Red Hat Enterprise Linux 7

パフォーマンスチューニングガイド

RHEL 7 でサブシステムのスループットの監視および最適化

Last Updated: 2024-09-24

Red Hat Enterprise Linux 7 パフォーマンスチューニングガイド

RHEL 7 でサブシステムのスループットの監視および最適化

Milan Navrátil
Red Hat Customer Content Services

Laura Bailey
Red Hat Customer Content Services

Charlie Boyle
Red Hat Customer Content Services

編集者

Marek Suchánek
Red Hat Customer Content Services
msuchane@redhat.com

法律上の通知

概要

Red Hat Enterprise Linux 7 パフォーマンスチューニングガイドでは Red Hat Enterprise Linux 7 のパフォーマンスを最適化する方法について説明しています。また、Red Hat Enterprise Linux 7 でパフォーマンス関連のアップグレードを行う方法についても触れています。 パフォーマンスチューニングガイドではフィールドテストで証明された手順しか掲載していません。ただし、予想される設定はすべて実稼動システムに適用する前にテスト環境でセットアップし試験を行ってください。また、チューニングを行う前に全データおよび設定のバックアップを取っておくことも推奨しています。 専門知識を深めるには、Red Hat Enterprise Linux パフォーマンスチューニング (RH442) トレーニングコースの受講を推奨します。

目次

第1章 はじめに

各 Red Hat Enterprise Linux 7 マイナーリリースで導入された機能については、Release Notes で各マイナーバージョンのリリースノートを参照してください。

パフォーマンスチューニングガイドは、特定の目的のために Red Hat Enterprise Linux 7 を設定するさまざまなサブシステムを最適化するための包括的なガイドです。本書では、Red Hat Enterprise Linux 7 で利用可能なパフォーマンス監視とチューニングツールについても簡単に説明しています。

チューニングを開始する前に、Red Hat は以下のことを行うことを強く推奨します。

設定前のバックアップ

Red Hat Enterprise Linux 7 のデフォルト設定は、負荷がそれほど大きくない状態で実行されているほとんどのサービスに適しています。特定のサブシステムのパフォーマンスを向上させると、別のシステムに悪い影響が及ぶことがあります。システムをチューニングする前に、すべてのデータおよび設定情報をバックアップしてください。

本番稼働用でない設定のテスト

パフォーマンスチューニングガイドで示されている手順は Red Hat エンジニアによってラボと現場で入念にテストされています。ただし、Red Hat は、これらの設定を本番稼働システムに適用する前に、計画されたすべての設定を安全なテスト環境でテストすることを推奨します

本書の対象読者

パフォーマンスチューニングガイドは、主に 2 つの異なるが重複する読者向けに書かれています。

システム管理者

パフォーマンスチューニングガイドでは、システム管理者が特定の目的のために Red Hat Enterprise Linux 7 を最適化できるよう各設定オプションの効果について詳しく説明します。本書の手順は Red Hat Certified Engineer (RHCE) 認定書または同等の経験 (3〜5 年の Linux ベースシステムのデプロイおよび管理経験) を持つシステム管理者向けです。

システムおよびビジネスアナリスト

本書では、Red Hat Enterprise Linux 7 のパフォーマンス機能について高度な説明をします。特定の負荷でサブシステムのパフォーマンスがどのように変化するかについての情報が提供されるため、アナリストは Red Hat Enterprise Linux 7 が各ユースケースに適しているかどうかを判断できます。

可能な場合、パフォーマンスチューニングガイドでは、読者に詳細なドキュメンテーションが紹介されます。これにより、読者は、インフラストラクチャーとデプロイメントの提案に必要な詳細なデプロイメントおよび最適化ストラテジーを作成するのに必要となる詳しい知識を得ることができます。

第2章 パフォーマンス監視ツール

本章では Red Hat Enterprise Linux 7 で用意されているパフォーマンスの監視および設定ツールのいくつかについて簡単に説明しています。可能な限り、ツールの使い方や実際にツールを使用して解決できる実践例などの詳細が記載された参考文献を掲載しています。

Red Hat Enterprise Linux での使用に適したパフォーマンス監視ツールの全リストについてはナレッジベース https://access.redhat.com/site/solutions/173863 をご覧ください。

2.1. /PROC

/proc ファイルシステム は、Linux カーネルの現在の状態を表すファイルの階層を含むディレクトリーです。ユーザーやアプリケーションはこのファイルシステムでシステムのカーネル状態を確認することができます。

/proc ディレクトリーには、システムハードウェアおよび現在実行中のプロセスに関する情報も含まれます。**/proc** ファイルシステムのほとんどのファイルは読み取り専用ですが、一部のファイル（主に /proc/sys のファイル）をユーザーおよびアプリケーションが操作して、設定の変更をカーネルに伝達できます。

/proc ディレクトリーでファイルを表示および編集する方法は、Red Hat Enterprise Linux 7 システム管理者のガイド を参照してください。

2.2. GNOME システムモニター

GNOME デスクトップ環境には、システム動作を監視、修正する際に役立つグラフィカルツールが収納されています。基本的なシステム情報を表示し、システムプロセスやリソース、ファイルシステムの使用量などを監視することができます。

システムモニター には 4 種類のタブがあり、システムに関するさまざまな情報を各タブに表示します。

システム
: このタブには、システムのハードウェアとソフトウェアに関する基本情報が表示されます。

プロセス
: 実行中のプロセスおよびそれらプロセスの関係に関する詳細情報を表示します。表示されたプロセスにフィルターをかけ特定のプロセスを見つけやすくすることができます。また、表示されたプロセスに起動、停止、強制終了、優先順位の変更などの動作を実行することもできます。

リソース
: このタブには、現在の CPU 時間の使用状況、メモリーとスワップスペースの使用状況、およびネットワークの使用状況が表示されます。

ファイルシステム
: マウントされているファイルシステムの全リスト、ファイルシステムのタイプやマウントポイント、メモリー使用量など各ファイルシステムの基本情報が表示されます。

システムモニターを起動するには Super キー (Win キー) を押してアクティビティー画面に入り システムモニター と入力してエンターを押します。

システムモニターに関する詳細は、アプリケーションのヘルプメニューを参照するか、Red Hat Enterprise Linux 7 システム管理者のガイドを参照してください。

2.3. ビルトインのコマンドラインツール

Red Hat Enterprise Linux 7 ではコマンドラインからシステムを監視できるツールをいくつか提供しています。これらのツールの利点は、ランレベル 5 以外で使用できる点です。本章では各ツールについて簡潔に説明し、各ツールの最適な使用方法に関する詳細が含まれるリンクを紹介します。

2.3.1. top

top ツールは procps-ng パッケージで提供され、稼働中のシステムのプロセスを動的に表示します。システム要約や Linux カーネルで現在管理されているタスクリストなどさまざまな情報を表示させることができます。また、限られてはいますがプロセスを操作したり、システムの再起動後も維持できる設定変更を行うなどの機能も備わっています。

デフォルトではプロセスは CPU 使用率に応じた順番で表示されるため、最もリソースを消費しているプロセスを簡単に見つけることができます。必要に応じた使用状況の統計に注目できるよう top で表示させる情報およびその動作はいずれも高度な設定が可能です。

top の使い方については man ページをご覧ください。

```
$ man top
```

2.3.2. ps

ps ツールは procps-ng パッケージで提供され、選択した実行中プロセスのグループのスナップショットを取得します。デフォルトでは、この対象グループは現行ユーザーが所有者のプロセスで ps を実行している端末に関連付けされているプロセスに限られます。

ps では top より詳細な情報を表示させることが可能ですが、デフォルトの表示はこのデータのスナップショットひとつのみで、プロセス ID 順に表示されます。

ps の使い方については man ページをご覧ください。

```
$ man ps
```

2.3.3. 仮想メモリーの統計 (vmstat)

仮想メモリー統計ツール vmstat ではシステムのプロセス、メモリー、ページング、ブロック入出力、割り込み、CPU 親和性などに関するインスタントレポートを提供します。サンプリングの間隔を設定できるためほぼリアルタイムに近いシステムのアクティビティーを観察することができます。

vmstat は、procps-ng パッケージにより提供されます。vmstat の使い方については man ページをご覧ください。

```
$ man vmstat
```

2.3.4. System Activity Reporter (sar)

System Activity Reporter (sar) はその日発生したシステムアクティビティーを収集および報告します。デフォルトの出力では、その日の始まり (00:00:00 -システムクロックに依存) から 10 分間隔で CPU の使用量が表示されます。

-i オプションを使用して間隔を秒単位で設定することもできます。**sar -i 60** は sar に対して毎分 CPU 使用率をチェックするように指示します。

sar は手作業でシステムアクティビティーの定期レポートを作成する top に代わる便利なレポート作成ツールになります。このツールは sysstat パッケージで提供されます。詳細については man ページをご覧ください。

```
$ man sar
```

2.4. PERF

perf ツール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。perf の各種サブコマンドは一般的なパフォーマンスイベントの統計を表示および記録したり、記録したデータを分析して報告したりします。

perf とそのサブコマンドの詳細については、「perf」 を参照してください。

また、より詳しい説明はRed Hat Enterprise Linux 7 開発者ガイド を参照してください。

2.5. TURBOSTAT

turbostat は kernel-tools パッケージで提供されます。Intel® 64 プロセッサーで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値、電力使用量などを報告します。

Turbostat は電力使用やアイドル時間などが非効率なサーバーを特定する際に役に立ちます。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。

Turbostat の実行には root 権限が必要になります。また、以下のプロセッサーサポートも必要になります。

- 不変タイムスタンプカウンター
- APERF モデル固有のレジスター
- MPERF モデル固有のレジスター

turbostat の出力およびその読み取り方法の詳細は、「turbostat」 を参照してください。

turbostat の詳細は、の man ページを参照してください。

```
$ man turbostat
```

2.6. IOSTAT

iostat ツールは sysstat パッケージで提供され、管理者が物理ディスク間で入出力負荷のバランスを取る方法を決定できるように、システムの入出力デバイスの読み込みを監視および報告します。**iostat** ツールは、**iostat** が最後に実行された時点または起動以降、プロセッサーまたはデバイスの使用状況を報告します。iostat(1) man ページで定義されているパラメーターを使用すると、特定のデバイスの報告の出力のみを表示できます。**await** の値とその値が高い原因の詳細については、Red Hat ナレッジベースの記事 iostat によって報告される "await" の意味は何ですか？を参照してください。

2.7. IRQBALANCE

irqbalance は、システムパフォーマンスを向上させるために、プロセッサー全体にハードウェア割り込みを分散するコマンドラインツールです。**irqbalance** の詳細は、「irqbalance」 または man ページを参照してください。

```
$ man irqbalance
```

2.8. SS

ss はソケットに関する統計情報を表示するコマンドラインツールです。長期間にわたるデバイスのパフォーマンスを評価することができます。デフォルトでは接続を確立しているオープンでリッスンしていない TCP ソケットを表示しますが、フィルターをかけ統計情報を特定のソケットに限定するオプションも用意されています。

Red Hat Enterprise Linux 7 では ss は netstat を導入して使用することを推奨しています。

一般的な使用方法の 1 つに、ソケットを使用して TCP ソケット、メモリー使用量、およびプロセスに関する詳細情報（内部情報を含む）を表示する **ss -tmpie** があります。

ss は、iproute パッケージで提供されます。詳細は man ページをご覧ください。

```
$ man ss
```

2.9. NUMASTAT

numastat ツールは、NUMA ノード単位でのプロセスおよびオペレーティングシステムのメモリー統計を表示します。

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の NUMA ヒット数ごとの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。**numastat** は、システム内の NUMA ノード間でシステムおよびプロセスメモリーを分散する方法を示す多数のコマンドラインオプションも提供します。

ノードごとの **numastat** 出力を CPU ごとの 上位 出力と相互参照して、メモリーが割り当てられている同じノードでプロセススレッドが実行していることを確認すると便利です。

numastat は **numactl** パッケージで提供されます。numastat の使い方については、「numastat」 を参照してください。numastat の詳細は man ページをご覧ください。

```
$ man numastat
```

2.10. NUMAD

numad は自動の NUMA 親和性管理デーモンです。システム内の NUMA トポロジーとリソース使用量を監視して、動的に NUMA のリソース割り当てと管理 (つまりシステムパフォーマンス) を改善します。システムの負荷に応じて numad はパフォーマンス基準の最大 50% の改善を実現します。また、各種のジョブ管理システムによるクエリーに対しそのプロセスに適した CPU とメモリーリソースの初期バインディングを提供するプレプレースメントアドバイスのサービスも用意しています。

numad は、**/proc** ファイルシステムの情報に定期的にアクセスすることで、ノードごとに利用可能なシステムリソースを監視します。指定されたリソース使用量のレベルを維持、必要に応じて NUMA ノード間でプロセスを移動しリソース割り当てバランスの再調整を行います。システムの NUMA ノードのサブセットで使用量が著しいプロセスを特定し隔離することで最適な NUMA パフォーマンスの実現を目指します。

numad は主に著しいリソース量を消費し、システムリソース全体の中の任意のサブセット内に限定されたプロセスで長期に渡り実行しているプロセスが複数あるシステムに役立ちます。また、複数 NUMA ノードに値するリソースを消費するアプリケーションにも有効な場合があります。ただし、システムリソース消費率の増加に伴い numad で得られる効果は低減します。

プロセスの実行時間がほんの数分であったり、消費するリソースが多くない場合、numad によるパフォーマンスの改善はあまり期待できません。大型のメモリー内データベースなど、予測不可能なメモリーアクセスパターンが継続的に見られるようなシステムの場合も numad による効果は期待できません。

numad の使い方については、「numad を使用した NUMA 親和性の自動管理」、「numad」、man ページなどをご覧ください。

```
$ man numad
```

2.11. SYSTEMTAP

SystemTap はトレースおよびプローブを行うためのツールです。オペレーティングシステム (特にカーネル) のアクティビティーを詳細に監視し分析を行います。top、ps、netstat、iostat などのツールの出力と同様の情報を提供しますが、収集したデータのフィルタリングや分析を行う際より多くのオプションが用意されています。

SystemTap を使用するとシステムアクティビティーやアプリケーション動作に関するより詳細で正確な分析ができるため、的確に弱点を見つけることができるようになります。

System Tap に関する詳細は、Red Hat Enterprise Linux 7 SystemTap ビギナーズガイド と Red Hat Enterprise Linux 7 SystemTap タップセットリファレンス を参照してください。

2.12. OPROFILE

OProfile はシステム全体のパフォーマンスを監視するツールです。プロセッサーのパフォーマンス監視専用ハードウェアを使用しカーネルやシステム実行可能ファイルに関する情報を読み出し、特定イベントの発生頻度、たとえばメモリー参照の発生時期や L2 キャッシュ要求の回数、ハードウェア要求の受信回数などを測定します。また、プロセッサーの使用量やもっとも頻繁に使用されているアプリケーションやサービスの特定にも使用できます。

ただし、OProfile にはいくつか制限があります。

- パフォーマンスのモニタリングサンプルが正確ではない場合があります。プロセッサーは順番通りに指示を実行しない場合があるので、割り込みを発生させた指示自体ではなく、近辺の指示からサンプルを記録する可能性があります。
- OProfile はプロセスが複数回にわたって開始、停止することを想定しています。このため、複数の実行からのサンプルの累積が可能です。前回の実行から得たサンプルデータの消去が必要な場合があります。
- アクセスが CPU に限定されているプロセスの問題の特定に特化しています。そのため、他のイベントでロックを待機している間はスリープ状態のプロセスを特定する場合には役に立ちません。

OProfile の詳細については、「OProfile」 または Red Hat Enterprise Linux 7 システム管理者のガイド を参照してください。または、**/usr/share/doc/oprofile-*version*** にあるシステムのドキュメントを参照してください。

2.13. VALGRIND

Valgrind はアプリケーションのパフォーマンスを改善するため検出とプロファイリングを行うツールを提供します。メモリーやスレッド関連のエラーの他、ヒープ、スタック、アレイのオーバーランなどを検出できるため、アプリケーションコード内のエラーを簡単に見つけ出して修正することができるようになります。また、キャッシュやヒープ、分岐予測のプロファイリングを行い、アプリケーションの速度を高めメモリーの使用量を最小限に抑える可能性のある要因を特定することもできます。

Valgrind はアプリケーションをシンセティック CPU で実行して既存アプリケーションコードのインストルメントを行いその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。インストルメント化のレベルは、使用している Valgrind ツールとその設定によって異なりますが、インストルメントしたコードの実行は通常の実行より 4 倍から 50 倍の時間がかかるので注意してください。

Valgrind は再コンパイルせずにそのままアプリケーション上で使用できます。しかし、Valgrind はコード内の問題の特定にデバッグ情報を使うので、デバッグ情報を有効にしてアプリケーションおよびサポートライブラリーをコンパイルしていない場合は、再コンパイルしてデバッグ情報を含ませることを推奨しています。

Valgrind はデバッグ効率を高めるため GNU Project Debugger (gdb) を統合しています。

Valgrind および付属のツールはメモリーのプロファイルを行う場合に便利です。システムメモリーのプロファイルに Valgrind を使用する方法については、「Valgrind を使用したアプリケーションのメモリー使用量のプロファイリング」 を参照してください。

Valgrind に関する詳細は、Red Hat Enterprise Linux 7 開発者ガイド を参照してください。

Valgrind の使い方については man ページをご覧ください。

```
$ man valgrind
```

valgrind パッケージのインストール時には、付属のドキュメントは **/usr/share/doc/valgrind-*version*** にもあります。

2.14. PQOS

intel-cmt-cat パッケージから利用可能な **pqos** ユーティリティーを使用すると、最近の 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できます。これは、ワークロードの分離や、マルチテナントデプロイメントでのパフォーマンス決定の向上に使用することができます。

リソースディレクターテクノロジー (RDT) 機能セットから以下のプロセッサー機能を公開します。

モニタリング

- Cache Monitoring Technology (CMT) を使用した Last Level Cache (LLC) の使用状況と競合の監視
- メモリー帯域幅監視 (MBM) テクノロジーを使用したスレッドごとのメモリー帯域幅監視

