかり、リソースの負荷が高くなります。そのため、CPU は、トランスレーションルックアサイドバッファー (TLB: Translation Lookaside Buffer) と呼ばれる、最近使用されたアドレスのキャッシュで構築されます。ただし、デフォルトの TLB は、特定のアドレスマッピングのみをキャッシュできます。

要求されたアドレスマッピングが TLB ミスと呼ばれる TLB にない場合、システムはページテーブルを読み込んで、物理から仮想アドレスへのマッピングを判断する必要があります。アプリケーションメモリー要件と、アドレスマッピングのキャッシュに使用されるページサイズ間の関係により、メモリー要件が高いアプリケーションは、最小限であるアプリケーションと比べて、TLB ミスによるパフォーマンスの低下の影響が高くなる可能性があります。したがって、可能であれば TLB ミスを回避するためには重要です。

HugeTLB 機能と Transparent Huge Page 機能の両方を使用すると、アプリケーションは **4 KB** よりも大きなページを使用できます。これにより、TLB に保存されているアドレスはより多くのメモリーを参照できます。これにより、TLB ミスが軽減され、アプリケーションのパフォーマンスが向上します。

第37章 SYSTEMTAP の使用

システム管理者は、SystemTap を使用して、実行中の Linux システムでバグやパフォーマンス問題の根本的な原因を特定することができます。

アプリケーション開発者は、SystemTap を使用して、アプリケーションが Linux システム内でどのように動作するかを詳細に監視できます。

37.1. SYSTEMTAP の目的

SystemTap は、オペレーティングシステム (特にカーネル) の動作を詳細に調査および監視するために使用できる追跡およびプロービングツールです。SystemTap は、**netstat**、**ps**、**top**、**iostat** などのツールの出力に似た情報を提供します。ただし、SystemTap は収集した情報をフィルタリング、分析するためのオプションがより多く用意されています。SystemTap スクリプトでは、SystemTap が収集する情報を指定します。

SystemTap は、カーネルアクティビティーを追跡するインフラストラクチャーを提供し、2 つの属性とこの機能を統合して、Linux 監視ツールの既存のスイートを補完することを目的としています。

柔軟性

SystemTap フレームワークを使用すると、さまざまなカーネル機能、システムコール、カーネルスペースで発生するその他のイベントについて調査および監視目的のシンプルなスクリプトを開発できます。つまり、SystemTap はツールというよりも、独自のカーネル固有のフォレンジックおよび監視ツールの開発を可能にするシステムといえます。

使いやすさ

SystemTap を使用すると、カーネルを再コンパイルしたり、システムを再起動したりせずに、カーネルのアクティビティーを監視できます。

37.2. SYSTEMTAP のインストール

SystemTap の使用を開始するには、必要なパッケージをインストールします。システムに複数のカーネルがインストールされている複数のカーネルで SystemTap を使用するには、カーネルバージョン ごと に必要な対応カーネルパッケージをインストールします。

前提条件

- デバッグリポジトリーおよびソースリポジトリーの有効化 で説明されているように、デバッグリポジトリーを有効にしている。

手順

1. 必要な SystemTap パッケージをインストールします。

```
# yum install systemtap
```

2. 必要なカーネルパッケージをインストールします。
   a. **stap-prep** の使用:

```
# stap-prep
```

   b. **stap-prep** が機能しない場合は、必要なカーネルパッケージを手動でインストールします。

```
# yum install kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -i)-
$(uname -r) kernel-devel-$(uname -r)
```

$(uname -i) は、システムのハードウェアプラットフォームに自動的に置き換えられ、**$(uname -r)** は、実行中のカーネルのバージョンに自動的に置き換えられます。

検証手順

- SystemTap でプローブするカーネルが現在使用中である場合 n には、インストールが成功したかどうかを確認します。

```
# stap -v -e 'probe kernel.function("vfs_read") {printf("read performed\n"); exit()}'
```

SystemTap デプロイメントに成功すると、以下のような出力が表示されます。

```
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)
```

出力の最後の 3 行 (**Pass 5** で開始) は、以下のようになります。

1. SystemTap は、正常にカーネルをプローブするインストルメーションを作成して実行しました。
2. SystemTap は、指定のイベントを検出しました (この場合、VFS の読み取り)。
3. SystemTap が有効なハンドラーを実行しました (テキストを出力した後、エラーなしで閉じました)。