割り当て

- キャッシュアロケーションテクノロジー (CAT) を使用した、特定のスレッドおよびプロセスに使用できる LLC スペースの量の制御

- コードおよびデータ優先度 (CDP) テクノロジーを使用した LLC でのコードおよび配置の制御

以下のコマンドを使用して、システムでサポートされる RDT 機能をリストし、現在の RDT 設定を表示します。

```
# pqos --show --verbose
```

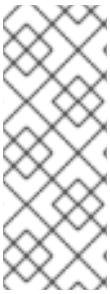
関連情報

- **pqos** の使用に関する詳細は、pqos(8) の man ページを参照してください。
- CMT、MBM、CAT、および CDP プロセッサー機能の詳細は、Intel の公式ドキュメントインテル® リソース・ディレクター・テクノロジー (インテル® RDT)を参照してください。

第3章 TUNED

3.1. TUNED の概要

Tuned は、**udev** を使用して接続されたデバイスを監視し、選択したプロファイルに従ってシステム設定を動的にチューニングするデーモンです。**Tuned** には、高スループット、低レイテンシー、省電力などの一般的なユースケース向けの定義済みプロファイルが多数含まれています。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。特定のプロファイルによるシステム設定に加えられたすべての変更を元に戻すには、別のプロファイルに切り替えるか、**tuned** サービスを非アクティブ化します。



注記

Red Hat Enterprise Linux 7.2 以降では、**Tuned** を **no-daemon** モード で実行できますが、常駐メモリーは必要ありません。このモードでは、**tuned** が設定を適用して終了します。この モードでは、**D-Bus** サポート、ホットプラグサポート、または設定のロールバックサポートなど、多くの **tuned** 機能がないため、**no-daemon** モードはデフォルトで無効になっています。**no-daemon** モード を有効にするには、**/etc/tuned/tuned-main.conf** ファイルで **daemon = 0** を設定します。

静的 チューニングは、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で設定されます。また、**Tuned** はシステムコンポーネントの使用を監視し、その監視情報に基づいてシステム設定を動的にチューニングします。

動的なチューニングでは、所定のシステムの稼働時間中に各種システムコンポーネントを異なる方法で使用する方法が考慮されます。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、これらのコンポーネントのアクティビティーを監視し、その使用の変化に対応します。

実際の例として、一般的なオフィスワークステーションを考えます。通常、イーサネットのネットワークインターフェイスはほとんど使用されず、ほんの数件の電子メールがたまに送受信されるか、Web ページが数ページロードされる程度だとします。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の監視とチューニングのプラグインがあり、これによりこの低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、通常は電力使用量が低くなります。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電子メールが開いている場合など、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すると、**tuned** はこれを検出し、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 およびハードディスクの他のプラグインにも使用されます。

動的チューニングは Red Hat Enterprise Linux ではグローバルに無効になり、**/etc/tuned/tuned-main.conf** ファイルを編集し、**dynamic_tuning** フラグを **1** に変更することで有効にできます。

3.1.1. プラグイン

Tuned は、*監視プラグインと チューニング プラグインの2 種類のプラグイン* を使用します。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。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。

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によって使用できます。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るため、パフォーマンスプロファイルで無効になります(**tuned** プロファイルで個々のプラグインの動的チューニングを有効または無効にできます)。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。

各チューニングプラグインは、個別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る複数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。提供されたプロファイルはワイルドカードを使用して個々のサブシステムのすべてのデバイスを照合します（これを変更する方法の詳細については、「カスタムプロファイル」を参照してください）。これにより、プラグインは必要な目標（選択されたプロファイル）に従ってこれらのサブシステムをチューニングできます。ユーザーが必要とする唯一のことは、正しい **tuned** プロファイルを選択することです。

現時点では、以下のチューニングプラグインが実装されています (動的チューニングはこれらの一部のプラグインのみで実装され、プラグインでサポートされたパラメーターもリストされます)。

cpu

CPU ガバナーを ***governor*** パラメーターで指定された値に設定し、CPU 負荷に応じて PM QoS CPU DMA レイテンシーを動的に変更します。CPU 負荷が ***load_threshold*** パラメーターで指定された値よりも低い場合、レイテンシーは ***latency_high*** パラメーターで指定された値に設定されます。それ以外の場合は、***latency_low*** で指定され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。これは、***force_latency*** パラメーターを必要なレイテンシー値に設定することで実行できます。

eeepc_she

CPU 負荷に応じて FSB 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 としても知られています。CPU 負荷が ***load_threshold_powersave*** パラメーターで指定された値以下の場合、プラグインは ***she_powersave*** パラメーターで指定された値に FSB の速度を設定します(FSB 周波数と対応する値の詳細は、カーネルのドキュメントを参照してください。ほとんどのユーザーに対して機能するはずです)。CPU 負荷が ***load_threshold_normal*** パラメーターで指定された値以上である場合、FSB 速度は ***she_normal*** パラメーターで指定された値に設定されます。静的チューニングはサポートされず、プラグインは透過的に無効になります (この機能のハードウェアサポートが検出されない場合)

net

wake-on-lan を ***wake_on_lan*** パラメーターで指定された値に設定します(**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

プラグインパラメーターで指定されたさまざまな **sysctl** 設定を設定します。構文は **name=value** で

す。**name** は **sysctl** ツールが提供する名前と同じです。このプラグインは、他のプラグインで指定できない設定を変更する必要がある場合に使用します (ただし、設定を特定のプラグインで指定できる場合は、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

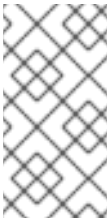
transparent_hugepages パラメーターのブール値に応じて、Transparent Huge Page を有効または無効にします。

audio

オーディオコーデックの autosuspend タイムアウトを、***timeout*** パラメーターで指定された値に設定します。現在、**snd_hda_intel** および **snd_ac97_codec** がサポートされています。値 **0** は、autosuspend が無効になっていることを意味します。また、ブール値パラメーター ***reset_controller*** を **true** に設定することで、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 を ***elevator*** パラメーターで指定された値に設定します。また、ALPM を ***alpm*** パラメーターで指定された値に設定し、ASPM を ***aspm*** パラメーターで指定された値に設定し、 スケジューラークォンタムを ***scheduler_quantum*** パラメーターで指定された値に、ディスクスピンダウンタイムアウトを ***spindown*** パラメーターで指定された値に、disk ***readahead*** を readahead パラメーターで指定された値に乗算できます。また、現在のディスクの readahead 値には ***readahead_multiply*** パラメーターで指定された定数を乗算できます。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と spindown タイムアウト設定が動的に変更されます。動的チューニングは、ブール値パラメーター ***dynamic*** で制御でき、デフォルトで有効になっています。



注記

別のディスクの readahead 値を指定する **tuned** プロファイルを適用すると、ディスク先読み値の設定が上書きされます(**udev** ルールを使用して設定されている場合)。Red Hat は、**tuned** ツールを使用してディスク先読み値を調整することを推奨します。

mounts

disable_barriers パラメーター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

このプラグインは、プロファイルがロードまたはアンロードされたときに実行する外部スクリプトの実行に使用されます。このスクリプトは、起動 または 停止 できる 1 つの引数によって呼び出されます（プロファイルの読み込み時にスクリプトが呼び出されるか、アンロードされるかによって異なります）。スクリプトファイル名は、***script*** パラメーターで指定できます。スクリプトで停止アクションを正しく実装し、変更したすべての設定を起動アクション中に元に戻す必要があることに注意してください。便宜上、関数 Bash ヘルパースクリプトはデフォルトでインストールされ、そで定義されているさまざまな関数をインポートして使用できます。この機能は、主に後方互換性を維持するために提供されます。この機能は最終手段として使用し、必要な設定を指定できる他のプラグインが存在する場合は、そのプラグインを使用することが推奨されます。

sysfs

プラグインパラメーターで指定されたさまざまな **sysfs** 設定を設定します。構文は **name=value** です。**name** は使用する **sysfs** パスです。このプラグインは、他のプラグインで指定できない設定を変更する必要がある場合に使用します (設定を特定のプラグインで指定できる場合は、そのプラグインを使用することが推奨されます)。

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 (現時点では、Radeon カードのみがサポートされます)。省電力レベルは、***radeon_powersave*** パラメーターを使用して指定できます。サポートされる値は、デフォルト の、**auto**、**low**、**mid**、**high**、および **dynpm** です。詳細については、http://www.x.org/wiki/RadeonFeature#KMS_Power_Management_Options を参照してください。このプラグインは試験目的で提供され、今後のリリースで変更されることがあることに注意してください。

bootloader

カーネルブートコマンドラインにパラメーターを追加します。このプラグインは、レガシー GRUB 1 および GRUB 2 をサポートし、さらに Extensible Firmware Interface (EFI) を使用する GRUB をサポートします。grub2 ***_cfg_*** file オプションで、grub2 設定ファイルのカスタマイズされた標準以外の場所を指定できます。これらのパラメーターは、現在の grub 設定とそのテンプレートに追加されます。カーネルパラメーターを有効にするには、マシンを再起動する必要があります。

パラメーターは次の構文で指定できます。

> *cmdline=arg*$_1$ *arg*$_2$ *... arg*$_n$.

3.1.2. インストールと使用方法

tuned パッケージをインストールするには、root として次のコマンドを実行します。

```
yum install tuned
```

tuned パッケージのインストールでは、お使いのシステムに最適なプロファイルも事前に設定されます。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。

throughput-performance

これは、コンピューターノードとして動作する Red Hat Enterprise Linux 7 オペレーティングシステムで事前に選択されます。このようなシステムの目的は、スループットパフォーマンスの最大化です。

virtual-guest

これは、仮想マシンで事前に選択されます。この目的はパフォーマンスの最大化です。最適なパフォーマンスに興味がない場合は、**balanced** プロファイルまたは **powersave** プロファイルに変更することをお勧めします（を参照）。

balanced

これは、他のすべてのケースで事前に選択されます。この目的は、パフォーマンスと電力消費の調和です。

tuned を起動するには、root で以下のコマンドを実行します。

```
systemctl start tuned
```

マシンが起動するたびに **tuned** を有効にするには、次のコマンドを入力します。

```
systemctl enable tuned
```

プロファイルや他のプロファイルの選択など、その他の **tuned** コントロールには、以下を使用します。

```
tuned-adm
```

このコマンドでは、**tuned** サービスが実行されている必要があります。

インストールされた利用可能なプロファイルを参照するには、以下のコマンドを実行します。

```
tuned-adm list
```

現在アクティブなプロファイルを参照するには、以下のコマンドを実行します。

```
tuned-adm active
```

プロファイルを選択またはアクティブ化するには、以下のコマンドを実行します。

```
tuned-adm profile profile
```

以下に例を示します。

```
tuned-adm profile powersave
```

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** アプリケーションは、ロード中にそれらをマージしようとします。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。これは自動的に行われ、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。あまり深く考えずにこの機能を使用すると、一部のパラメーターが反対の方向でチューニングされ、生産性が上がらないことがあります。このような状況の例として、**throughput-performance** プロファイルを使用し、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 値に設定することで、高い スループットのためにディスクを設定します。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システムが最適化され、同時に、低消費電力を実現するようシステムがチューニングされます (低消費電力が最優先である場合)。

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

既存のプロファイルを変更したり、インストール中に使用したのと同じロジックを使用したりせずに **tuned** がシステムに最適なプロファイルを推奨できるようにするには、以下のコマンドを実行します。

```
tuned-adm recommend
```

Tuned 自体には、手動で実行する場合に使用できる追加オプションがあります。ただし、このオプションは推奨されず、主にデバッグ目的で提供されています。利用可能なオプションは、以下のコマンドを使用して表示できます。

```
tuned --help
```

3.1.3. カスタムプロファイル

ディストリビューション固有のプロファイルは、**/usr/lib/tuned/** 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** という名前のメイン設定ファイルと、オプションでヘルパースクリプトなどの他のファイルで設定されます。

プロファイルをカスタマイズする必要がある場合は、プロファイルディレクトリーをカスタムプロファイルに使用される **/etc/tuned/** ディレクトリーにコピーします。同じ名前のプロファイルが 2 つある場合は、**/etc/tuned/** に含まれるプロファイルが使用されます。

<!-- ディレクトリーに独自のプロファイルを作成して、特定のパラメーターが調整または上書きされた状態で **/usr/lib/tuned/** に含まれるプロファイルを使用することもできます。

tuned.conf ファイルには複数のセクションが含まれます。**[main]** セクションが 1 つあります。他のセクションはプラグインインスタンス向けの設定です。**[main]** セクションを含む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。

[main] セクションには、次のオプションがあります。

include=*profile*

指定したプロファイルが含まれます。たとえば、**include=powersave** は **powersave** プロファイルが含まれます。

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 は、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* は、チューニングプラグインのタイプです。チューニングプラグインのリストと説明については、「プラグイン」 を参照してください。*DEVICES* は、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。**devices** 行には、リスト、ワイルドカード(*)、および否定(!)を含めることができます。また、ルールを組み合わせることもできます**devices** 行がない場合は、**TYPE** のシステムに存在する、または後で接続されたすべてのデバイスがプラグインインスタンスによって処理されます。これは、devices = * を使用する場合と同じです。プラグインのインスタンスが指定されない場合、プラグインは有効になりませんプラグインがさらに多くのオプションをサポートする場合は、プラグインセクションでそれらのプラグインを指定することもできます。オプションが指定されないと、デフォルト値が使用されます (インクルードされたプラグインで以前に指定されていない場合)。プラグインオプションのリストについては、「プラグイン」 を参照してください。

例3.1 プラグインインスタンスの記述

以下の例では、**sda** や **sdb** などの **sd** で始まるすべてのものに一致し、それらに対するバリアは無効になりません。

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

以下の例では、**sda1** および **sda2** 以外はすべて一致します。

```
[data_disk]
```

```
type=disk
devices=!sda1, !sda2
disable_barriers=false
```

プラグインインスタンスのカスタム名を必要とせず、設定ファイルでインスタンスの定義が 1 つしかない場合、Tuned は以下の短い構文をサポートします。

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

この場合は、**type** 行を省略できます。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。

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

if the same section is specified more than multiple using the **include** option, then the settings are merge.競合のため、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。場合によっては、以前に定義された内容がわからないことがあります。このような場合は、**replace** ブール値オプションを使用して、これを **true** に設定できます。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は、プラグインを無効にすると便利です。

以下は、**balanced** プロファイルをベースとし、すべてのデバイスの ALPM が最大省電力に設定されるように拡張されたカスタムプロファイルの例です。

```
[main]
include=balanced

[disk]
alpm=min_power
```

以下に、**isolcpus=2** をカーネルブート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
```

変更を反映するには、プロファイルの適用後にマシンを再起動する必要があります。

3.1.4. Tuned-adm

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。Red Hat Enterprise Linux 7 には、一般的なユースケース向けの定義済みプロファイルが多数含まれています。これらは、**tuned-adm** ユーティリティーを使用して簡単にアクティベートできます。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。

利用可能な全プロファイルをリスト表示して、現在アクティブなプロファイルを特定するには、以下を実行します。

```
tuned-adm list
```

現在アクティブなプロファイルのみを表示するには、以下を実行します。

```
tuned-adm active
```

別のプロファイルに切り替える場合は、以下を実行します。

```
tuned-adm profile profile_name
```

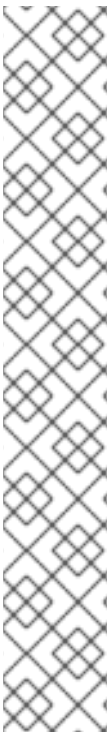
例を示します。

```
tuned-adm profile latency-performance
```

すべてのチューニングを無効にする場合は、以下を実行します。

```
tuned-adm off
```

一般的なユースケースの事前定義済みのプロファイルのリストを以下に示します。



備考

以下のプロファイルは、使用されているリポジトリーファイルに応じて、ベースパッケージでインストールされる場合とされない場合があります。システムにインストールされている **tuned** プロファイルを表示するには、root で以下のコマンドを実行します。

```
tuned-adm list
```

インストールする **tuned** プロファイルの一覧を表示するには、root で以下のコマンドを実行します。

```
yum search tuned-profiles
```

tuned プロファイルをシステムにインストールするには、root で以下のコマンドを実行します。

```
yum install tuned-profiles-profile-name
```

profile-name は、インストールするプロファイルに置き換えます。

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ようとします。ほとんどの負荷で良い結果をもたらします。唯一の欠点はレイテンシーが増加することです。現在の **tuned** リリースでは、CPU、ディスク、音声、およびビデオプラグインが有効になり、**conservative** ガバナーが有効になります。***radeon_powersave*** は **auto** に設定されます。

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。現在の **tuned** リリースでは、USB 自動サスペンド、WiFi 省電力、および SATA ホストアダプターの ALPM 省電力が有効になります。また、ウェイクアップ率が低いシステ

ムのマルチコアの省電力をスケジュールし、**ondemand** ガバナーを有効にします。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 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。

備考

省電力 プロファイルが必ずしも最も効率的であるとは限りません。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、マシンが少ない電力を消費することがあります。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、**balanced** プロファイルが一般的に適切なオプションになります。

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。省電力メカニズムが無効になり、sysctl 設定を有効にして、ディスク、ネットワーク IO、および **deadline** スケジューラーへの切り替えを行います。CPU ガバナーは、**performance** に設定されます。

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)。

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いています。また、透過的な巨大ページと NUMA 調整が無効になり、複数の他のネットワーク関連の sysctl パラメーターがチューニングされます。

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いています。また、カーネルネットワークバッファーが増加されます。

virtual-guest

エンタープライズストレージプロファイルに基づいた、Red Hat Enterprise Linux 7 仮想マシンおよび VMware ゲスト向けに設計されたプロファイルで、他のタスクの中で、仮想メモリーのスワップを減らしディスクの readahead 値を増やします。ディスクバリアは無効になりません。

virtual-host

enterprise-storage プロファイルに基づく仮想ホスト向けプロファイル。仮想メモリーの swappiness を減らし、ディスクの readahead 値を増やし、ダーティーページのより積極的な値を可能にします。

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、一部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。これは、Red Hat Enterprise Linux 6.8 以降で利用可能です。

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。

cpu-partitioning

cpu-partitioning プロファイルは、システム CPU を分離されたハウスキーピング CPU に分割します。分離された CPU のジッターと割り込みを減らすために、プロファイルは分離された CPU を、ユーザー空間プロセス、可動カーネルスレッド、割り込みハンドラー、およびカーネルタイマーから削除します。

ハウスキーピング CPU は、すべてのサービス、シェルプロセス、およびカーネルスレッドを実行できます。

/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。設定オプションは以下のようになります。

isolated_cores=*cpu-list*

分離する CPU をリスト表示します。分離された CPU のリストはコンマで区切るか、ユーザーが範囲を指定できます。3-5 などのダッシュ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。

no_balance_cores=*cpu-list*

システム全体のプロセスの負荷分散時に、カーネルに考慮されない CPU のリストを表示します。このオプションは任意です。通常、これは **isolated_cores** と同じリストです。

cpu-partitioning の詳細は、tuned-profiles-cpu-partitioning(7) の man ページを参照してください。

注記

さらに製品固有なプロファイルまたはサードパーティー製の Tuned プロファイルが利用可能なことがあります。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。

追加の定義済みプロファイルは、**Optional** チャンネルで利用可能な tuned-profiles-compat パッケージでインストールできます。これらのプロファイルは、後方互換性を維持するために提供され、開発は行われていません。基本パッケージの一般的なプロファイルを使用すると、ほとんどの場合、同等のことやそれ以外のことも行えます。特別な理由がない限り、基本パッケージのプロファイルを使用することが推奨されます。互換プロファイルは以下のとおりです。

default

これは、利用可能なプロファイルの省電力への影響が最も低く、調整 された の CPU プラグインおよびディスクプラグインのみを有効にします。

desktop-powersave

デスクトップシステム向けの節電プロファイル。SATA ホストアダプターの ALPM 省電力と、**tuned** の CPU、イーサネット、およびディスクプラグインを有効にします。

laptop-ac-powersave

AC 電源で稼働するノートパソコン向け省電力プロファイル (影響度は中)。SATA ホストアダプターの ALPM 省電力、WiFi 省電力、および **tuned** の CPU、イーサネット、およびディスクプラグインを有効にします。

laptop-battery-powersave

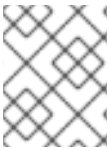
バッテリーで稼働するラップトップ向け省電力プロファイル (影響度は大)。現在の **tuned** 実装では、**powersave** プロファイルのエイリアスです。

spindown-disk

スピンダウン時間を最大化する、従来の HDD が搭載されたマシン向け省電力プロファイル。**tuned** 省電力メカニズムを無効にし、USB 自動サスペンドを無効にし、Bluetooth を無効にし、Wi-Fi 省電力を有効にし、ログの同期を無効にし、ディスクライトバック時間を増やし、ディスクの swappiness を減らします。すべてのパーティションは、**noatime** オプションで再マウントされます。

enterprise-storage

I/O スループットを最大化する、エンタープライズ級のストレージ向けサーバープロファイル。**throughput-performance** プロファイルと同じ設定をアクティベートし、readahead 設定を乗算し、root 以外のパーティションおよびブートパーティション以外のパーティションでバリアを無効にします。



備考

物理マシンで **atomic-host** プロファイルを使用し、仮想マシンで **atomic-guest** プロファイルを使用します。

Red Hat Enterprise Linux Atomic Host の **tuned** プロファイルを有効にするには、tuned-profiles-atomic パッケージをインストールします。root で次のコマンドを実行します。

```
yum install tuned-profiles-atomic
```

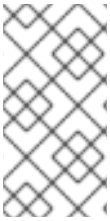
Red Hat Enterprise Linux Atomic Host 用に チューニング された 2 つのプロファイルは次のとおりです。

atomic-host

Red Hat Enterprise Linux Atomic Host 向けに最適化されたプロファイル。ベアメタルサーバーでホストシステムとして使用する場合は、throughput-performance プロファイルを使用します。さらに、SELinux AVC キャッシュと PID 制限が増加し、netfilter 接続追跡がチューニングされます。

atomic-guest

Red Hat Enterprise Linux Atomic Host 向けに最適化されたプロファイル (virtual-guest プロファイルに基づいてゲストシステムとして使用される場合)。さらに、SELinux AVC キャッシュと PID 制限が増加し、netfilter 接続追跡がチューニングされます。



備考

製品固有またはサードパーティーの **tuned** プロファイルが利用可能です。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。カーネルコマンドラインの編集を可能にする **tuned** プロファイル 3 つ(**realtime**、**realtime-virtual-host**、および **realtime-virtual-guest**)が利用できます。

realtime プロファイルを有効にするには、tuned-profiles-realtime パッケージをインストールします。root で次のコマンドを実行します。

```
yum install tuned-profiles-realtime
```

realtime-virtual-host プロファイルおよび **realtime-virtual-guest** プロファイルを有効にするには、tuned-profiles-nfv パッケージをインストールします。root で次のコマンドを実行します。

```
yum install tuned-profiles-nfv
```

3.1.5. powertop2tuned

powertop2tuned ユーティリティーは、**PowerTOP** の提案からカスタムの **tuned** プロファイルを作成できるツールです。

powertop2tuned アプリケーションをインストールするには、root で以下のコマンドを実行します。

```
yum install tuned-utils
```

カスタムプロファイルを作成するには、以下のコマンド root で実行します。

```
powertop2tuned new_profile_name
```

デフォルトでは、現在選択されている **tuned** プロファイルをベースに **/etc/tuned** ディレクトリーにプロファイルが作成されます。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。これを有効にするには、**/etc/tuned/*プロファイル*/tuned.conf** で、対象のチューニングの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** が提案するほとんどのチューニングで新しいプロファイルを生成できます。USB 自動サスペンドなどの一部の危険なチューニングは引き続き無効になります。これらのチューニングが必要な場合は、手動でコメント解除する必要があります。デフォルトでは、新しいプロファイルはアクティブ化されていません。有効にするには、以下のコマンドを実行します。

```
tuned-adm profile new_profile_name
```

powertop2tuned がサポートするオプションの完全なリストについては、以下のコマンドを入力します。

```
powertop2tuned --help
```

3.2. TUNED と TUNED-ADM によるパフォーマンスチューニング

tuned チューニングサービスを使用すると、チューニングプロファイルを設定して、特定のワークロードでより良くパフォーマンスを発揮するようにオペレーティングシステムを調整することができます。**tuned-adm** コマンドラインツールを使用すると、ユーザーは異なるチューニングプロファイルを切り替えることができます。

tuned プロファイルの概要

一般的なユースケースにはいくつかの事前定義済みプロファイルが含まれていますが、**tuned** ではカスタムプロファイルを定義することもできます。このプロファイルは、事前定義済みプロファイルの 1 つをベースにすることも、ゼロから定義することも可能です。Red Hat Enterprise Linux 7 では、デフォルトのプロファイルは **throughput-performance** です。

tuned で提供されるプロファイルは、省電力プロファイルと performance-boosting プロファイルの 2 つのカテゴリーに分類されます。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待ち時間
- ストレージおよびネットワークの高いスループット
- 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

tuned ブートローダープラグイン

tuned Bootloader プラグイン を使用して、パラメーターをカーネル（ブートまたは dracut）コマンドラインに追加できます。GRUB 2 ブートローダーのみがサポートされ、プロファイルの変更を適用するには再起動が必要なことに注意してください。たとえば、***quiet*** パラメーターを **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
```

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、追加のパラメーターが削除されます。システムをシャットダウンまたは再起動すると、カーネルパラメーターは **grub.cfg** ファイルで永続化されます。

環境変数とデプロイメント tuned 組み込み関数

GRUB 2 設定の更新後に **tuned-adm profile *profile_name*** を実行してから **grub2-mkconfig -o *profile_path*** を実行すると、Bash 環境変数を使用できます。これは **grub2-mkconfig** の実行後に展開されます。たとえば、以下の環境変数は **nfsroot=/root** に展開されます。

```
[bootloader]
cmdline="nfsroot=$HOME"
```

環境変数の代わりに **tuned** 変数を使用できます。以下の例では、$ **{isolated_cores}** は 1,2 に 展開 されるため、カーネルは **isolcpus=1,2** パラメーターで起動します。

```
[variables]
isolated_cores=1,2

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

以下の例では、$ **{non_isolated_cores}** は **0,3** -5 に展開され、**cpulist_invert** の組み込み関数は **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 の一覧を反転します。6 CPU マシンの場合、反転は **1**,2 で、カーネルは **isolcpus=1,2** コマンドラインパラメーターで起動します。

tuned 環境変数を使用すると、必要な入力の量が削減されます。調整 された変数とともに、さまざまな

組み込み関数を使用することもできます。組み込み関数がニーズを満たさない場合は、Python でカスタム関数を作成し、プラグインの形式で **tuned** に追加できます。tuned プロファイルがアクティベートされた場合、変数と組み込み関数は実行時にデプロイメントされます。

変数は個別のファイルで指定できます。たとえば、以下の行を **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 を **/etc/tuned/*my-variables.conf*** ファイルに追加すると、カーネルは **isolcpus=1,2** パラメーターで起動します。

デフォルトのシステム tuned プロファイルの変更

デフォルトのシステム **tuned** プロファイルを変更するには、2 つの方法があります。**tuned** プロファイルディレクトリーを新規作成するか、システムプロファイルのディレクトリーをコピーして、必要に応じてプロファイルを編集できます。

手順**3.1** 新しい **tuned** プロファイルディレクトリーの作成

1. **/etc/tuned/** で、作成するプロファイルと同じ新しいディレクトリーを作成します(**/etc/tuned/*my_profile_name*/**)。
2. 新しいディレクトリーで、ファイル **tuned.conf** を作成し、先頭に以下の行を追加します。

   ```
   [main]
   include=profile_name
   ```

3. プロファイルの変更を含めます。たとえば、**vm.swappiness** の値をデフォルトの 10 ではなく 5 に設定して **throughput-performance** プロファイルからの設定を使用するには、以下の行を追加します。

   ```
   [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 my_profile_name
   ```

新しい **tuned.conf** ファイルでディレクトリーを作成すると、システム tuned プロファイルの更新後にプロファイルの変更をすべて保持できます。

または、システムプロファイルを含むディレクトリーを **/user/lib/tuned/** から **/etc/tuned/** にコピーします。以下に例を示します。

```
# cp -r /usr/lib/tuned/throughput-performance /etc/tuned
```

次に、必要に応じて **/etc/tuned** でプロファイルを編集します。同じ名前のプロファイルが 2 つある場合は、**/etc/tuned/** にあるプロファイルが読み込まれることに注意してください。このアプローチの欠点

は、チューニング されたアップグレード後にシステムプロファイルが更新されても、現在古い変更されたバージョンには変更が反映されないことです。

関連情報

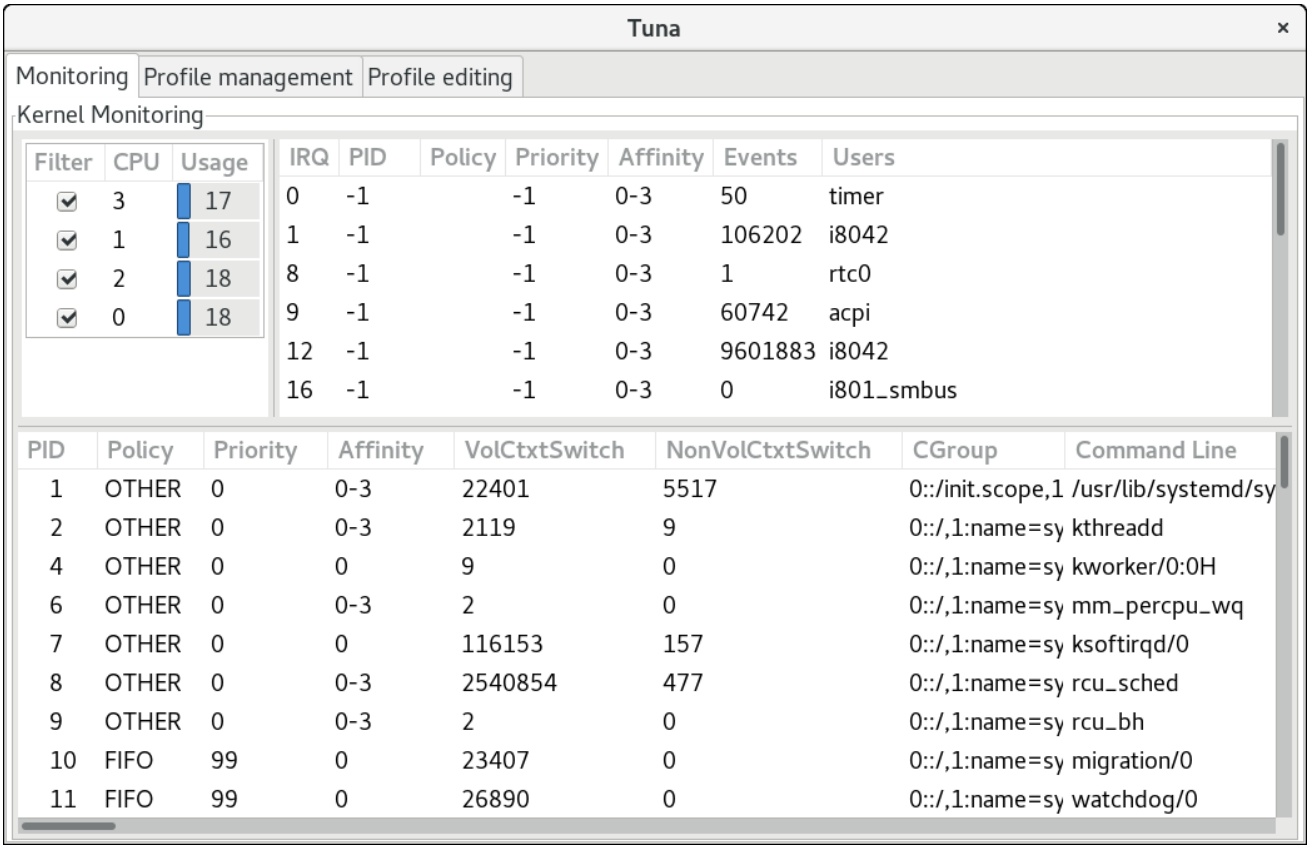
詳細については、「tuned」 および 「tuned-adm」 を参照してください。**tuned** および **tuned-adm** の使用に関する詳細は、tuned(8) および tuned-adm(1) の man ページを参照してください。

第4章 TUNA

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、チューニングスレッドの優先度、IRQ ハンドラー、CPU コアとソケットの分離を行うことができます。Tuna はチューニングタスクの実行の複雑さを解消するのが目的で使用されます。

tuna パッケージをインストールしたら、引数なしで **tuna** コマンドを使用して、Tuna グラフィカルユーザーインターフェイス(GUI)を起動します。**tuna -h** コマンドを使用して、利用可能なコマンドラインインターフェイス(CLI)オプションを表示します。tuna(8) の man ページでは、アクションと修飾子が区別されることに注意してください。

Tuna GUI および CLI は、同等の機能を提供します。GUI は、問題を特定しやすくするために、1 つの画面に CPU トポロジーを表示します。Tuna GUI では、実行中のスレッドを変更することができ、変更の結果を即座に確認することができます。CLI では、Tuna は複数のコマンドラインパラメーターを使用でき、これらのパラメーターを順番に処理します。これらのコマンドは、設定コマンドとしてアプリケーションの初期スクリプトで使用することができます。



Tuna GUI の監視タブ

重要

tuna --save=*filename* コマンドを説明的なファイル名と共に使用して、現在の設定を保存します。このコマンドは、Tuna が変更できるすべてのオプションを保存せず、カーネルスレッドの変更のみを保存します。変更時に実行されていないプロセスは保存されません。

4.1. TUNA を使用したシステムの確認

変更を行う前に、Tuna を使用してシステムの現在の状況を表示することができます。

現在のポリシーと優先度を表示するには、**tuna --show_threads** コマンドを使用します。

-

```
# 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や一致するコマンド名のみを表示するには、**--show_threads** の前に **--threads** オプションを追加します

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

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

現在の割り込み要求(IRQ)およびそれらのアフィニティーを表示するには、**tuna --show_irqs** コマンドを使用します。

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

IRQ 番号に対応する特定の割り込みリクエストや一致する IRQ ユーザー名のみを表示する場合は、**--show_irqs** の前に **--irqs** オプションを追加します。

```
# tuna --irqs=number_or_user_list --show_irqs
```

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

4.2. TUNA を使用した CPU のチューニング

Tuna コマンドは個別の CPU を対象にすることができます。システム上の CPU を一覧表示するには、Tuna GUI の Monitoring タブまたは **/proc/cpuinfo** ファイルを参照してください。

コマンドの影響を受ける CPU のリストを指定するには、以下を使用します。

```
# tuna --cpus=cpu_list --run=COMMAND
```

CPU を分離すると、その CPU 上で現在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 を分離するには、以下を実行します。

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

CPU を含めると、指定の CPU でスレッドを実行することができます。CPU を含めるには、以下を実行します。

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

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**

4.3. TUNA を使用した IRQ のチューニング

システムで現在実行している IRQ の一覧を表示するには、Tuna GUI の Monitoring タブまたは **/proc/interrpupts** ファイルを参照してください。**tuna --show_irqs** コマンドを使用することもできます。

コマンドの影響を受ける IRQ のリストを指定するには、**--irqs** パラメーターを使用します。

```
# tuna --irqs=irq_list --run=COMMAND
```

割り込みを指定の CPU に移動するには、**--move** パラメーターを使用します。

```
# tuna --irqs=irq_list --cpus=cpu_list --move
```

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

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**

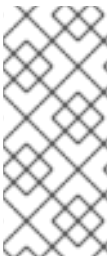
たとえば、名前が **sfc1** で始まるすべての割り込みをターゲットにして、2 つの CPU に分散するには、以下を実行します。

```
# tuna --irqs=sfc1\* --cpus=7,8 --move --spread
```

設定した変更を検証するには、**--move** パラメーターで IRQ を変更する前および後に **--show_irqs** パラメーターを使用します。

```
# tuna --irqs=128 --show_irqs

  # users          affinity
128 iwlwifi        0,1,2,3

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

# tuna --irqs=128 --show_irqs

  # users          affinity
128 iwlwifi              3
```