37.3. SYSTEMTAP を実行する特権

SystemTap スクリプトを実行するには、システム権限の昇格が必要になりますが、場合によっては、権限のないユーザーが自身のマシン上で SystemTap インストルメンテーションを実行する必要がある場合があります。

ユーザーが root アクセスなしで SystemTap を実行できるようにするには、以下の **両方** のユーザーグループにユーザーを追加します。

stapdev

このグループのメンバーは **stap** を使用して SystemTap スクリプトを実行したり、**staprun** を使用して SystemTap インストルメンテーションモジュールを実行したりできます。

stap の実行では、SystemTap スクリプトがカーネルモジュールにコンパイルされ、それがカーネルに読み込まれます。これにはシステムに対する権限の昇格が必要となり、**stapdev** メンバーにはそれが付与されます。ただし、この権限は **stapdev** メンバーに有効な root アクセスも付与することに

なります。このため、**stapdev** グループのメンバーシップは、root アクセスを信頼して付与できるメンバーにのみ許可してください。

stapusr

このグループのメンバーが SystemTap インストルメンテーションモジュールの実行に使用できるのは、**staprun** のみです。また、これらのモジュールは **/lib/modules/kernel_version/systemtap/** ディレクトリーからしか実行できません。このディレクトリーの所有や書き込みが可能なのは root ユーザーだけでなければなりません。

37.4. SYSTEMTAP スクリプトの実行

SystemTap スクリプトは、標準入力またはファイルから実行できます。

SystemTap のインストールと合わせて配布されるサンプルスクリプトは、**/usr/share/systemtap/examples** ディレクトリーにあります。

前提条件

1. Installing Systemtap で説明されているように、SystemTap および関連する必須カーネルパッケージがインストールされている。
2. SystemTap スクリプトを通常のユーザーとして実行するには、そのユーザーを SystemTap グループに追加します。

```
# usermod --append --groups
stapdev,stapusr user-name
```

手順

- SystemTap スクリプトを実行します。
  - 標準入力の場合:

    ```
    # echo "probe timer.s(1) {exit()}" | stap -
    ```

    このコマンドは、**stap** に対して、**echo** で標準入力に渡したスクリプトを実行するように指示します。**stap** オプションを追加するには、**-** 文字の前に入力します。たとえば、このコマンドの結果を詳細化するには以下のコマンドを使用します。

    ```
    # echo "probe timer.s(1) {exit()}" | stap -v -
    ```

  - ファイルから:

    ```
    # stap file_name
    ```

第38章 SYSTEMTAP のクロスインストルメンテーション

SystemTap のクロスインストルメンテーションは、あるシステムで SystemTap スクリプトから SystemTap インストルメンテーションモジュールを作成し、SystemTap が完全にデプロイされていない別のシステムで使用します。

38.1. SYSTEMTAP のクロスインストルメンテーション

ユーザーが SystemTap スクリプトを実行すると、そのスクリプトからカーネルモジュールが構築されます。次に SystemTap はモジュールをカーネルに読み込みます。

通常、SystemTap スクリプトは SystemTap がデプロイされているシステムでのみ実行できます。SystemTap を 10 台のシステムで実行するには、このようなすべてのシステムに SystemTap をデプロイする必要があります。場合によっては、これは実現不可能または推奨されない場合があります。たとえば、企業のポリシーで、特定のマシンにコンパイラーやデバッグ情報を提供するパッケージのインストールが禁止されている場合には、SystemTap のデプロイメントができなくなります。

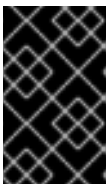
この状況を回避するために、**クロスインストルメンテーション** を使用します。クロスインストルメンテーションとは、あるコンピューターの SystemTap スクリプトから SystemTap インストルメンテーションモジュールを生成して、別のシステムで使用するプロセスです。このプロセスには、以下のような効果があります。

- 各種マシンのカーネル情報パッケージを単一のホストマシにインストールできる。

重要

カーネルのパッケージ化にバグがあると、インストールが妨げられる場合があります。このような場合に **ホストシステム** と **ターゲットシステム** の **kernel-debuginfo** および **kernel-devel** パッケージが同じでなければなりません。バグが発生した場合は、 https://bugzilla.redhat.com/ でバグを報告します。

- 生成された SystemTap インストルメンテーションモジュール (**systemtap-runtime**) を使用するには、**ターゲットマシン** ごとインストールする必要があるパッケージは 1 つだけです。



重要

構築された **インストルメンテーションモジュール** が機能するには、**ホストシステム** と **ターゲットシステム** が同一アーキテクチャーで同じ Linux ディストリビューションを実行している必要があります。

用語

インストルメンテーションモジュール

SystemTap スクリプトから構築したカーネルモジュール。SystemTap モジュールは **ホストシステム** 上に構築され、**ターゲットシステム** の **ターゲットカーネル** に読み込まれます。

ホストシステム

ターゲットシステム に読み込めるように (SystemTap スクリプトから) インストルメンテーションモジュールをコンパイルするシステム。

ターゲットシステム

(SystemTap スクリプトから) **インストルメンテーションモジュール** を構築するシステム。

ターゲットカーネル

ターゲットシステム のカーネル。**インストルメンテーションモジュール** を読み込み、実行するカーネルです。

38.2. SYSTEMTAP のクロスインストルメンテーションの初期化

SystemTap のクロスインストルメンテーションを初期化し、あるシステムで SystemTap スクリプトから SystemTap インストルメンテーションモジュールを構築して SystemTap が完全にデプロイされていない別のシステムで使用します。

前提条件

- SystemTap のインストール で説明されているように、SystemTap が **ホストシステム** にインストールされている。
- 各 **ターゲットシステム** に **systemtap-runtime** パッケージがインストールされている。

```
# yum install systemtap-runtime
```

- **ホストシステム** と **ターゲットシステム** の両方のアーキテクチャーが同じである。
- **ホストシステム** と **ターゲットシステム** の両方が Red Hat Enterprise Linux (例: Red Hat Enterprise Linux 8) の同じバージョンをしている。ただし、異なるマイナーバージョンを使用することは **可能です** (例: 8.1 および 8.2)。

重要

カーネルパッケージのバグが原因で複数の **kernel-debuginfo** と **kernel-devel** パッケージがシステムにインストールできない場合があります。このような場合は、**ホストシステム** と **ターゲットシステム** のマイナーバージョンが同じでなければなりません。バグが発生した場合は、https://bugzilla.redhat.com/ で報告してください。

手順

1. 各 **ターゲットシステム** で実行しているカーネルを確認します。

```
$ uname -r
```

ターゲットシステム ごとにこの手順を繰り返します。

2. Systemtap のインストール で説明されている方法に従って、**ホストシステム** で各 **ターゲットシステム** の **ターゲットカーネル** と関連パッケージをインストールします。

3. **ホストシステム** でインストルメンテーションモジュールを構築し、このモジュールをコピーして **ターゲットシステム** でこのモジュールを実行します。

   a. リモート実装の使用

   ```
   # stap --remote target_system script
   ```

   このコマンドは、指定したスクリプトを **ターゲットシステム** にリモートで実装します。これを正常に実行するには、 **ホストシステム** から **ターゲットシステム** に SSH 接続できるようしておく必要があります。

   b. 手動:

      i. **ホストシステム** でインストルメンテーションモジュールを構築します。

      ```
      # stap -r kernel_version script -m module_name -p 4
      ```

      ここでは、kernel_version は手順 1 で判断した **ターゲットカーネル** を、script は **インストルメンテーションモジュール** に変換するスクリプトを、module_name は **インストルメンテーションモジュール** の任意の名前を指します。**-p4** オプションは、SystemTap にコンパイルしたモジュールを読み込まないように指示します。

      ii. **インストルメンテーションモジュール**がコンパイルされたら、ターゲットシステムにコピーして、以下のコマンドを使用して読み込みます。

      ```
      # staprun module_name.ko
      ```

第39章 SYSTEMTAP でのネットワークアクティビティーの監視

/usr/share/systemtap/testsuite/systemtap.examples/ ディレクトリーで利用可能な SystemTap スクリプトの例を使用して、**systemtap-testsuite** パッケージをインストールし、システムのネットワークアクティビティーを監視して調べることができます。