これで、変更の前後に、選択した IRQ の状態を比較することができます。



注記

Tuna GUI を特定の場合で使用するとより便利です。実行する CPU を指定して IRQ およびスレッドを移動すると、CPU マスクの作成に複数の手順が必要となるため、時間がかかり、困難な可能性があります。Tuna GUI はこの処理を自動化します。Tuna GUI では、スレッドと IRQ を選択して目的の CPU へドラッグすることもできるため、トポロジーの変更が簡単になります。

4.4. TUNA を使用したタスクのチューニング

スレッドのポリシーおよび優先度の情報を変更するには、**--priority** パラメーターを使用します。

```
# tuna --threads=pid_or_cmd_list --priority=[policy:]rt_priority
```

- *pid_or_cmd_list* 引数は、PID またはコマンド名パターンのコンマ区切りリストです。
- *ポリシー* をラウンドロビンの場合は **RR** に設定し、デフォルトポリシーの場合は最初に **FIFO**、またはデフォルトポリシーの場合は **OTHER** に設定します。

  スケジューリングポリシーの概要は、「スケジューリングポリシーの調整」 を参照してください。
- *rt_priority* を 1 から 99 までの範囲で設定します。1 の優先度が最も低くなり、99 の優先度が最も高くなります。

以下に例を示します。

```
# tuna --threads=7861 --priority=RR:40
```

設定した変更を確認するには、**--priority** パラメーターの変更前と後に **--show_threads** パラメーターを使用します。

```
# tuna --threads=sshd --show_threads --priority=RR:40 --show_threads

                thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
 1034   OTHER     0  0,1,2,3        12           17           sshd
                thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
 1034      RR    40  0,1,2,3        12           17           sshd
```

これで、変更の前後に、選択したスレッドの状態を比較することができます。

4.5. TUNA の使用例

例4.1 特定 CPU へのタスク割り当て

以下の例では、4 つ以上のプロセッサーを持つシステムを使用し、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示しています。

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

上記のコマンド例は、以下の操作を順番に実行します。

1. CPU 0 および 1 を選択します。
2. **ssh** で始まるすべてのスレッドを選択します。
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、0 および 1、16 進数マスクで **0x3**、またはバイナリーで **11** として表すことができます。
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選択した 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適切な CPU に設定します。CPU は、2 および 3、16 進数マスクで **0xC**、またはバイナリーでは **1100** として表すことができます。

例4.2 現行設定の表示

以下の例は、**--show_threads** (**-P**) パラメーターを使用して現行の設定を表示し、要求した変更が想定どおりに行われたことをテストします。

```
# 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
```

上記のコマンド例は、以下の操作を順番に実行します。

1. **gnome-sc** 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、ユーザーがアフィニティーマスクと RT の優先度を確認できるようにします。
3. CPU 0 を選択します。
4. **gnome-sc** スレッドを選択した CPU (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 1 にリセットします。
7. **gnome-sc** スレッドを選択した CPU (CPU 1)に移動します。
8. 移動の結果を表示します。
9. CPU 0 を CPU リストに追加します。
10. **gnome-sc** スレッドを選択した 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。

-

第5章 PERFORMANCE CO-PILOT (PCP)

5.1. PCP の概要およびリソース

Red Hat Enterprise Linux 7 は **Performance Co-Pilot** (**PCP**)のサポートを提供します。これは、システムレベルのパフォーマンス測定値を監視、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スイートです。軽量な分散 アーキテクチャーであることから複雑なシステムの一元的な分析に非常に適しています。パフォーマンスメトリックスは、Python、Perl、C++、および C インターフェイスを使用して追加できます。分析ツールではクライアントの API (Python、C++、C) を直接使用でき、リッチな Web アプリケーションでは 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確認することができます。

PCP では以下を行うことができます。

- リアルタイムデータの監視と管理
- 履歴データのログと取得

履歴データを使用して、現在の結果とアーカイブされたデータを比較し、問題のあるパターンを分析することができます。

Performance Metric Collection Daemon (**pmcd**)は、ホストシステムでパフォーマンスデータを収集し、**pminfo** や **pmstat** などのさまざまなクライアントツールを使用して、同じホストまたはネットワークでこのデータを取得、表示、アーカイブ、および処理するために使用できます。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。グラフィカルツールには pcp-gui パッケージが必要です。

PCP で配布されるシステムサービスとツールの一覧については、表A.1「Red Hat Enterprise Linux 7 で Performance Co-Pilot により配布されるシステムサービス」 と 表A.2「Red Hat Enterprise Linux 7 で Performance Co-Pilot により配布されるツール」 を参照してください。

関連情報

- **PCPIntro** という名前の man ページでは、Performance Co-Pilot について紹介しています。利用可能なツールのリスト、利用可能な設定オプションの説明、および関連する man ページのリストが提供されます。デフォルトでは、包括的なドキュメントは **/usr/share/doc/pcp-doc/** ディレクトリー（特に 『Performance Co-Pilot User's and Administrator's Guide』 および 『Performance Co-Pilot Programmer's Guide』 ）にインストールされます。
- PCP については、Red Hat カスタマーポータルのIndex of Performance Co-Pilot (PCP) articles, solutions, tutorials and white papersを参照してください。
- すでに精通している古いツールの機能がある PCP ツールを調べる必要がある場合は、Red Hat ナレッジベースの記事Side-by-side comparison of PCP tools with legacy tools を参照してください。
- Performance Co-Pilot の詳細な説明と使用方法については、DOCUMENTATIONから公式な PCP ドキュメントを参照してください。Red Hat Enterprise Linux ですぐに PCP を使用する場合は、PCP Quick Reference Guideを参照してください。公式な PCP Web サイトには、FAQ (よくある質問のリスト) も含まれます。

5.2. PERFORMANCE CO-PILOT による XFS ファイルパフォーマンスの分析

ここでは、PCP XFS パフォーマンスメトリックとその使用方法について説明します。Performance

Metric Collector Daemon (PMCD) が起動すると、インストールされた Performance Metric Domain Agent (PMDA) からパフォーマンスデータの収集を開始します。PMDA は、システムで個別にロードまたはアンロードでき、同じホスト上の PMCD によって制御されます。PCP の XFS ファイルシステムのパフォーマンメトリックデータを収集するには、デフォルトの PCP インストールの一部である XFS PMDA が使用されます。

PCP で配布されるシステムサービスとツールの一覧については、表A.1「Red Hat Enterprise Linux 7 で Performance Co-Pilot により配布されるシステムサービス」 と 表A.2「Red Hat Enterprise Linux 7 で Performance Co-Pilot により配布されるツール」 を参照してください。

5.2.1. XFS PMDA をインストールして PCP で XFS データを収集

XFS PMDA は pcp パッケージの一部として提供され、インストールではデフォルトで有効になっています。PCP をインストールするには以下を行います。

```
# yum install pcp
```

pcp および pcp-gui パッケージのインストール後にホストマシンで PMDA サービスを有効化および起動するには、以下のコマンドを実行します。

```
# systemctl enable pmcd.service
```

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

PMCD プロセスがホストで実行され、XFS PMDA が設定で有効になっていることを検証するため、PCP 環境にクエリーを行うには、以下を入力します。

```
# pcp

Performance Co-Pilot configuration on workstation:

platform: Linux workstation 3.10.0-123.20.1.el7.x86_64 #1 SMP Thu Jan
29 18:05:33 UTC 2015 x86_64
hardware: 2 cpus, 2 disks, 1 node, 2048MB RAM
timezone: BST-1
services pmcd
pmcd: Version 3.10.6-1, 7 agents
pmda: root pmcd proc xfs linux mmv jbd2
```

XFS PMDA の手動インストール

XFS PMDA が、PCP 設定の読み出し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。PMDA インストールスクリプトによって、PMDA ロールの指定を求められます (collector、monitor、both)。

- **collector** ロールを使用すると、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** ロールを使用すると、システムはローカルシステム、リモートシステム、またはその両方を監視できます。

デフォルトのオプションは collector と **monitor** の両方 であり、ほとんどのシナリオで XFS PMDA が正常に動作できるようにします。

XFS PMDA を手動インストールするには、xfs ディレクトリーに移動します。

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

xfs ディレクトリーで、以下を入力します。

```
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
```

5.2.2. XFS パフォーマンスメトリックの設定および検証

pminfo でのメトリックの検証

PCP がインストールされ、XFS PMDA が有効になっている場合は、「XFS PMDA をインストールして PCP で XFS データを収集」 で手順を利用できます。PCP および XFS で利用可能なパフォーマンスメトリックを確認する最も簡単な方法は、**pminfo** ツールを使用することです。これにより、利用可能なパフォーマンスメトリックに関する情報が表示されます。コマンドは、XFS PMDA によって提供される利用可能なメトリックをすべて表示します。

XFS PMDA によって提供される利用可能なメトリックスをすべて表示するには、以下を実行します。

```
# pminfo xfs
```

以下のオプションを使用して、選択したメトリックの情報を表示します。

-t *metric*

選択したメトリックを説明するヘルプ情報を 1 行で表示します。

-T *metric*

選択したメトリックを説明するより詳細なヘルプテキストを表示します。

-f *metric*

メトリックスに対応するパフォーマンスの現在の読み取り値を表示します。

メトリックのグループまたは個別のメトリックに **-t**、**-T**、および **-f** オプションを使用できます。ほとんどのメトリックデータは、プロービング時にシステム上のマウントされた各 XFS ファイルシステムに提供されます。

ドット(.)を区切り文字として使用し、異なる グループ がルート XFS メトリックの新しいリーフノードになるように配置される XFS メトリックのグループは複数あります。リーフノードセマンティック

(ドット) はすべての PCP メトリックに適用されます。各グループで利用できるメトリックの種類の概要は、表A.3「XFS の PCP メトリックグループ」 を参照してください。

また、XFS のドキュメントには XFS ファイルシステムの監視に関するセクションChapter 13. XFS Monitoringが含まれています。

例5.1 pminfo ツールを使用した XFS 読み書きメトリックの検証

xfs.write_bytes メトリックを説明するヘルプ情報を 1 行表示するには、次のコマンドを実行します。

```
# pminfo -t xfs.write_bytes

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

xfs.read_bytes メトリックを説明するより詳細なヘルプテキストを表示するには、次のコマンドを実行します。

```
# pminfo -T 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 メトリックに対応するパフォーマンスの現在の読み取り値を取得するには、次のコマンドを実行します。

```
# pminfo -f xfs.read_bytes

xfs.read_bytes
    value 4891346238
```

pmstore でのメトリックの設定

PCP では、特にメトリックが制御変数として動作する場合（例： **xfs.control.reset** メトリック）、特定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。

例5.2 pmstore を使用した xfs.control.reset メトリックのリセット

この例は、**xfs.control.reset** メトリックで **pmstore** を使用して、XFS PMDA の記録されたカウンター値をゼロにリセットする方法を示しています。

```
$ pminfo -f xfs.write

xfs.write
    value 325262
```

```
# pmstore xfs.control.reset 1

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

```
$ pminfo -f xfs.write

xfs.write
    value 0
```

5.2.3. ファイルシステムごとに利用できる XFS メトリックの検証

Red Hat Enterprise Linux 7.3 より、PCP は XFS PMDA を有効にし、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。各グループのファイルシステムごとに使用できるメトリックの種類の概要については、表A.4「デバイスごとの XFS の PCP メトリックグループ」 を参照してください。

例5.3 pminfo でのデバイスごとの XFS メトリックの取得

pminfo コマンドは、マウントされた各 XFS ファイルシステムのインスタンス値を提供するデバイスごとの XFS メトリックを提供します。

```
# pminfo -f -t 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
```

5.2.4. pmlogger でのパフォーマンスデータのロギング

PCP を使用すると、後で再生できるパフォーマンスメトリック値をログに記録することができ、回顧的なパフォーマンス分析に使用できます。**pmlogger** ツールを使用して、システムで選択したメトリックのアーカイブログを作成します。

pmlogger を使用すると、システム上で記録するメトリックと記録する頻度を指定することができます。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。

pmlogger を使用してローカルマシンのメトリック値をログに記録するには、プライマリーロギングインスタンスを開始します。

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

```
# systemctl enable pmlogger.service
```

pmlogger が有効で、デフォルトの設定ファイルが設定されている場合は、pmlogger 行が PCP 設定に含まれます。

```
# pcp

Performance Co-Pilot configuration on workstation:
```

```
platform: Linux workstation 3.10.0-123.20.1.el7.x86_64 #1 SMP Thu Jan
[...]
pmlogger: primary logger:/var/log/pcp/pmlogger/workstation/20160820.10.15
```

pmlogconf での pmlogger 設定ファイルの編集

pmlogger サービスの実行中に、PCP はホスト上のデフォルトのメトリクス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、必要に応じて XFS ロギンググループを有効にできます。有効にする重要な XFS グループには、**XFS information**、**XFS data**、および **log I/O traffic** グループが含まれます。

pmlogconf のプロンプトに従って、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すると、有効な各グループのロギング間隔を制御します。グループを選択するには、プロンプトに応答して **y** (yes)または **n** (no)を押します。pmlogconf で汎用 PCP アーカイブのロガー設定を作成または編集するには、以下を入力します。

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

pmlogger 設定ファイルの手動編集

pmlogger 設定ファイルを手動で編集し、指定した間隔で特定のメトリックを追加して、ロギング設定を作成できます。

例5.4 pmlogger 設定ファイルと XFS メトリック

以下の例は、特定の XFS メトリックが追加された **pmlogger config.default** ファイルの抜粋を示しています。

```
# 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;
```

PCP ログアーカイブの再生

メトリックデータの記録後、以下の方法でシステムの PCP ログアーカイブを再生できます。

- **pmdumptext**、**pmrep**、**pmlogsummary** などの PCP ユーティリティーを使用して、ログをテキストファイルにエクスポートし、スプレッドシートにインポートすることができます。

- PCP Charts アプリケーションのデータを再生し、システムの現在のデータとともに、グラフを使用して回顧的なデータを可視化します。「PCP Charts での視覚追跡」を参照してください。

pmdumptext ツールを使用してログファイルを表示できます。pmdumptext を使うと、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できます。pmdumptext ツールを使用すると、アーカイブログ全体をダンプすることができ、コマンドラインでメトリックを指定してログから選択したメトリックの値のみをダンプすることもできます。

例5.5 特定の **XFS** メトリックログ情報の表示

たとえば、5 秒間隔でアーカイブに収集された **xfs.perdev.log** メトリックのデータを表示し、すべてのヘッダーを表示するには、次のコマンドを実行します。

```
$ pmdumptext -t 5seconds -H -a 20170605 xfs.perdev.log.writes

Time local::xfs.perdev.log.writes["/dev/mapper/fedora-home"]
local::xfs.perdev.log.writes["/dev/mapper/fedora-root"]
? 0.000 0.000
none count / second count / second
Mon Jun 5 12:28:45 ? ?
Mon Jun 5 12:28:50 0.000 0.000
Mon Jun 5 12:28:55 0.200 0.200
Mon Jun 5 12:29:00 6.800 1.000
```

詳細は、pcp-doc パッケージから利用できる pmdumptext(1) man ページを参照してください。

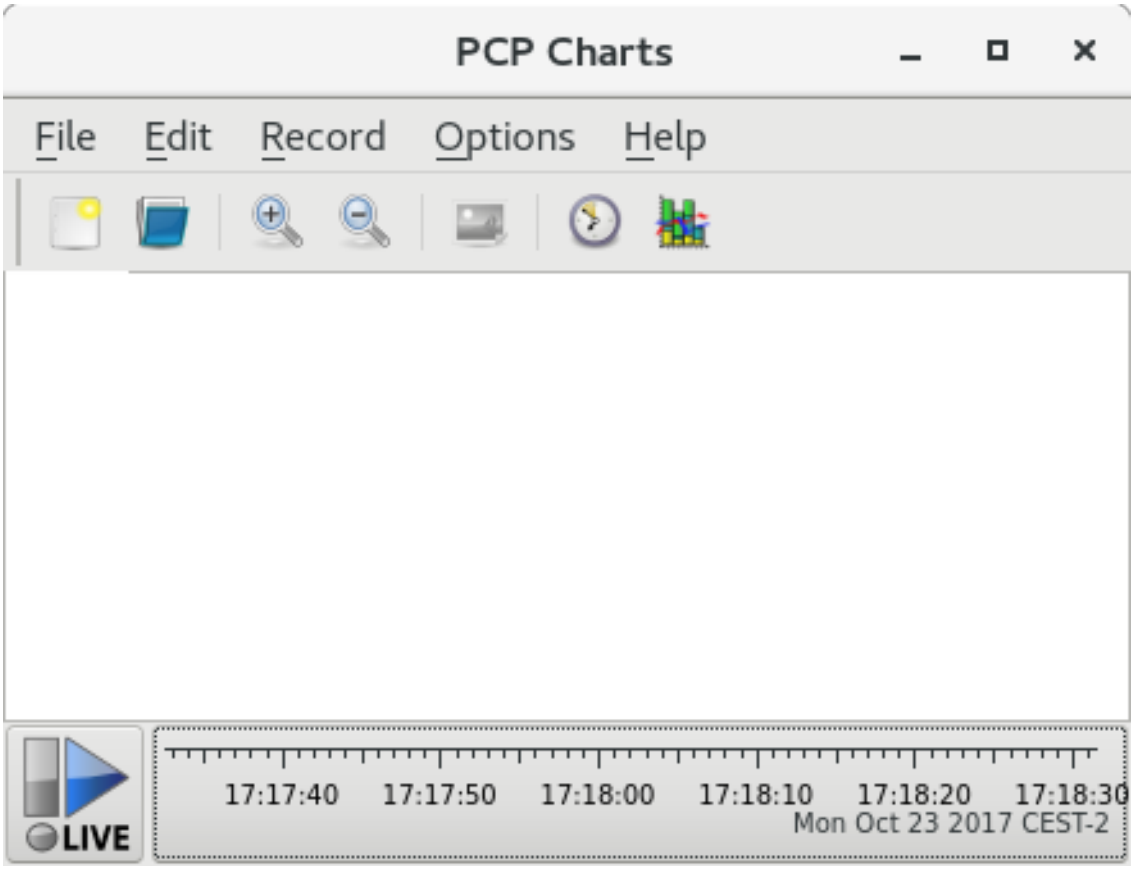
5.2.5. PCP Charts での視覚追跡

グラフィカルな PCP Charts アプリケーションを使用するには、pcp-gui パッケージをインストールします。

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

PCP Charts アプリケーションを使用してパフォーマンスメトリック値をグラフにすることができます。PCP Charts アプリケーションを使用すると、複数のチャートを同時に表示することができます。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。コマンドラインから PCP Charts を起動するには、**pmchart** コマンドを使用します。

PCP Charts の起動後、GUI が表示されます。



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

File → New Chart に移動し、ホスト名またはアドレスを指定して、ローカルマシンとリモートマシンの両方からメトリックを選択します。その後、リモートホストからパフォーマンス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。

PCP Charts で作成されたビューを記録したり、イメージを取得するオプションは複数あります。

- **File → 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 → Start** をクリックして記録を開始します。**Record → Stop** をクリックして、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。

PCP Charts インターフェイスをカスタマイズすると、以下を含む複数の方法でパフォーマンスメトリックからデータを表示できます。

- 折れ線グラフ
- 棒グラフ
- 使用状況グラフ

PCP Charts では、ビュー と呼ばれるメイン設定ファイルにより、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。カスタム ビュー 設定を作成し、**File → Save View** をクリックして保存し、後で **view** 設定を読み込むことができます。ビュー 設定ファイルとその構文の詳細は、pmchart(1) の man ページを参照してください。

例5.6 PCP Charts の view 設定での積み上げチャートグラフ

PCP Charts ビュー設定ファイルの例では、指定の XFS ファイルシステム **loop1** に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"
```