39.1. SYSTEMTAP でのネットワークアクティビティーのプロファイル

nettop.stp のサンプルの SystemTap スクリプトを使用して、ネットワークアクティビティーのプロファイルを作成できます。このスクリプトは、システムでネットワークトラフィックを生成しているプロセスを追跡し、各プロセスに関する以下の情報を提供します。

PID
リスト表示されているプロセスの ID。

UID
ユーザー ID。ユーザー ID が 0 の場合は、root ユーザーを指します。

DEV
データの送受信に使用されるイーサネットデバイス (eth0、eth1 など)。

XMIT_PK
プロセスが送信したパケットの数。

RECV_PK
プロセスが受信したパケットの数。

XMIT_KB
プロセスにより送信されたデータの量 (キロバイト)。

RECV_KB
サービスが受信したデータの量 (キロバイト単位)。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **nettop.stp** スクリプトを実行します。

```
# stap  --example nettop.stp
```

nettop.stp スクリプトは、5 秒間隔でネットワークプロファイルのサンプリングを行います。

nettop.stp スクリプトの出力は、以下のようになります。

```
[...]
  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. SYSTEMTAP でネットワークソケットコードで呼び出される関数のトレース

socket-trace.stp のサンプルの SystemTap スクリプトを使用して、カーネルの net/socket.c ファイルから呼び出された関数を追跡できます。これにより、各プロセスがカーネルレベルでネットワークとどのように相互作用するかを、詳細に把握できます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **socket-trace.stp** スクリプトを実行します。

```
# stap  --example socket-trace.stp
```

socket-trace.stp スクリプトの 3 秒の抜粋は、以下のようになります。

```
[...]
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. SYSTEMTAP でのネットワークパケットドロップの監視

Linux のネットワークスタックは、様々な理由でパケットを破棄する場合があります。一部の Linux カーネルには、パケットが破棄される場所を追跡するトレースポイント **kernel.trace ("kfree_skb")** が組み込まれています。

dropwatch.stp SystemTap スクリプトは、**kernel.trace("kfree_skb")** を使用してパケットの破棄を追跡します。スクリプトは、5 秒間隔でパケットを破棄する場所を要約します。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **dropwatch.stp** スクリプトを実行します。

```
# stap  --example dropwatch.stp
```

dropwatch.stp スクリプトを 15 秒間実行すると、以下のような出力になります。

```
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
```

注記

パケットドロップの場所をより意味のあるものにするには、**/boot/System.map-$(uname -r)** を参照してください。このファイルは、各関数の開始アドレスのリストを表示し、**dropwatch.stp** スクリプトの出力内のアドレスを特定の関数名にマップできるようにします。**/boot/System.map-$(uname -r)** ファイルの以下のようなスニペットの場合、アドレス **0xffffffff8024cd0f** は関数 **unix_stream_recvmsg** に、アドレス **0xffffffff8044b472** は関数 **arp_rcv** にマッピングされます。

```
[...]
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
[...]
```

第40章 SYSTEMTAP でのカーネルアクティビティーのプロファイル

次のスクリプトを使用して、関数呼び出しを監視することにより、カーネルアクティビティーをプロファイリングできます。

40.1. SYSTEMTAP での関数呼び出しのカウント

functioncallcount.stp SystemTap スクリプトを使用して、特定のカーネル関数呼び出しを数えることができます。このスクリプトを使用して、複数のカーネル関数をターゲットにすることもできます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **functioncallcount.stp** スクリプトを実行します。

```
# stap --example functioncallcount.stp 'argument'
```

このスクリプトは、ターゲットのカーネル関数を引数として取ります。引数のワイルドカードを使用すると、ある程度まで複数のカーネル関数を対象にできます。

スクリプトの出力には、アルファベット順に、呼び出された関数の名前と、サンプル時間中に呼び出された回数が含まれています。

以下の例を考慮してください。

```
# stap -w -v --example functioncallcount.stp "*@mm*.c" -c /bin/true
```

ここでは、以下のようになります。

- -w : 警告を表示しません。
- -v : 起動したカーネルの出力を表示します。
- -c **コマンド** : コマンドの実行中に関数呼び出しを数えるように SystemTap に指示します (この例では /**bin**/**true**)。
この出力は、以下のようになります。

```
[...]
__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. SYSTEMTAP での関数呼び出しのトレース

para-callgraph.stp SystemTap スクリプトを使用して、関数呼び出しと関数戻りを追跡できます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **para-callgraph.stp** スクリプトを実行します。