5.3. ファイルシステムデータを収集するための最小限の PCP 設定

以下の手順は、Red Hat Enterprise Linux で統計を収集するために最低限の PCP 設定をインストールする方法を表しています。最低限の設定を行うには、今後の分析用にデータを収集するために必要な最低限のパッケージを実稼働システムに追加します。

作成される **pmlogger** 出力の **tar.gz** アーカイブは、PCP Charts などのさまざまな PCP ツールを使用して分析でき、他のパフォーマンス情報ソースと比較できます。

1. pcp パッケージをインストールします。

```
# yum install pcp
```

2. **pmcd** サービスを起動します。

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

3. **pmlogconf** ユーティリティーを実行して **pmlogger** 設定を更新し、XFS 情報、XFS データ、およびログ I/O トラフィックグループを有効にします。

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

4. **pmlogger** サービスを開始します。

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

5. XFS ファイルシステム上で操作を実行します。

6. **pmlogger** サービスを停止します。

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

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

7. 出力を収集し、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)
```

第6章 CPU

本章では Red Hat Enterprise Linux 7 でアプリケーションのパフォーマンスに影響を与える CPU ハードウェアおよび設定オプションについて簡単に説明します。「留意事項」 ではパフォーマンスに影響を与える CPU 関連の要因について説明します。「パフォーマンスの問題の監視と診断」 では CPU ハードウェアや設定内容に関連するパフォーマンスの問題を分析する Red Hat Enterprise Linux 7 ツールについて説明しています。「推奨設定」 では Red Hat Enterprise Linux 7 で CPU に関するパフォーマンスの問題を解決する際に利用できるツールやストラテジーについて説明しています。

6.1. 留意事項

このセクションを読んで、システムとアプリケーションのパフォーマンスが次の要因によってどのように影響を受けるかを理解してください。

- プロセッサーがどのように相互に、およびメモリーなどの関連リソースに接続されているか。
- プロセッサーによる実行用スレッドのスケジュール方式
- Red Hat Enterprise Linux 7 でのプロセッサーによる割り込み処理の方法

6.1.1. システムのトポロジー

最近のシステムの多くはプロセッサーを複数搭載しているため、**central** processing unit (CPU - 中央処理装置) という考えが誤解を招く恐れがあります。このような複数のプロセッサー同士がどのように接続されているか、また他のリソースとはどのように接続されているか (システムの トポロジー) が、システムやアプリケーションのパフォーマンスおよびシステムチューニング時の留意事項に大きな影響を及ぼす可能性があります。

最近のコンピューティングで使用されているトポロジーには主に 2 種類のタイプがあります。

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。各ソケットにはメモリー専用の領域があり、メモリーへのローカルアクセスがある複数のプロセッサーをまとめてひとつのノードと呼んでいます。

同じノードのプロセッサーはそのノードのメモリーバンクへは高速でアクセスでき、別のノードのメモリーバンクへのアクセスは低速になります。したがってローカルメモリー以外へアクセスする場合にはパフォーマンスが低下します。

この点を考慮するとパフォーマンス重視のアプリケーションは NUMA トポロジーの場合アプリケーションを実行してい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させるようにして、できるだけリモートのメモリーへのアクセスは避けるようにします。

NUMA トポロジーのシステムでアプリケーションのパフォーマンスを調整する場合、アプリケーションが実行される場所そして実行ポイントに最も近いメモリーバンクはどれになるのかを考慮しておくことが重要となります。

NUMA トポロジーのシステムでは、**/sys** ファイルシステムには、プロセッサー、メモリー、および周辺機器の接続方法に関する情報が含まれます。**/sys/devices/system/cpu** ディレクトリーには、システム内のプロセッサーが互いにどのように接続されているかに関する詳細が含まれます。**/sys/devices/system/node** ディレクトリーには、システム内の NUMA ノードに関する情報と、それらのノード間の相対距離に関する情報が含まれます。

6.1.1.1. システムのトポロジーの判断

トポロジーを理解するのに役立つコマンドが数種類あります。**numactl --hardware** コマンドを使用すると、システムのトポロジーの概要を説明します。

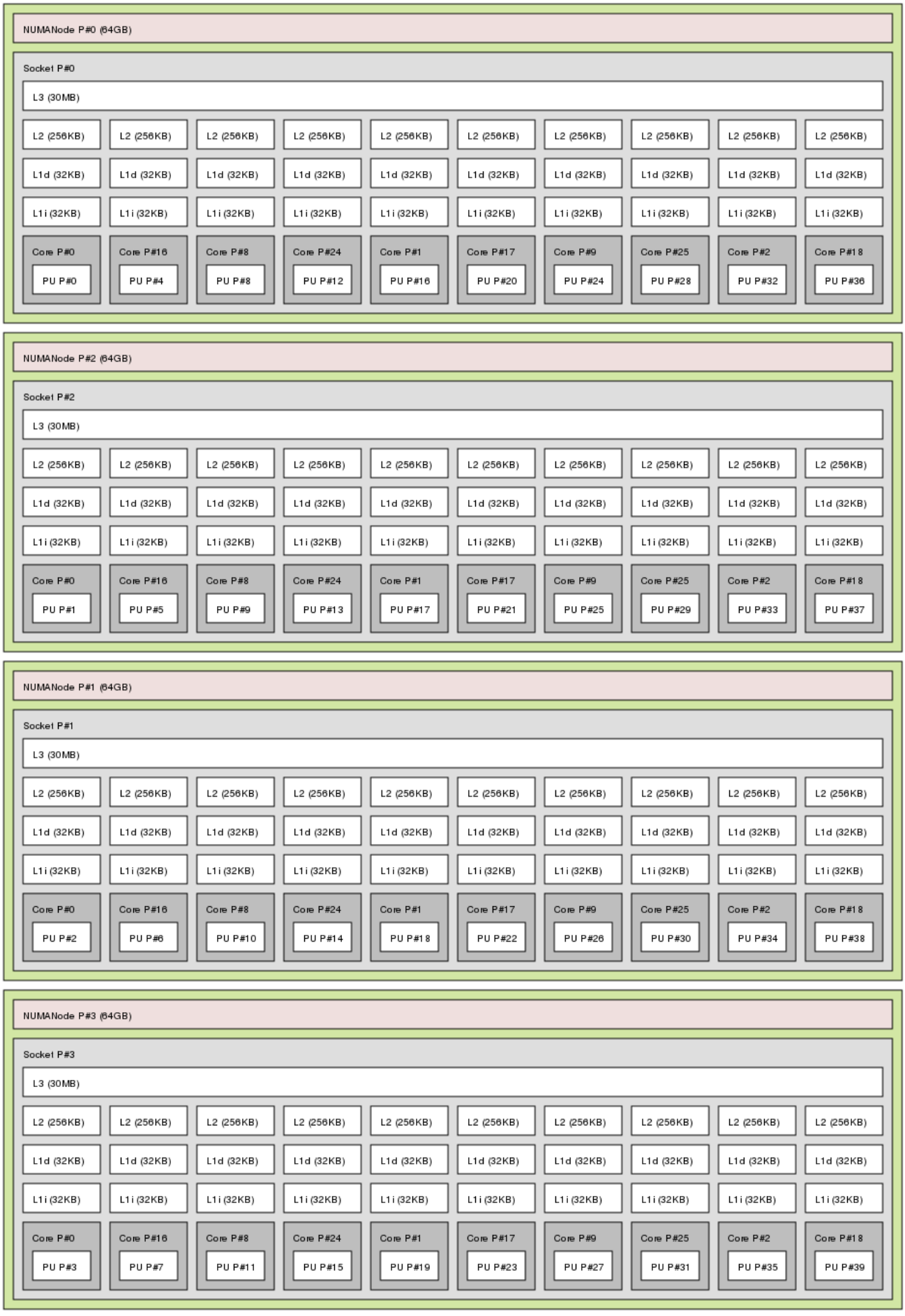
```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
node 1 cpus: 2 6 10 14 18 22 26 30 34 38
node 1 size: 65536 MB
node 1 free: 44321 MB
node 2 cpus: 1 5 9 13 17 21 25 29 33 37
node 2 size: 65536 MB
node 2 free: 44304 MB
node 3 cpus: 3 7 11 15 19 23 27 31 35 39
node 3 size: 65536 MB
node 3 free: 44329 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

lscpu コマンドは util-linux パッケージで提供されます。CPU 数、スレッド数、コア数、ソケット数、NUMA ノード数など、CPU アーキテクチャーに関する情報を収集します。

```
$ 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
```

lstopo コマンドは hwloc パッケージで提供され、システムをグラフィカルに表示します。**lstopo-no-graphics** コマンドは、詳細なテキスト出力を提供します。



lstopo コマンドの出力

6.1.2. スケジューリング

Red Hat Enterprise Linux ではプロセス実行の最小単位を スレッド と呼んでいます。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、システムにとっての最優先事項はシステムを無駄なく稼動させることであるため、アプリケーションのパフォーマンスという観点からは最適なスケジューリングではないかもしれません。

たとえば、NUMA システムで ノード B のプロセッサーが利用可能になったときアプリケーションは ノード A で実行中だったと仮定します。ノード B のプロセッサーを稼働状態に保つため、スケジューラーはアプリケーションのスレッドの 1 つをノード B に移動します。しかし、このアプリケーションスレッドは引き続きノード A のメモリーへのアクセスを必要とします。移動したスレッドはノード B で実行され、ノード A のメモリーはこのスレッドのローカルメモリーではなくなるため、アクセスに時間がかかるようになります。ノード A のプロセッサーが利用できるようになるまで待機する時間や、ローカルメモリーにアクセスして以前のノードでスレッドを実行する時間よりも、このスレッドがノード B で実行を完了する時間の方が長くなる可能性があります。

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。パフォーマンス重視のアプリケーションのニーズに合うよう適切にスレッドのスケジュールを行う方法については、「スケジューリングポリシーの調整」 を参照してください。

6.1.2.1. カーネルティック

Red Hat Enterprise Linux の旧バージョンでは Linux カーネルは完了すべき作業確認のため各 CPU に定期的な割り込みを行っていました。その結果を使用してプロセスのスケジューリングや負荷分散に関する決定を下していました。この定期的な割り込みをカーネル *ティック* と呼んでいました。

コアで行う作業があったかどうかに関わらずティックは発生していました。つまり、アイドル状態のコアであっても割り込み応答を定期的に強制されるため電力消費が高くなっていました (最大 1000 倍/秒)。このためシステムは最近の x86 プロセッサーに搭載されているディープスリープ状態を効率的に利用することができませんでした。

Red Hat Enterprise Linux 6 および 7 ではカーネルはデフォルトでは電力消費が低いアイドル状態の CPU には割り込みをしないようになります。ティックレスカーネルと呼ばれる動作です。実行しているタスクが少ない場合、定期的な割り込みはオンデマンドの割り込みに代わっているためアイドル状態の CPU はその状態またはそれ以下の電力消費状態により長く維持され節電となります。

Red Hat Enterprise Linux 7 では動的なティックレスオプション (**nohz_full**) が用意されユーザー領域タスクによるカーネルの干渉を低減することで決定論がさらに向上されています。このオプションは、***nohz_full*** カーネルパラメーターを使用して、指定したコアで有効にできます。コアでオプションを有効にすると時間管理に関するアクティビティーはすべて待ち時間に制約のないコアに移動されます。ユーザー領域のタスクがカーネルタイマーティック関連の待ち時間にマイクロ秒単位で制約がある高性能な計算やリアルタイムの計算を必要とする作業に便利です。

Red Hat Enterprise Linux 7 で動的なティックレス動作を有効にする方法については、「カーネルティックタイムの設定」 を参照してください。

6.1.3. IRQ (Interrupt Request – 割り込み要求) の処理

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な割り込みを送信できるようひとつまたは複数の IRQ 番号が割り当てられます。割り込みを有効にすると割り込み要求を受け取るプロセッサーは割り込み要求に応答するため現在のアプリケーションスレッドの実行を直ちに中断します。

通常の動作を停止するため、割り込み率が高ければシステムのパフォーマンスが著しく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、または優先度の低い複数の割り込みを 1 つのバッチ (複数の割り込みをまとめる) にして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可能です。

割り込み要求の調整については、「AMD64 および Intel 64 での割り込み親和性の設定」 または「Tuna を使用した CPU、スレッド、割り込みの親和性の設定」 を参照してください。ネットワーク割り込みの詳細については、9章 ネットワーク を参照してください。

6.2. パフォーマンスの問題の監視と診断

Red Hat Enterprise Linux 7 ではシステムパフォーマンスの監視およびプロセッサーやプロセッサーの設定に関連するパフォーマンスの問題の診断を行う際に便利なツールがいくつか用意されています。このセクションではこうしたツールの概要、および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方法を例を使用して説明していきます。

6.2.1. turbostat

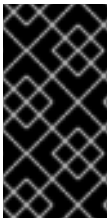
Turbostat は指定された間隔でカウンターの結果を出力し、過剰な電力使用量、ディープスリープ状態に入ることの失敗、システム管理割り込み(SMI)が不必要に作成されるなど、サーバーで予期しない動作を特定するのに役立ちます。

turbostat ツールは、kernel-tools パッケージの一部です。AMD64 および Intel® 64 プロセッサー搭載のシステムでの使用に対応しています。root 権限の実行、不変タイムスタンプカウンター、APERF および MPERF モデル固有のレジスターが必要になります。

使用例については man ページをご覧ください。

```
$ man turbostat
```

6.2.2. numastat



重要

このツールは Red Hat Enterprise Linux 6 ライフサイクルで大幅な更新が行われています。デフォルトの出力は Andi Kleen により開発されたオリジナルツールとの互換性を維持していますが numastat へのオプションやパラメーターを付ける形式についてはその出力から大きく変更されています。

numastat ツールは、プロセスおよびオペレーティングシステムの NUMA ノードごとのメモリー統計を表示し、プロセス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。

numastat 出力をプロセッサーごとの **top** 出力と相互参照し、プロセスメモリーが割り当てられている同じノードでプロセススレッドが実行していることを確認します。

numastat は **numactl** パッケージで提供されます。**numastat** 出力の詳細は man ページをご覧ください。

```
$ man numastat
```

6.2.3. /proc/interrupts

/proc/interrupts ファイルには、特定の I/O デバイスから各プロセッサーに送信された割り込みの数が一覧表示されます。内容は割り込み要求 (IRQ) 数、各プロセッサーで処理されたタイプ別割り込み要求数、送信された割り込みタイプ、記載割り込み要求に応答したデバイスをコンマで区切ったリストなどになります。

特定のアプリケーションまたはデバイスが大量の割り込みを生成しリモートのプロセッサーに処理させようとしている場合はパフォーマンスに影響を及ぼす可能性が高くなります。このような場合はそのアプリケーションまたはデバイスが割り込み要求を処理しているノード同じノードのプロセッサーに処理をさせるようにするとパフォーマンスの低下を軽減することができます。割り込み処理を特定のプロセッサーに割り当てる方法については、「AMD64 および Intel 64 での割り込み親和性の設定」 を参照してください。

6.2.4. pqos でのキャッシュおよびメモリー帯域の監視

intel-cmt-cat パッケージから利用可能な **pqos** ユーティリティーを使用すると、最近の 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できます。

pqos ユーティリティーは、**top** ユーティリティーと同様に、キャッシュとメモリーの監視ツールを提供します。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。

以下のコマンドを使用して監視ツールを開始します。

```
# pqos --mon-top
```

出力のアイテムは LLC 占有の高い順に並べ替えられます。

関連情報

- **pqos** ユーティリティーおよび関連プロセッサー機能の概要は、「pqos」 を参照してください。
- CAT を使用して、DPDK (Data Plane Development Kit) のネットワークパフォーマンスでノイジーネイバー仮想マシンの影響を最小限にする方法については、Intel ホワイトペーパーIncreasing Platform Determinism with Platform Quality of Service for the Data Plane Development Kitを参照してください。

6.3. 推奨設定

Red Hat Enterprise Linux ではシステムの設定に役立つツールをいくつか提供しています。このセクションではこうしたツールを簡単に説明し、Red Hat Enterprise Linux 7 でそのツールを使用してプロセッサー関連の問題を解決する例を紹介します。

6.3.1. カーネルティックタイムの設定

Red Hat Enterprise Linux 7 はデフォルトではティックレスカーネルを使用します。このカーネルは節電のためアイドル状態の CPU には割り込みを行わせないようにして新しいプロセッサーがディープスリープ状態に入れるようにします。

Red Hat Enterprise Linux 7 では動的ティクレスのオプションも用意されています (デフォルトでは無効)。高性能な計算やリアルタイムの計算など待ち時間に厳しい制約のある作業に便利です。

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それらのコアを指定します。16 コアシステムなら **nohz_full=1-15** と指定すると動的ティックレス動作が 1 から 15 までのコアで有効になり時間管理はすべて未指定のコアに移動されます (コア 0)。この動作は、システムの起動時に一時的に有効にするか、**/etc/default/grub** ファイルの **GRUB_CMDLINE_LINUX** オプションを使用して永続的に有効にできます。永続的な動作を得るには、**grub2-mkconfig -o /boot/grub2/grub.cfg** コマンドを実行して設定を保存します。

動的ティックレス動作を有効にする場合は手作業による管理が必要になります。

- システムが起動したら手作業で rcu スレッドを待ち時間に制約のないコアに移動します。この場合コア 0 に移動しています。

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

- 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。

- オプションでカーネルの write-back bdi-flush スレッドの CPU 親和性を housekeeping コアに設定することができます。

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

動的ティックレス設定が正しく動作しているか次のコマンドを実行して確認します。*stress* には 1 秒は CPU で実行しているプログラムを入力します。

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

stress の代替のひとつに、**while :; do d=1; done** などを実行するスクリプトが考えられます。

デフォルトのカーネルタイマー設定は、ビジーな CPU の 1000 ティックを示しています。

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

動的ティックレスカーネルを設定すると、代わりに 1 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 stress -t 1 -c 1
1 irq_vectors:local_timer_entry
```

6.3.2. ハードウェアパフォーマンスポリシーの設定 (x86_energy_perf_policy)

x86_energy_perf_policy ツールを使用すると、管理者はパフォーマンスとエネルギー効率の相対的な重要度を定義できます。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。

デフォルトでは、パフォーマンス モードのすべてのプロセッサーで動作します。**CPUID.06H.ECX.bit3** の有無によって示され、root 権限で実行する必要があります。

x86_energy_perf_policy は、kernel-tools パッケージで提供されます。**x86_energy_perf_policy** の使用方法は、「x86_energy_perf_policy」 または man ページを参照してください。

```
$ man x86_energy_perf_policy
```

6.3.3. taskset でプロセスの親和性を設定

taskset ツールは、util-linux パッケージで提供されます。**taskset** を使用すると、管理者は実行中のプロセスのプロセッサーアフィニティーを取得して設定したり、指定のプロセッサーアフィニティーでプロセスを起動したりできます。

重要

taskset はローカルメモリー割り当てを保証しません。ローカルメモリー割り当てによる追加のパフォーマンス上の利点が必要な場合、Red Hat は **taskset** の代わりに **numactl** を使用することを推奨します。

taskset の詳細は、「taskset」 または man ページを参照してください。

```
$ man taskset
```

6.3.4. numactl で NUMA 親和性を管理

管理者は **numactl** を使用して、指定したスケジューリングまたはメモリー配置ポリシーでプロセスを実行できます。**numactl** は、共有メモリーセグメントまたはファイルに永続ポリシーを設定し、プロセスのプロセッサーアフィニティーとメモリーアフィニティーを設定することもできます。

NUMA トポロジーのシステムではプロセッサーのメモリーへのアクセス速度はプロセッサーとメモリーバンク間の距離が離れるほど低下していきます。したがってパフォーマンス重視のアプリケーションの場合はメモリー割り当てをできるだけ近距離のメモリーバンクから行うよう設定することが重要となります。メモリーと CPU は同じ NUMA ノードのものを使用するのが最適です。

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。つまり、それぞれのスレッドがキャッシュをミスするため、メモリーからデータを取り出すため実行時間を浪費してキャッシュの入れ替えを行います。「perf」 に記載されている **perf** ツールを使用して、過剰な数のキャッシュミスの有無を確認できます。

numactl は プロセッサーとメモリーアフィニティーの管理に役立つ多くのオプションを提供します。詳細については、「numastat」 または man ページをご覧ください。

```
$ man numactl
```

注記

numactl パッケージには **libnuma** ライブラリーが含まれています。このライブラリーは、カーネルがサポートする NUMA ポリシーにシンプルなプログラミングインターフェイスを提供し、**numactl** アプリケーションよりもきめ細かい調整に使用できます。詳細は man ページをご覧ください。

```
$ man numa
```

6.3.5. numad を使用した NUMA 親和性の自動管理

numad は自動 NUMA アフィニティー管理デーモンです。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。

numad は、さまざまなジョブ管理システムでクエリーできるプレプレースメントアドバイスサービスも提供し、それらのプロセスに対する CPU およびメモリーリソースの初期バインディングのサポートを提供します。numad が実行可能ファイルとして実行しているのかサービスとして実行しているのかに関わらずこのプレプレースメントアドバイスを利用することができます。

numad の使い方については、「numad」 または man ページを参照してください。

```
$ man numad
```

6.3.6. スケジューリングポリシーの調整

Linux スケジューラーではスレッドの実行場所と実行期間を決定する数種類のスケジューリングポリシーを実装しています。大きく分けると通常ポリシーとリアルタイムポリシーの 2 種類のカテゴリーに分けられます。通常スレッドは、通常の優先度のタスクに使用されます。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、ブロックされる、終了する、自発的に停止する、より優先度の高いスレッドが取って代わるなどが起こ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。

6.3.6.1. スケジューリングポリシー

6.3.6.1.1. SCHED_FIFO を使用した静的優先度のスケジューリング

SCHED_FIFO（静的優先度スケジューリングとも呼ばれる）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。このポリシーを使用するとイベントの応答時間を改善し待ち時間を低減させることができるため、長期間は実行しない時間的制約のあるタスク対しての使用が推奨されます。

SCHED_FIFO を使用すると、スケジューラーは **SCHED_FIFO** スレッドの一覧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で最も優先度の高いスレッドを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。この場合、99 は最も高い優先度として処理されます。Red Hat では最初は低い順位で使用を開始し、待ち時間に関する問題が見つかった場合にのみ徐々に優先度を上げていく方法を推奨しています。

警告

リアルタイムスレッドはタイムスライスに依存しないため、Red Hat では優先度 99 の設定は推奨していません。この優先度を使用するとプロセスは移行スレッドや監視スレッドと同じ優先レベルにすることになってしまいます。このスレッドが演算ループに陥りブロックされるとスレッドの実行は不可能になります。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** 帯域幅を制限して、リアルタイムアプリケーションのプログラマーがプロセッサーを独占するリアルタイムタスクを開始しないようにすることができます。

/proc/sys/kernel/sched_rt_period_us

上記のパラメーターはプロセッサー帯域幅の 100% とみなされるべき時間をマイクロ秒で定義します。デフォルト値は **1000000** DHEs (1 秒)です。

/proc/sys/kernel/sched_rt_runtime_us

上記のパラメーターはリアルタイムスレッドの実行に当てられる時間をマイクロ秒で定義します。デフォルト値は **950000** DHEs または 0.95 秒です。

6.3.6.1.2. SCHED_RR を使用したラウンドロビン方式の優先度スケジューリング

SCHED_RR は、**SCHED_FIFO** のラウンドロビン版です。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** スレッドの一覧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で最も優先度の高いスレッドをスケジュールします。ただし、**SCHED_FIFO** とは異なり、同じ優先度を持つスレッドは、特定のタイムスライス内でラウンドロビン方式にスケジュールされます。

このタイムスライスの値は、***sched_rr_timeslice_ms*** カーネルパラメーター (**/proc/sys/kernel/sched_rr_timeslice_ms**)を使用してミリ秒単位で設定できます。最も小さい値は 1 ミリ秒になります。

6.3.6.1.3. SCHED_OTHER を使用した通常のスケジュール

SCHED_OTHER は、Red Hat Enterprise Linux 7 のデフォルトの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。多量のスレッドやデータの処理が重要な場合にはこのポリシーの方が長い期間で見ると効率的なスケジュールになります。

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。管理者側でプロセスの niceness 値を変更することはできますがスケジューラーの動的な優先リストを直接変更することはできません。

プロセスの niceness の変更に関する詳細は、Red Hat Enterprise Linux 7 システム管理者のガイド を参照してください。

6.3.6.2. CPU の分離

isolcpus ブートパラメーターを使用して、スケジューラーから 1 つ以上の CPU を分離できます。これによりスケジューラーがその CPU 上にあるユーザー領域のスレッドをスケジュールしないよう保護します。

CPU を分離させた場合は CPU 親和性のシステムコールか numactl コマンドを使用してその CPU に手動でプロセスを割り当てなければなりません。

3 番目の CPU と 6 番目から 8 番目の CPU を分離させる場合は以下をカーネルのコマンドラインに追加します。

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

Tuna ツールを使用して CPU を分離することもできます。Tuna は、起動時にだけでなく、いつでも CPU を分離できます。ただし、この方法は ***isolcpus*** パラメーターとは大きく異なるため、現在 ***isolcpus*** に関連するパフォーマンスの向上は実現しません。ツールの詳細については、「Tuna を使用した CPU、スレッド、割り込みの親和性の設定」 を参照してください。

6.3.7. AMD64 および Intel 64 での割り込み親和性の設定

割り込み要求には、割り込み要求を処理するプロセッサーを定義する関連するアフィニティープロパティー **smp_affinity** があります。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。

重要

本セクションでは、AMD64 および Intel 64 のアーキテクチャーのみを説明します。割り込み親和性の設定は、他のアーキテクチャーとは大きく異なります。

手順6.1 割り込みの自動分散

- 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。

  irqbalance の設定に関する詳細は、「irqbalance」 を参照してください。

手順6.2 割り込みの手動分散

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。

   Red Hat Enterprise Linux 7.5 より、システムが最適な割り込み親和性を特定のデバイスおよびそれらのドライバーに自動的に設定するようになりました。親和性を手動で設定する必要はありません。これは以下のデバイスを対象とします。

   - **be2iscsi** ドライバーを使用するデバイス
   - NVMe PCI デバイス。

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。

   - 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。

   - そうでない場合、チップセットは常にすべての割り込みを 1 つの静的な 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 APIC (Advanced thumbnailer Interrupt Controller) モードが使用されているかを確認します。

   物理以外のフラットモード(フラット)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。

   ```
   $ journalctl --dmesg | grep APIC
   ```

コマンド出力で、以下を行います。

- 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。

システムが **x2apic** モードを使用している場合は、ブートローダー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。

4. **smp_affinity** マスクを計算します。

smp_affinity 値は、システム内のすべてのプロセッサーを表す 16 進数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** です。つまり、割り込み要求はシステム内のどのプロセッサーでも処理できます。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6.3 マスクの計算

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、CPU 0 および CPU 7 で割り込みを処理するには、**0000000010000001** をバイナリーコードとして使用します。

   表6.1 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に対応するようにするには、**0xffffffff,00000000** を使用します。

5. **smp_affinity** マスクを設定します。

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。

計算されたマスクを関連ファイルに書き込みます。

```
# echo mask > /proc/irq/irq_number/smp_affinity
```

関連情報

- 割り込みステアリングをサポートするシステムでは、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、カーネルからの介入なしに特定のプロセッサーで割り込みを処理する決定がハードウェアレベルで行われます。

  割り込みステアリングについては、9章 *ネットワーク* を参照してください。

6.3.8. Tuna を使用した CPU、スレッド、割り込みの親和性の設定

Tuna は実行中のプロセスを調整するためのツールで、CPU、スレッド、および割り込みアフィニティーを制御できます。また、制御できるエンティティーのタイプごとに多くのアクションも提供します。**Tuna** の詳細は、4章 *Tuna* を参照してください。

第7章 MEMORY

本章では、Red Hat Enterprise Linux 7 のメモリー管理機能について簡単に説明します。「留意事項」ではパフォーマンスに影響を与えるメモリー関連の要因について説明します。「パフォーマンスの問題の監視と診断」 ではメモリーの使用や設定内容に関連するパフォーマンス問題の分析に Red Hat Enterprise Linux 7 のツールを利用する方法について説明しています。「システムメモリー容量の設定」 と 「HugeTLB Huge Page の設定」 では Red Hat Enterprise Linux 7 でメモリーに関するパフォーマンスの問題を解決する際に利用できるツールやストラテジーについて説明しています。

7.1. 留意事項

Red Hat Enterprise Linux 7 はデフォルトでは適度な負荷向けに最適化されています。使用するアプリケーションまたは負荷が大量のメモリーを必要とする場合は、システムが仮想メモリーを管理する方法を変更すると、アプリケーションのパフォーマンスが改善される場合があります。

7.1.1. 大きなページサイズ

物理メモリーは、ページと呼ばれる単位で管理されます。Red Hat Enterprise Linux 7 でサポートされるほとんどの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、さまざまな種類の負荷をサポートする Red Hat Enterprise Linux 7 などの汎用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、4 KB ページの使用時に、数百メガバイト、数十ギガバイトの大容量かつ比較的固定のデータのセットで動作するアプリケーションは、パフォーマンスの問題を抱える可能性があります。このようなデータセットには数十万の 4 KB ページが必要になることがあるため、オペレーティングシステムと CPU でオーバヘッドが発生することがあります。

Red Hat Enterprise Linux 7 では、大きなデータセットを使用するアプリケーションに対して大きなページサイズを使用できます。大きなページサイズを使用すると、このようなアプリケーションのパフォーマンスが向上することがあります。

Red Hat Enterprise Linux 7 では、**HugeTLB** 機能（本書では **static huge pages** とも呼ばれる）と **Transparent Huge Page** 機能の 2 つの異なる Huge Page 機能を利用できます。

7.1.2. TLB (Translation Lookaside Buffer) サイズ

ページテーブルからのアドレスマッピングの読み込みには時間とリソースを要するため、CPU は最近使用したアドレスのキャッシュ (Translation Lookaside Buffer (TLB)) とともに構築されます。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 にない場合 (つまり、TLB が不明) は、物理から仮想へのアドレスマッピングを見つけるためにページテーブルを読み取る必要があります。

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 および Huge Page の機能の両方を使用すると、アプリケーションは 4 KB を超える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。

7.2. パフォーマンスの問題の監視と診断

Red Hat Enterprise Linux 7 ではシステムのパフォーマンス監視やシステムメモリーに関連するパフォーマンスの問題の診断を行う際に便利なツールがいくつか用意されています。このセクションではこうしたツールを簡単に説明し、メモリー関連のパフォーマンス問題を監視、診断する方法を例を使用して説明していきます。

7.2.1. vmstat を使用したメモリー使用量の監視

vmstat は procps-ng パッケージで提供され、システムのプロセス、メモリー、ページング、入出力のブロック、割り込み、および CPU アクティビティーに関するレポートを出力します。最後にマシンを起動した時点または最後の報告を行った時点からのイベント平均を瞬時に報告します。

次のコマンドは各種イベントカウンターおよびメモリー統計の表を表示します。

```
$ vmstat -s
```

vmstat の使い方については、「vmstat」 または man ページを参照してください。

```
$ man vmstat
```

7.2.2. Valgrind を使用したアプリケーションのメモリー使用量のプロファイリング

Valgrind は、ユーザー空間バイナリーへのインストルメンテーションを提供するフレームワークです。プログラムのパフォーマンスのプロファイリングや分析に使用できるツールがいくつか収納されています。このセクションで説明されている **valgrind** ツールは、初期化されていないメモリー使用や不適切なメモリー割り当てや割り当て解除などのメモリーエラーを検出するのに役立ちます。

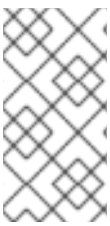
valgrind もしくはそのツールを使用するには、valgrind パッケージをインストールします。

```
# yum install valgrind
```

7.2.2.1. Memcheck を使用したメモリー使用量のプロファイリング

Memcheck 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、これらのエラーのみを報告でき、エラーの発生を防ぐことはできません。通常、セグメンテーション違反が発生するような形でプログラムによるメモリーアクセスが行われればセグメンテーション違反は発生します。ただし、**memcheck** は障害の直前にエラーメッセージをログに記録します。

memcheck はインストルメンテーションを使用するため、**memcheck** で実行されるアプリケーションは、通常よりも 10 倍から 30 倍遅くなります。

アプリケーションで **memcheck** を実行するには、以下のコマンドを実行します。

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

以下のオプションを使用して、特定の問題タイプで **memcheck** 出力に集中することもできます。

--leak-check

アプリケーションの実行が終了すると、**memcheck** はメモリーリークを検索します。デフォルト値は **--leak-check=summary** で、検出されたメモリーリークの数を出力します。**--leak-check=yes** または **--leak-check=full** を指定して、個別のリークの詳細を出力できます。無効にするには、**--leak-check=no** を指定します。

--undef-value-errors

デフォルト値は **--undef-value-errors=yes** で、未定義の値が使用されている場合にエラーを報告します。**--undef-value-errors=no** を指定することもできます。これにより、このレポートが無効になり、Memcheck が若干高速になります。

--ignore-ranges

メモリーアドレス可能性を確認する際に **memcheck** が無視する 1 つ以上の範囲を指定します（例：**--ignore-ranges=0xPP-0xQQ,0xRR-0xSS** ）。

memcheck オプションの全一覧は、**/usr/share/doc/valgrind-version/valgrind_manual.pdf** に含まれているドキュメントを参照してください。

7.2.2.2. Cachegrind を使用したキャッシュ使用量のプロファイリング

Cachegrind 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。シュミレーションされた第一レベルの指示とデータキャッシュの使用量を追跡し、このレベルのキャッシュで不良なアプリケーションコードのやりとりを検出します。また、メモリーへのアクセスを追跡するため最後のレベルのキャッシュ (第 2 または第 3 レベル) も追跡します。そのため、**cachegrind** で実行されるアプリケーションは、通常よりも 20 倍から 100 倍遅くなります。

Cachegrind は、アプリケーションの実行中に統計を収集し、コンソールに要約を出力します。アプリケーションで **cachegrind** を実行するには、以下のコマンドを実行します。

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

以下のオプションを使用して、特定の問題に **cachegrind** 出力に集中することもできます。

--I1

--I1= size、***associativity***、***line_size***のように、最初のレベルの命令キャッシュの***サイズ***、連想性、行サイズ を指定します。

--D1

--D1= size、***associativity***、***line_size***のように、第 **1** レベルのデータキャッシュの***サイズ***、連想性、行サイズ を指定します。

--LL

--LL= size、***associativity***、***line_size***のように、最後のレベルのキャッシュの***サイズ***、連想性、行サイズ を指定します。

--cache-sim

キャッシュアクセスおよびミスの数の収集を有効化または無効化します。これはデフォルトで有効になっています(**--cache-sim=yes**)。このオプションと **--branch-sim** の両方を無効にすると、cachegrind には収集する情報がなくなります。

--branch-sim

ブランチ命令と誤った予測数の収集を有効化または無効化します。これはデフォルトで有効になっています(**--branch-sim=yes**)。このオプションと **--cache-sim** の両方を無効にすると、cachegrind には収集する情報がなくなります。

Cachegrind は、詳細なプロファイリング情報をプロセスごとの **cachegrind.out.*pid*** ファイルに書き込みます。*pid* はプロセス ID です。この詳細情報は、次のようなコンパニオン cg_annotate ツールでさらに処理できます。

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

Cachegrind は cg_diff ツールも提供します。これにより、コードの変更前後にプログラムのパフォーマンスを簡単に確認できます。出力ファイルを比較するには次のコマンドを実行します。first には初期プロファイルの出力ファイル、second には次のプロファイルの出力ファイルを入力します。