```
# stap --example para-callgraph.stp 'argument1' 'argument2'
```

para-callgraph.stp スクリプトは、コマンドライン引数を 2 つ取ります。

1. その開始または終了が追跡対象となっている関数の名前。
2. オプションのトリガー関数。スレッド単位でのトレースを有効または無効にします。trigger function が終了していなければ、各スレッドにおける追跡は継続されます。

以下の例を考慮してください。

```
# stap -wv --example para-callgraph.stp 'kernel.function("*@fs/proc.c*")' 'kernel.function("vfs_read")' -c "cat /proc/sys/vm/* || true"
```

ここでは、以下のようになります。

- -w : 警告を表示しません。
- -v : 起動したカーネルの出力を表示します。
- -c **コマンド** : コマンドの実行中に関数呼び出しを数えるように SystemTap に指示します (この例では **/bin/true**)。

この出力は、以下のようになります。

```
[...]
   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. SYSTEMTAP でカーネルとユーザー空間で費やす時間の決定

thread-times.stp SystemTap スクリプトを使用して、指定したスレッドがカーネルまたはユーザー空間で費やす時間を指定できます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **thread-times.stp** スクリプトを実行します。

  ```
  # stap --example thread-times.stp
  ```

  このスクリプトでは、5 秒間に CPU 時間を使用している上位 20 のプロセスと、サンプル中に作成された CPU ティックの合計数が表示されます。このスクリプトの出力は、各プロセスが使用した CPU 時間のパーセント表示と、その時間がカーネルスペースかユーザースペースで費やされたかも示します。

  ```
  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. SYSTEMTAP を使用したポーリングアプリケーションの監視

timeout.stp SystemTap スクリプトを使用して、ポーリングしているアプリケーションを特定し、監視できます。これにより、不要なポーリングや過剰なポーリングの追跡が可能になります。これにより、CPU 使用率や消費電力の改善領域を特定しやすくなります。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **timeout.stp** スクリプトを実行します。

  ```
  # stap --example timeout.stp
  ```

  このスクリプトは、各アプリケーションが時間とともに以下のシステムコールを使用する回数を追跡します。

- **poll**
- **select**
- **epoll**
- **itimer**
- **futex**
- **nanosleep**
- **signal**

この例の出力では、どのプロセスがどのシステムコールを使用したか、また何回目のシステムコールを使用したかを確認できます。

```
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. SYSTEMTAP で最も頻繁に使用されるシステムコールの追跡

topsys.stp SystemTap スクリプトを使用すると、5 秒間隔でシステムが使用するシステムコールの上位 20 件をリスト表示できます。また、同期間に各システムコールが使用された回数も表示されます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **topsys.stp** スクリプトを実行します。

```
# stap --example topsys.stp
```

以下の例を考慮してください。

```
# stap -v --example topsys.stp
```

ここで、-v は、起動しているカーネルの出力を表示します。

この出力は、以下のようになります。

```
---------------------------------------------------------------
                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. SYSTEMTAP を使用したプロセスごとのシステムコールボリュームの追跡

syscalls_by_proc.stp SystemTap スクリプトを使用すると、どのプロセスが最大量のシステムコールを実行しているかを確認できます。これは、システムコールのほとんどを実行している 20 のプロセスを表示します。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **syscalls_by_proc.stp** スクリプトを実行します。

```
# stap --example syscalls_by_proc.stp
```

syscalls_by_proc.stp スクリプトの出力は、以下のようになります。

```
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
[...]
```

第41章 SYSTEMTAP でのディスクおよび I/O アクティビティーの監視

次のスクリプトを使用して、ディスクと I/O のアクティビティーを監視できます。

41.1. SYSTEMTAP でのディスクの読み取り/書き込みトラフィックの概要

disktop.stp SystemTap スクリプトを使用して、システムで最も重いディスク読み取りおよび書き込みを実行しているプロセスを特定できます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **disktop.stp** スクリプトを実行します。