```
# cg_diff first second
```

cg_annotate ツールを使用して、出力ファイルをより詳細に表示できます。

cachegrind オプションの全一覧は、**/usr/share/doc/valgrind-version/valgrind_manual.pdf** に含まれているドキュメントを参照してください。

7.2.2.3. Massif を使用したヒープとスタックの領域プロファイリング

massif 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領域を測定します。測定対象は、有用な領域および会計、調整用に割り当てられている追加領域の両方になりますmassif は、アプリケーションのメモリー使用量を減らして、実行速度を高め、アプリケーションがシステムのスワップ領域を使い切る可能性を減らす方法を理解するのに役立ちます。massif で実行されるアプリケーションは、通常よりも約 20 倍遅くなります。

アプリケーションで massif を実行するには、以下のコマンドを実行します。

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

以下のオプションを使用して、特定の問題に massif 出力をフォーカスすることもできます。

--heap

massif がヒープをプロファイリングするかどうかを指定します。デフォルト値は **--heap=yes** です。ヒーププロファイリングを無効にするには、これを **--heap=no** に設定します。

--heap-admin

ヒープのプロファイリングを有効にした場合の管理に使用するブロックごとのバイト数を指定します。デフォルト値は **8** バイトです。

--stacks

massif がスタックをプロファイリングするかどうかを指定します。スタックのプロファイリングでは massif が大幅に遅くなる可能性があるため、デフォルト値は **--stack=no** です。スタックのプロファイリングを有効にするには、このオプションを **--stack=yes** に設定します。massif は、プロファイル対象のアプリケーションに関連するスタックサイズの変更をより適切に示すために、メインスタックがゼロで始まることを前提としています。

--time-unit

massif がプロファイリングデータを収集する間隔を指定します。デフォルト値は **i**（命令が実行される）です。**ms**（ミリ秒またはリアルタイム）および **B**（ヒープおよびスタックで割り当てられたバイトまたは割り当て解除されたバイト）を指定することもできます。ハードウェアが異なる場合でもほぼ再現が可能なため短時間実行のアプリケーションやテスト目的の場合には割り当てられたバイト数を確認すると役に立ちます。

massif はプロファイリングデータを **massif.out.*pid*** ファイルに出力します。ここで、*pid* は指定されたアプリケーションのプロセス識別子です。ms_print ツールは、このプロファイリングデータをグラフ化して、アプリケーションの実行におけるメモリー消費と、メモリー割り当てのピーク時に割り当てを行うサイトに関する詳細情報を表示します。**massif.out.*pid*** ファイルからデータをグラフ化するには、以下のコマンドを実行します。

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

Massif オプションの全一覧は、**/usr/share/doc/valgrind-version/valgrind_manual.pdf** に含まれているドキュメントを参照してください。

7.3. HUGETLB HUGE PAGE の設定

Red Hat Enterprise Linux 7.1 以降、Huge Page を予約する方法は、ブート時とランタイム時にの 2 つの方法があります。起動時に予約すると、メモリーの断片化がそれほど行われていないため、成功の可能性が高くなります。ただし、NUMA マシンでは、複数のページが NUMA ノード間で自動的に分割されます。実行時の方法では、NUMA ノードごとに Huge Page を予約できます。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。

7.3.1. 起動時の Huge Page の設定

起動時に Huge Page を設定するには、カーネルブートコマンドラインに次のパラメーターを追加します。

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。デフォルト値は 0 です。物理的に近接な空きページが十分にある場合にしか Huge Page を割り当てることはできません。このパラメーターで予約されるページは他の目的には使用できません。

この値は起動後に調整するには、**/proc/sys/vm/nr_hugepages** ファイルの値を変更します。

NUMA システムの場合、このパラメーターで割り当てた Huge Page はノード間で平等に分割されます。ノードの **/sys/devices/system/node/*node_id*/hugepages/hugepages-1048576kB/nr_hugepages** ファイルの値を変更することで、ランタイム時に **Huge Page** を特定のノードに割り当てることができます。

詳細は、デフォルトで **/usr/share/doc/kernel-doc-*kernel_version*/Documentation/vm/hugetlbpage.txt** にインストールされている関連カーネルのドキュメントを参照してください。

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。使用できる値は 2 MB と 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。使用できる値は 2 MB と 1 GB です。デフォルト値は 2 MB です。

カーネルブートコマンドラインにパラメーターを追加する方法は、Red Hat Enterprise Linux 7 カーネル管理ガイドの第 3 章カーネルパラメーターおよび値の表示を参照してください。

手順**7.1** 起動初期時に **1 GB** ページを予約

HugeTLB サブシステムでサポートされ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。AMD64 および Intel 64 アーキテクチャーの場合、2 MB の Huge Page と 1 GB の巨大ページがサポートされます。

1. root として **/etc/default/grub** ファイルのカーネルコマンドラインオプションに次の行を追加して、1 GB ページの HugeTLB プールを作成します。

```
default_hugepagesz=1G hugepagesz=1G
```

2. 編集したデフォルトファイルを使用して GRUB2 設定を再生成します。BIOS ファームウェアを使用しているシステムの場合は次のコマンドを実行します。

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

UEFI ファームウェアを使用しているシステムの場合は次のコマンドを実行します。

```
 # grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

3. 以下の内容で **/usr/lib/systemd/system/hugetlb-gigantic-pages.service** という名前のファイルを作成します。

```
[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=/usr/lib/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

4. 以下の内容で **/usr/lib/systemd/hugetlb-reserve-pages.sh** という名前のファイルを作成します。

```
#!/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 number_of_pages node
```

最後の行で、*number_of_pages* を予約する 1GB ページの数に置き換え、 *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。

例7.1 **node0** および **node1** でのページの予約

たとえば、**node0** に 2 つの 1GB ページと **node1** に 1 GB ページを確保するには、最後の行を以下のコードに置き換えます。

```
reserve_pages 2 node0
reserve_pages 1 node1
```

必要に応じて変更したり、行を追加してメモリーを他のノードに予約することができます。

5. スクリプトを実行可能にします。

```
# chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
```

6. 初期のブート予約を有効にします。

```
# systemctl enable hugetlb-gigantic-pages
```

注記

nr_hugepages にいつでも書き込みを行い、ランタイム時に 1 GB 以上のページを予約してみてください。ただし、メモリーの断片化による障害を防ぐために、起動プロセス中に 1GB のページを早期に予約します。

7.3.2. 実行時の Huge Page の設定

次のパラメーターを使用して実行時の Huge Page の動作に影響を与えることができます。

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

指定 NUMA ノードに割り当てる指定サイズの Huge Page 数を定義します。これは Red Hat Enterprise Linux 7.1 からの対応になります。次の例では、2048 kB の Huge Page を **node2** に追加します。

```
# 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
# echo 20 > /sys/devices/system/node/node2/hugepages/hugepages-2048kB/nr_hugepages
# numastat -cm | egrep 'Node|Huge'
              Node 0 Node 1 Node 2 Node 3  Total
AnonHugePages         0      2      0      8     10
HugePages_Total       0      0     40      0     40
HugePages_Free        0      0     40      0     40
HugePages_Surp        0      0      0      0      0
```

/proc/sys/vm/nr_overcommit_hugepages

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 のプールを使い切ってしまった場合にカーネルの通常ページのプールから指定した Huge Page 数が取得されます。この余剰 Huge Page については未使用になると解放されカーネルの通常プールに戻されます。

7.4. TRANSPARENT HUGE PAGE の設定

Transparent Huge Page (THP) は、HugeTLB の代わりとなるソリューションです。THP では、カーネルにより自動的に Huge Page がプロセスに割り当てられるため、Huge Page は手動で予約する必要がありません。

THP 機能には、システム全体とプロセスごとの 2 つの動作モードがあります。THP がシステム全体で有効な場合、カーネルは Huge Page を任意のプロセスに割り当てようとします (Huge Page を割り当てることができ、プロセスが大規模な連続仮想メモリー領域を使用しているとき)。THP がプロセスごとに有効になっている場合、カーネルは、**madvise()** システムコールで指定された個々のプロセスのメモリー領域にのみ Huge Page を割り当てます。

THP 機能では 2-MB のページのみがサポートされることに注意してください。Transparent Huge Page はデフォルトで有効になります。現在のステータスを確認するには、次のコマンドを実行します。

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

Transparent Huge Page を有効にするには、次のコマンドを実行します。

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

アプリケーションが必要とする以上のメモリーリソースを割り当てるのを防ぐには、Huge Page をシステム全体で無効にし、次のコマンドを実行して MADV_HUGEPAGE madvise リージョン内部でのみ Huge Page を有効にします。

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```

Transparent Huge Page を無効にするには、次のコマンドを実行します。

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

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。このような場合、THP を有効にしたままにして 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。直接圧縮を無効にするには、次のコマンドを実行します。

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
```

Transparent Huge Page の包括的な情報については、 **kernel-doc** パッケージのインストール後に利用できる /usr/share/doc/kernel-doc-*kernel_version*/Documentation/vm/transhuge.txt ファイルを参照してください。

7.5. システムメモリー容量の設定

このセクションではメモリーの使用量を改善する場合に役立つメモリー関連のカーネルパラメーターについて説明しています。**/proc** ファイルシステム内の対応するファイルの値を変更することで、テスト目的で一時的に設定することができます。ユースケースに最適なパフォーマンスを生成する値を決定したら、**sysctl** コマンドを使用して永続的に設定することができます。

メモリーの使用量は一般的にはカーネルパラメーター値で設定されます。一時的に設定する場合は /proc ファイルシステム内のファイルの内容を変更し、永続的に変更する場合は、procps-ng パッケージで提供される sysctl ツールを使用して行います。

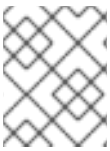
たとえば overcommit_memory パラメーターを一時的に 1 に設定する場合は次のコマンドを実行します。

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

この値を永続的に設定するには、**/etc/ sysctl.conf** に **sysctl vm.overcommit_memory=1** を追加してから、以下のコマンドを実行します。

```
# sysctl -p
```

システムに対するパラメーターの影響を確認する場合には一時的な設定が便利です。パラメーター値で期待する影響を得られたことを確認したら永続的な設定を行います。



注記

専門知識を深めるには、Red Hat Enterprise Linux パフォーマンスチューニング (RH442) トレーニングコースの受講を推奨します。

7.5.1. 仮想メモリーのパラメーター

このセクションにリストされているパラメーターは、特に指定がない限り **/proc/sys/vm** にあります。

dirty_ratio

パーセンテージの値。指定したパーセント値の合計メモリーが変更されるとシステムはその変更を **pdflush** 演算でディスクに記述し始めます。デフォルト値は **20** パーセントです。

dirty_background_ratio

パーセンテージの値。指定したパーセント値の合計メモリーが変更されるとシステムはその変更をバックグラウンドでディスクに記述し始めます。デフォルト値は **10** パーセントです。

overcommit_memory

大量メモリーの要求を許可するか拒否するか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、利用可能なメモリー量と大きすぎる要求の失敗を予測することで、ヒューリスティックなメモリーのオーバーコミット処理を実行します。ただし、メモリーは正確なアルゴリズムではなくヒューリスティックで割り当てられるため、この設定ではメモリーをオーバーロードすることができます。

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実行しません。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 swap 領域の合計および ***overcommit_ratio*** で指定されている物理 RAM の割合と同じか、それ以上のメモリー要求を拒否します。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。

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。

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(**65530**)は、ほとんどの場合に適しています。アプリケーション側でこのファイル数以上の数をマッピングする必要がある場合はこの値を増やします。

min_free_kbytes

システム全体で維持する空領域の最小値をキロバイト単位で指定します。これを使用して各低メモリーゾーンに適切な値が確定され、そのサイズに比例した空き予約ページ数が割り当てられます。

警告

設定値が低すぎるとシステムを破損する恐れがあります。***min_free_kbytes*** を非常に低い値に設定すると、システムがメモリーを回収できなくなります。これにより、システムがハングし、プロセスが OOM で強制終了される可能性があります。ただし、***min_free_kbytes*** の設定が高すぎる（システムメモリーの合計を 5-10% など）に設定すると、システムがメモリー不足の状態に即座に入り、システムにメモリーの回収に時間がかかりすぎます。

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、**oom_killer** 関数は、**oom_score** が最も高いプロセスから開始して、システムが復旧するまでプロセスを強制終了します。

oom_adj パラメーターは、プロセスの ***oom_score*** を決定するのに役立ちます。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。他の有効な値は **-16** から **15** までです。

注記

調整されたプロセスで起動するプロセスは、プロセスの **oom_score** を継承します。

スワップ

swappiness 値(**0** から **100** まで)は、システムが匿名メモリーまたはページキャッシュを優先するレベルを制御します。値が大きい場合は、ファイルシステムのパフォーマンスが改善され、あまり活発ではないプロセスが RAM から積極的にスワップされます。値が小さい場合は、メモリーからのプロセスのスワップが回避されます。通常この場合は、レイテンシーが低下しますが I/O パフォーマンスが犠牲になります。デフォルト値は **60** です。

警告

swappiness==0 を設定すると、スワップが積極的に回避されます。これにより、メモリーおよび I/O の圧力が高まる場合に OOM による強制終了のリスクが高まります。

7.5.2. ファイルシステムのパラメーター

このセクションにリストされているパラメーターは、特に指定がない限り **/proc/sys/fs** にあります。

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** です。この値を変更しても、カーネルデータ構造を事前に割り当てたり、サイズの変更がされないことに留意してください。

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 7 のデフォルト値は、最大 **8192** またはカーネルの起動時に利用可能な空きメモリーページの 10 分の 1 です。

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。

7.5.3. カーネルパラメーター

/proc/sys/kernel/ ディレクトリーにある以下のパラメーターのデフォルト値は、利用可能なシステムリソースに応じて起動時にカーネルによって計算できます。

msgmax

メッセージキュー内の 1 メッセージの最大許容サイズをバイト単位で定義します。この値は、キューのサイズ(***msgmnb***)を超えることはできません。システムの現在の ***msgmax*** 値を確認するには、以下を使用します。

```
# sysctl kernel.msgmax
```

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。システムの現在の ***msgmnb*** 値を確認するには、以下を使用します。

```
# sysctl kernel.msgmnb
```

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。システムの現在の ***msgmni*** 値を確認するには、以下を使用します。

```
# sysctl kernel.msgmni
```

shmall

一度にシステムで使用できる共有メモリーページの合計量を定義します。ちなみに、AMD64 および Intel 64 アーキテクチャーでは 1 ページは 4096 バイトに相当します。

システムの現在の ***shmall*** 値を確認するには、以下を使用します。

```
# sysctl kernel.shmall
```

shmmax

カーネルで許容される 1 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。システムの現在の ***shmmax*** 値を確認するには、以下を使用します。

```
# sysctl kernel.shmmax
```

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。すべてのシステムでは、デフォルト値は **4096** です。

threads-max

システム全体でカーネルが一度に使用できるスレッドの最大数を定義します。システムの現在の ***threads-max*** 値を確認するには、以下を使用します。

```
# sysctl kernel.threads-max
```

デフォルト値は、以下の式で算出されます。

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

最小値は **20** です。

第8章 ストレージとファイルシステム

本章では Red Hat Enterprise Linux 7 で I/O やファイルシステムに関するアプリケーションパフォーマンスに影響を与える対応ファイルシステムおよび設定オプションについて簡単に説明します。「留意事項」 ではパフォーマンスに影響を与える I/O およびファイルシステム関連の要因について説明しています。「パフォーマンスの問題の監視と診断」 では I/O やファイルシステムの設定内容に関連するパフォーマンスの問題を分析する Red Hat Enterprise Linux 7 ツールについて説明しています。「設定ツール」 では Red Hat Enterprise Linux 7 で I/O やファイルシステムに関するパフォーマンスの問題を解決する際に利用できるツールやストラテジーについて説明しています。

8.1. 留意事項

ストレージおよびファイルシステムのパフォーマンスに適した設定はそのストレージの使用用途に大きく依存します。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。

- データの書き込みと読み取りのパターン
- 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

この章を読んで、ファイルシステムのスループット、スケーラビリティ、応答性、リソースの使用、および可用性に影響するフォーマットおよびマウントオプションについて理解してください。

8.1.1. I/O スケジューラー

ストレージデバイスでの I/O の実行時間や実行のタイミングを決定するのが I/O スケジューラーです。I/O エレベーターとも呼ばれています。

Red Hat Enterprise Linux 7 では 3 種類の I/O スケジューラーを用意しています。

deadline

SATA ディスク以外のすべてのブロックデバイス向けのデフォルト I/O スケジューラーです。**deadline** は、要求が I/O スケジューラーに到達する時点からの要求の保証されたレイテンシーを提供しようとします。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。

キュー待ち I/O 要求は、読み取りまたは書き込みのバッチに分けられてから、増大している論理ブロックアドレス順に実行スケジュールに入れられます。アプリケーションは読み取り I/O でブロッ