```
# stap --example disktop.stp
```

このスクリプトでは、ディスクへの最も重い読み取りまたは書き込みを行う上位 10 プロセスが表示されます。

この出力には、リスト表示されているプロセスごとに、以下のデータが含まれます。

UID
ユーザー ID。**0** のユーザー ID は、root ユーザーを指します。

PID
リスト表示されているプロセスの ID。

PPID
リスト表示されているプロセスの親プロセスのプロセス ID。

CMD
リスト表示されているプロセスの名前。

デバイス
リスト表示されているプロセスが、読み取りまたは書き込みを行っているストレージデバイス。

T
リスト表示されているプロセスが実行するアクションの種類。**W** は書き込みを、**R** は読み取りを指します。

BYTES
ディスクに対して読み書きされるデータの量。

disktop.stp スクリプトの出力は、以下のようになります。

```
[...]
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. SYSTEMTAP での各ファイルの読み取りまたは書き込みの I/O 時間の追跡

iotime.stp SystemTap スクリプトを使用して、各プロセスが任意のファイルへの読み取りまたは書き込みを行う場合にかかる時間を監視できます。これにより、システムへの読み込みに時間がかかっているファイルを判断する上で役立ちます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **iotime.stp** スクリプトを実行します。

  ```
  # stap --example iotime.stp
  ```

  このスクリプトは、システムコールが開いたり、閉じたり、読み込みを行ったり、ファイルに書き込んだりするたびに追跡します。システムコールがアクセスするファイルごとに、読み取りもしくは書き込みが終了するまでの時間をマイクロ秒単位でカウントし、読み取りもしくは書き込みされたデータ量をバイト単位で追跡します。

  この出力には、以下が含まれます。

- タイムスタンプ (マイクロ秒)
- プロセス ID およびプロセス名
- **access** フラグまたは **iotime** フラグ
- アクセスしたファイル
  プロセスがデータの読み取りまたは書き込みが可能であった場合は、アクセス行と **iotime** 行のペアが一緒に表示されます。アクセス行は、指定したプロセスがファイルのアクセスを開始した時間を指します。アクセス行の末尾には、読み取りまたは書き込みのデータ量が表示されます。**iotime** の行には、プロセスが読み取りまたは書き込みを実行するのにかかった時間がマイクロ秒単位で表示されます。

iotime.stp スクリプトの出力は、以下のようになります。

```
[...]
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. SYSTEMTAP で累積 I/O の追跡

traceio.stp SystemTap スクリプトを使用して、システムへの I/O の累積量を追跡できます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **traceio.stp** スクリプトを実行します。

```
# stap --example traceio.stp
```

このスクリプトでは、時間の経過とともに I/O トラフィックを生成する上位 10 個の実行ファイルが出力されます。また、これらの実行可能ファイルが実行した I/O 読み取りおよび書き込みの累積量も追跡します。この情報は追跡され、1 秒間隔で降順で出力されます。

traceio.stp スクリプトの出力は、以下のようになります。

```
[...]
          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. SYSTEMTAP を使用した、特定デバイスでの I/O アクティビティーの監視

traceio2.stp SystemTap スクリプトを使用して、特定のデバイスにおける I/O アクティビティーを監視できます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **traceio2.stp** スクリプトを実行します。

```
# stap --example traceio2.stp 'argument'
```

このスクリプトは、デバイス番号全体を引数として取ります。この番号を確認するには、以下のコマンドを実行します。

```
# stat -c "0x%D" directory
```

監視するデバイスの**ディレクトリー** がある場所。

この出力には、以下が含まれます。

- 読み取りまたは書き込みを実行するプロセスの名前と ID
- 実行中の機能 (**vfs_read** または **vfs_write**)
- カーネルデバイス番号

以下のような **# stap traceio2.stp 0x805** の出力を検討してください。

```
[...]
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. SYSTEMTAP を使用したファイルの読み取りと書き込みの監視

inodewatch.stp SystemTap スクリプトを使用すると、ファイルの読み取りと書き込みをリアルタイムで監視できます。

前提条件

- SystemTap のインストール の説明に従って、SystemTap をインストールしている。

手順

- **inodewatch.stp** スクリプトを実行します。

```
# stap --example inodewatch.stp 'argument1' 'argument2' 'argument3'
```

スクリプト **inodewatch.stp** では、コマンドラインの引数を 3 つ使用します。

1. ファイルのメジャーデバイス番号。
2. ファイルのマイナーデバイス番号。

3. ファイルの inode 番号。

この番号は、以下を使用して取得できます。

```
# stat -c '%D %i' filename
```

filename は絶対パスです。

以下の例を見てみましょう。

```
# stat -c '%D %i' /etc/crontab
```

出力は以下のようになります。

```
805 1078319
```

ここでは、以下のようになります。

- **805** は、ベース 16 (16 進数) のデバイス番号です。最後の 2 桁はマイナーデバイス番号で、残りの 2 桁はメジャー番号です。
- **1078319** は、inode 番号です。

/etc/crontab の監視を開始するには、次のコマンドを実行します。

```
# stap inodewatch.stp 0x8 0x05 1078319
```

最初の 2 つの引数では、基数 16 の数に 0x の接頭辞を使用する必要があります。

この出力には、以下が含まれます。

- 読み取りまたは書き込みを実行するプロセスの名前と ID
- 実行中の機能 (**vfs_read** または **vfs_write**)
- カーネルデバイス番号

この例の出力は、以下のようになります。

```
cat(16437) vfs_read 0x800005/1078319
cat(16437) vfs_read 0x800005/1078319
```

第42章 BPF コンパイラーコレクションでシステムパフォーマンスの分析

システム管理者として BPF コンパイラーコレクション (BCC) ライブラリーで Linux オペレーティングシステムのパフォーマンスを分析するツールを作成します。ただし、他のインターフェイス経由での取得は困難な場合があります。

42.1. BCC-TOOLS パッケージのインストール

bcc-tools パッケージをインストールします。これにより、依存関係として BPF Compiler Collection (BCC) ライブラリーもインストールされます。

手順

1. **bcc-tools** をインストールします。

```
# yum install bcc-tools
```

   BCC ツールは、**/usr/share/bcc/tools/** ディレクトリーにインストールされます。

2. 必要に応じて、ツールを検証します。

```
# 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
...
```

   上記のリストにある **doc** ディレクトリーには、各ツールのドキュメントが含まれます。

42.2. BCC-TOOLS でパフォーマンスの分析

BPF Compiler Collection (BCC) ライブラリーから事前に作成された特定のプログラムを使用して、システムパフォーマンスをイベントごとに効率的かつセキュアに分析します。BCC ライブラリーで事前作成されたプログラムセットは、追加プログラム作成の例として使用できます。

前提条件

- bcc-tools パッケージがインストールされている
- root 権限がある。

execsnoop を使用したシステムプロセスの検証

1. 1 つの端末で **execsnoop** プログラムを実行します。

```
# /usr/share/bcc/tools/execsnoop
```

2. たとえば、別のターミナルで次のように実行します。

```
$ ls /usr/share/bcc/tools/doc/
```

これにより、**ls** コマンドの短命プロセスが作成されます。

3. **execsnoop** を実行している端末は、以下のような出力を表示します。

```
PCOMM PID    PPID   RET ARGS
ls    8382   8287     0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

execsnoop プログラムは、新しいプロセスごとに出力行を出力するため、システムリソースを消費します。また、**ls** などの非常に短期間に実行されるプログラムのプロセスを検出します。なお、ほとんどの監視ツールはそれらを登録しません。

execsnoop 出力には以下のフィールドが表示されます。

PCOMM
: 親プロセス名。(**ls**)

PID
: プロセス ID。(**8382**)

PPID
: 親プロセス ID。(**8287**)

RET
: **exec()** システムコールの戻り値 (**0**)。プログラムコードを新規プロセスに読み込みます。

ARGS
: 引数を使用して起動したプログラムの場所。

execsnoop の詳細、例、およびオプションを確認するには、**/usr/share/bcc/tools/doc/execsnoop_example.txt** ファイルを参照してください。

exec() の詳細は、**exec(3)** man ページを参照してください。

opensnoop を使用した、コマンドにより開かれるファイルの追跡

1. 1 つのターミナルで **opensnoop** プログラムを実行します。

```
# /usr/share/bcc/tools/opensnoop -n uname
```

上記の出力では、**uname** コマンドのプロセスによってのみ開かれるファイルの内容が出力されます。

2. 別のターミナルで、次のように実行します。

```
$ uname
```

上記のコマンドは、特定のファイルを開きます。このファイルは次のステップでキャプチャーされます。

3. **opensnoop** を実行している端末は、以下のような出力を表示します。

```
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 プログラムは、システム全体で **open()** システム呼び出しを監視し、**uname** が開こうとしたファイルごとに出力行を出力します。

opensnoop 出力には、以下のフィールドが表示されます。

PID
: プロセス ID。(**8596**)

COMM
: プロセス名。(**uname**)

FD
: ファイルの記述子。開いたファイルを参照するために **open()** が返す値です。(**3**)

ERR
: すべてのエラー。

PATH
: **open()** で開こうとしたファイルの場所。

コマンドが、存在しないファイルを読み込もうとすると、**FD** コラムは **-1** を返し、**ERR** コラムは関連するエラーに対応する値を出力します。その結果、**opennoop** は、適切に動作しないアプリケーションの特定に役立ちます。

opensnoop の詳細、例、およびオプションを確認するには、**/usr/share/bcc/tools/doc/opensnoop_example.txt** ファイルを参照してください。

open() の詳細は、**open(2)** man ページを参照してください。

ディスク上の I/O 操作を調べるための biotop の使用

1. 1 つのターミナルで **biotop** プログラムを実行します。

```
# /usr/share/bcc/tools/biotop 30
```

このコマンドにより、ディスク上で I/O 操作を実行する上位のプロセスを監視できます。この引数は、コマンドが 30 秒の概要を生成するようにします。

注記

引数を指定しないと、デフォルトでは 1 秒ごとに出力画面が更新されます。

2. 別の端末で、たとえば次のように実行します。

```
# dd if=/dev/vda of=/dev/zero
```

上記のコマンドは、ローカルのハードディスクデバイスからコンテンツを読み込み、出力を **/dev/zero** ファイルに書き込みます。この手順では、**biotop** を示す特定の I/O トラフィックを生成します。

3. **biotop** を実行している端末は、以下のような出力を表示します。

```
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 出力には、以下のフィールドが表示されます。

PID
: プロセス ID。(**9568**)

COMM
: プロセス名。(**dd**)

DISK
: 読み取り操作を実行するディスク。(**vda**)

I/O
: 実行された読み取り操作の数。(16294)

Kbytes
: 読み取り操作によって使用したバイト数 (KB)。(14,440,636)

AVGms
: 読み取り操作の平均 I/O 時間。(3.69)

biotop の詳細、例、およびオプションを確認するには、**/usr/share/bcc/tools/doc/biotop_example.txt** ファイルを参照してください。

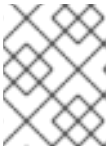
dd の詳細は、**dd(1)** man ページを参照してください。

xfsslower を使用した、予想外に遅いファイルシステム動作の明確化

1. 1 つのターミナルで **xfsslower** プログラムを実行します。

```
# /usr/share/bcc/tools/xfsslower 1
```

上記のコマンドは、XFS ファイルシステムが、読み込み、書き込み、開く、または同期 (**fsync**) 操作を実行するのに費やした時間を測定します。**1** 引数を指定すると、1 ms よりも遅い操作のみが表示されます。



注記

引数を指定しないと、**xfsslower** はデフォルトで 10 ms よりも低速な操作を表示します。

2. 別のターミナルで、たとえば次のように入力します。

```
$ vim text
```

上記のコマンドは、**vim** エディターでテキストファイルを作成し、XFS ファイルシステムと特定の対話を開始します。

3. **xfsslower** を実行している端末は、前の手順でファイルを保存した場合と同様の内容を示しています。

```
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'
...
```

上記の各行はファイルシステム内の操作を表し、特定のしきい値よりも時間がかかります。**xfsslower** は、操作に想定以上に時間がかかるなど、ファイルシステムで発生し得る問題を表面化するのに適しています。

xfsslower 出力には、以下のフィールドが表示されます。

COMM
: プロセス名。(**b'bash'**)

T
: 操作の種類。(**R**)

- **R**ead
- **W**rite
- **S**ync

OFF_KB
: ファイルオフセット (KB)。(0)

FILENAME
: 読み取り、書き込み、または同期中のファイル。

xfsslower の詳細、例、およびオプションについては、**/usr/share/bcc/tools/doc/xfsslower_example.txt** ファイルを参照してください。

fsync の詳細は、man ページの **fsync(2)** を参照してください。