クする可能性の方が高いため、デフォルトでは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。バッチが処理されると、**deadline** は書き込み操作がプロセッサー時間を使い果たした時間を確認し、必要に応じて次の読み取りまたは書き込みバッチをスケジュールします。1バッチで処理する要求数、書き込みのバッチ1つに対して発行する読み取りのバッチ数、要求が期限切れになるまでの時間などはすべて設定、変更が可能です。詳細は、「deadline スケジューラーのチューニング」をご覧ください。

cfq

SATA ディスクとして識別されるデバイスに限りデフォルトとなるスケジューラーです。Completely Fair Queueing スケジューラー **cfq** は、プロセスをリアルタイム、ベストエフォート、アイドルの 3 つのクラスに分割します。リアルタイムクラスのプロセスは常にベストエフォートのプロセスより先に処理され、ベストエフォートのプロセスはアイドルクラスのプロセスより先に処理されます。つまり、リアルタイムクラスのプロセスは、ベストエフォートおよびアイドルのクラスプロセス時間を奪うことになります。デフォルトでは、プロセスはベストエフォートクラスに割り当てられます。

cfq は履歴データを使用して、アプリケーションが近い将来より多くの I/O 要求を発行するかどうかを予測します。より多くの I/O が予想される場合、**cfq** は、他のプロセスからの I/O が処理を待機している場合でも、新しい I/O を待機するようアイドル状態になります。

cfq スケジューラーはアイドリング状態になりやすいため、意図的な調整を行わない限り、シーク時間を多く要さないハードウェアとは併用しないようにしてください。また、ホストベースのハードウェア RAID コントローラーなど、負荷節約型ではない他のスケジューラーとの併用も避けてください。こうしたスケジューラーを重ねると待ち時間が長くなる傾向があります。

cfq の動作は高度な設定が可能です。詳細は、「CFQ スケジューラーのチューニング」 を参照してください。

noop

noop I/O スケジューラーは、単純な FIFO（先入れ先出し）スケジューリングアルゴリズムを実装します。要求は 単純な last-hit キャッシュを介して汎用のブロック層でマージされます。これは、高速なストレージを使用した CPU バインドシステムに最適な I/O スケジューラーとなります。

異なるデフォルト I/O スケジューラーを設定する方法、特定のデバイスに別のスケジューラーを指定する方法などについては、「設定ツール」 を参照してください。

8.1.2. ファイルシステム

本セクションを読むと Red Hat Enterprise Linux 7 で対応しているファイルシステム、推奨される使用事例、一般的にファイルシステムに対して使用できるフォーマットとマウントオプションなどについて理解することができます。ファイルシステムを調整する際の推奨については、「パフォーマンス改善を目的としたファイルシステムの設定」 をご覧ください。

8.1.2.1. XFS

XFS は堅牢で拡張性の高い 64 ビットファイルシステムです。Red Hat Enterprise Linux 7 ではデフォルトのファイルシステムになります。XFS ではエクステントベースの割り当てと、事前割り当てや遅延割り当てを含む複数の割り当てスキームを使用できます。事前割り当てと遅延割り当てを使用すると、断片化が低減し、パフォーマンスが向上します。また、メタデータジャーナリング機能もサポートされるため、クラッシュから容易にリカバリーできます。XFS はマウントしてアクティブな状態のまま最適化や拡張を行うことができます。Red Hat Enterprise Linux 7 では XFS 固有のバックアップや復元用ユーティリティーに対応しています。

Red Hat Enterprise Linux 7.0 GA からは最大ファイルサイズ 500 TB、ファイルの最大オフセット 8 EB

(sparse ファイル) に対応します。XFS を管理する方法についてはRed Hat Enterprise Linux 7 ストレージ管理ガイドを参照してください。目的別に XFS を調整する方法については、「XFS チューニング」を参照してください。

8.1.2.2. Ext4

ext4 は ext3 ファイルシステムの拡張性を高めたファイルシステムです。デフォルトの動作でほとんどの作業に適しています。ただし、対応しているファイルシステムの最大サイズは 50 TB まで、ファイルサイズは最大 16 TB までになります。ext4 の管理についてはRed Hat Enterprise Linux 7 ストレージ管理ガイドを参照してください。目的別に ext4 を調整する方法については、「ext4 のチューニング」 を参照してください。

8.1.2.3. Btrfs (テクノロジープレビュー)

Red Hat Enterprise Linux 7 のデフォルトのファイルシステムは XFS です。Btrfs (B-tree file system) は比較的新しい copy-on-write (COW) ファイルシステムであり、テクノロジープレビュー として提供されます。一部のユニークな Btrfs の機能は以下のとおりです。

- ファイルシステム全体ではなく特定のファイル、ボリューム、またはサブボリュームのスナップショットを取得する機能。
- 複数バージョンのRedundant Array of Inexpensive Disks (RAID) をサポート
- マップ I/O エラーをファイルシステムオブジェクトに逆参照
- 透過的な圧縮 (パーティション上のすべてのファイルが自動的に圧縮)。
- データとメタデータのチェックサム。

Btrfs は安定的なファイルシステムと見なされますが、開発が続行されているため、成熟したファイルシステムと比べて修復ツールなどの一部の機能は限定的です。

現時点では、高度な機能 (スナップショット、圧縮、ファイルデータチェックサムなど) が必要な場合は、Btrfs を選択することが適切ですが、パフォーマンスは相対的に重要ではありません。高度な機能が必要でない場合は、障害が発生したり、時間とともにパフォーマンスが低下したりするため、他のファイルシステムを使用することが推奨されます。他のファイルシステムと比較した場合の別の欠点は、サポートされる最大ファイルシステムサイズが 50 TB であることです。

詳細については、「Btrfs のチューニング」 と Red Hat Enterprise Linux 7 ストレージ管理ガイド の Btrfs の章を参照してください。

8.1.2.4. GFS2

Global File System 2 (GFS2) は、クラスター化ファイルシステムのサポートを Red Hat Enterprise Linux 7 に提供する High Availability Add-On の一部です。GFS2 は、1 クラスター内の全サーバーで整合性のあるファイルシステムイメージを提供し、サーバーが 1 つの共有ファイルシステムに対する読み取りと書き込みを行うことができるようにします。

GFS2 ファイルシステムは最大 100 TB のサイズまで対応しています。

GFS2 の管理に関する詳細はGlobal File System 2 またはRed Hat Enterprise Linux 7 ストレージ管理ガイドを参照してください。目的別に GFS2 を調整する方法については、「GFS2 のチューニング」 を参照してください。

8.1.3. ファイルシステムの一般的なチューニングで考慮すべき点

本セクションではすべてのファイルシステムに共通した注意点について記載しています。特定のファイルシステムのチューニングに関する推奨事項については、「パフォーマンス改善を目的としたファイルシステムの設定」 をご覧ください。

8.1.3.1. 時刻のフォーマットに関する注意点

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。本セクションではストレージデバイスのフォーマット化を行う前に決定しておかなければならないオプションについて説明します。

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。ファイルシステムのサイズが小さければ比例してバックアップに要する時間も短くなり、ファイルシステムのチェックにかかる時間やメモリーも少なくてすみます。ただし、ファイルシステムが小さ過ぎると深刻な断片化によるパフォーマンスの低下が発生します。

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 (または複数ブロックのサイズ) は一度に読み取られるまたは書き込まれる一般的なデータ量と同じか若干大きい方がファイルシステムのパフォーマンスが向上しデータをより効率的に格納するようになります。ファイルサイズは小さくても 1 ブロック全体が使用されます。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。

デバイスを **mkfs** コマンドでフォーマットする場合、ブロックサイズはファイルシステムのオプションの一部として指定されます。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。詳細は **mkfs** の man ページをご覧ください。たとえば、XFS ファイルシステムをフォーマット化する場合に利用できるオプションを表示させるには次のコマンドを実行します。

```
$ man mkfs.xfs
```

配置

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。デバイスでこれらの推奨事項をエクスポートしない場合、または推奨される設定を変更する場合は、**mkfs** でデバイスをフォーマットする際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。詳細は、ファイルシステムの **mkfs** man ページを参照してください。たとえば、ext4 ファイルシステムのフォーマット時に使用できるオプションを表示させる場合は次のコマンドを実行します。

```
$ man mkfs.ext4
```

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。

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。外部のジャーナルデバイスが失われ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。詳細は **mkfs** および **mount** の man ページを参照してください。

```
$ man mkfs
```

```
$ man mount
```

8.1.3.2. マウント時の注意点

このセクションでは、ほとんどのファイルシステムに適用され、デバイスのマウント時に指定できるチューニングの決定について説明します。

バリア

ファイルシステムバリアによりメタデータが正しく順番通りに永続ストレージに書き込まれ、停電時には **fsync** で送信したデータが必ず維持されるようになります。Red Hat Enterprise Linux の旧バージョンではファイルシステムバリアを有効にすると **fsync** に大きく依存するアプリケーションや小さなファイルを多数作成したり削除したりするアプリケーションなどの速度が著しく低下することがありました。

Red Hat Enterprise Linux 7 ではファイルシステムバリアのパフォーマンスが改善され、ファイルシステムバリアを無効にしても、パフォーマンスに対する影響はごくわずかになります (3 % 未満)。

詳細はRed Hat Enterprise Linux 7 ストレージ管理ガイド を参照してください。

Access Time

ファイルが読み込まれるたびに、そのメタデータはアクセスが発生した時刻(**atime**)で更新されます。この際、追加の書き込み I/O が行われます。ほとんどの場合、Red Hat Enterprise Linux 7 はデフォルトで **atime** フィールドを更新するため、以前のアクセス時間が最後の変更(**mtime**)またはステータス変更(**ctime**)よりも古い場合にのみ atime フィールドが更新されます。

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。

先読み (read-ahead)

先読みは、すぐに必要になる可能性が高いデータを先に取得してページキャッシュにロードすることでディスクからより早くデータを読み出すことが可能になるため、ファイルアクセスの速度が速くなります。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 7 で用意されているストレージ関連の調整済みプロファイルでは LVM ストライプを使用するため先読み値が高く設定されています。しかし、この調整が必ずしもあらゆる作業負荷に十分なわけではありません。

先読み動作を定義するパラメーターはファイルシステムにより異なります。詳細は mount の man ページをご覧ください。

```
$ man mount
```

8.1.3.3. メンテナンス

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。未使用ブロックの破棄方法はバッチ破棄とオンライン破棄の 2 通りの方法があります。

バッチ破棄

このタイプの破棄は **fstrim** コマンドの一部になります。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 7 は、物理的な破棄操作に対応する XFS および ext4 形式のデバイスでのバッチ破棄をサポートします（つまり、**/sys/block/*devname*/queue/discard_max_bytes** の値がゼロではない HDD デバイス、**/sys/block/*devname*/queue/discard_granularity** の値が **0**ではない SSD デバイス）。

オンライン破棄

このタイプの破棄動作はマウント時に **discard** オプションを使用して設定します。ユーザーの介入を必要とせずリアルタイムで実行されます。ただし、オンライン破棄は使用中から空きの状態に移行しているブロックしか破棄しません。Red Hat Enterprise Linux 7 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。

パフォーマンス維持にオンライン破棄が必要な状況やシステムの作業負荷上バッチ破棄が実行できないような状況を除き、バッチ破棄の使用を推奨しています。

事前割り当て

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 7 では、マウント時の XFS デバイス、ext4 デバイス、および GFS2 デバイスでの領域の事前割り当てに対応しています。ファイルシステムに適したパラメーターについては **mount** man ページを参照してください。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。

8.2. パフォーマンスの問題の監視と診断

Red Hat Enterprise Linux 7 ではシステムパフォーマンスの監視、I/O およびファイルシステムやその設定に関連するパフォーマンスの問題の診断を行う際に便利なツールがいくつか用意されています。このセクションではこうしたツールの概要、および I/O およびファイルシステム関連のパフォーマンス問題を監視、診断する方法を例を使用して説明していきます。

8.2.1. vmstat を使用したシステムパフォーマンスの監視

vmstat 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、CPU アクティビティーについて報告を行います。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。

I/O パフォーマンスに最も関連する情報は以下のコラムです。

si

スワップインまたはスワップ領域からの読み取り (KB 単位)。

so

スワップアウトまたはスワップ領域への書き込み (KB 単位)。

bi

ブロックインまたはブロック書き込み操作 (KB 単位)。

bo

ブロックアウトまたはブロック読み取り操作 (KB 単位)。

wa

I/O 操作の完了を待機しているキューの部分。

swap 領域とデータが同じデバイスにある場合、スワップインとスワップアウトはメモリー使用量の目安として特に役立ちます。

また、free、buff、cache の各コラムはライトバック頻度を確認する際に役立ちます。cache の値が突然下がって free の値が増えるような場合、ライトバックとページのキャッシュの無効化が始まったことを指しています。

vmstat による分析で、I/O サブシステムがパフォーマンス低下の原因であることを示している場合、管理者は **iostat** を使用して責任のある I/O デバイスを判断できます。

vmstat は procps-ng パッケージで提供されます。**vmstat** の使い方については man ページをご覧ください。

```
$ man vmstat
```

8.2.2. iostat を使用した I/O パフォーマンスの監視

iostat は sysstat パッケージで提供されます。システム内の I/O デバイスの負荷について報告します。**vmstat** による分析で、I/O サブシステムがパフォーマンス低下を担当していることが示されている場合は、**iostat** を使用して I/O デバイスに関する責任を判断できます。

iostat の man ページで定義されているパラメーターを使用すると、**iostat** レポートの出力にフォーカスできます。

```
$ man iostat
```

8.2.2.1. blktrace を使用した詳細な I/O 分析

blktrace は、I/O サブシステムで費やされた時間に関する詳細情報を提供します。コンパニオンユーティリティー **blkparse** は、**blktrace** から未加工の出力を読み取り、**blktrace** によって記録された入出力操作の人間が判読できる要約を生成します。

このツールの詳細は、blktrace(8) および blkparse(1) の man ページを参照してください。

```
$ man blktrace
```

```
$ man blkparse
```

8.2.2.2. btt を使用した blktrace 出力の分析

btt 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** 出力を分析し、I/O スタックの各領域でデータが費やした時間を表示するため、I/O サブシステムのボトルネックの特定が容易になります。

blktrace メカニズムによって追跡され、**btt** によって分析される重要なイベントの一部は次のとおりです。

- I/O イベントのキューイング(**Q**)
- ドライバーイベント（）への I/Oのディスパッチ
- I/O イベントの完了(**C**)

イベントの組み合わせを検証して、I/O のパフォーマンスに関与する要因を含むまたは除外することができます。

各 I/O デバイスのサブポートのタイミングを検査するには、I/O デバイスのキャプチャーされた **blktrace** イベント間のタイミングを確認します。たとえば、以下のコマンドは、スケジューラー、ドライバー、およびハードウェアレイヤーを含む、カーネル I/O スタックの下位部分に費やされた合計時間(**Q2C**)を待機時間の平均として報告します。

```
$ iostat -x

[...]
Device:         await r_await w_await
vda             16.75    0.97  162.05
dm-0            30.18    1.13  223.45
dm-1             0.14    0.14    0.00
[...]
```

デバイスが要求の処理に長い時間がかかる場合(**D2C**)、デバイスがオーバーロードしているか、デバイスに送信されたワークロードが最適ではない可能性があります。ストレージデバイスにディスパッチされる前にブロック I/O が長時間キューに置かれている場合(**Q2G**)、使用中のストレージが I/O 負荷を提供できないことを示す場合があります。たとえば、LUN のキューが満杯の状態になり、I/O をストレージデバイスにディスパッチできないことがあります。

前後の I/O のタイミングを調べると、ボトルネックの状況が分かります。たとえば、**btt** がブロック層に送信されるリクエスト間の時間(**Q2Q**)が、ブロックレイヤーで費やされた要求の合計時間(**Q2C**)よりも大きいことを示す場合、I/O 要求と I/O サブシステムの間にアイドル時間があり、I/O サブシステムがパフォーマンスの問題を担当しない可能性があることを意味します。

隣接 I/O の **Q2C** 値を比較すると、ストレージサービス時間の変動量を示すことができます。値は以下のいずれかになります。

- 小さい範囲ではほぼ一貫している。
- 分布範囲では変動が大きく、ストレージデバイス側での輻輳問題の可能性がある。

このツールの詳細については、btt(1) の man ページをご覧ください。

```
$ man btt
```

8.2.2.3. iowatcher を使用した blktrace 出力の分析

iowatcher ツールは **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。

このツールの詳細については、iowatcher(1) の man ページをご覧ください。

8.2.3. SystemTap を使用したストレージの監視

Red Hat Enterprise Linux 7 SystemTap ビギナーズガイド には、ストレージパフォーマンスのプロファイリングおよび監視に便利な複数のサンプルスクリプトが含まれています。

以下の **SystemTap** のサンプルスクリプトはストレージパフォーマンスに関連し、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を診断する際に役に立つ場合があります。デフォルトでは、これらは **/usr/share/doc/systemtap-client/examples/io** ディレクトリーにインストールされます。

disktop.stp

ディスクの読み取りや書き込みの状態を 5 秒ごとにチェックして上位 10 位のエントリーを出力します。

iotime.stp

読み取りおよび書き込みの動作に費やした時間、読み取りと書き込みのバイト数を出力します。

traceio.stp

監視した累積 I/O トラフィックに応じた上位 10 位の実行可能ファイルを毎秒出力します。

traceio2.stp

指定デバイスに対して読み取りおよび書き込みが発生するとその実行可能ファイル名とプロセス ID を出力します。

inodewatch.stp

指定のメジャーまたはマイナーデバイス上の指定 inode に対して読み取りや書き込みが発生するたびにその実行可能ファイル名とプロセス ID を出力します。

inodewatch2.stp

指定のメジャーまたはマイナーデバイス上の指定 inode で属性が変更されるたびにその実行可能ファイル名、プロセス ID、属性を出力します。

8.3. ソリッドステートディスク

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。論理ブロックアドレスの全範囲にわたってデータへのアクセス時間は一定になります。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、「メンテナンス」 を参照してください。

デフォルトの I/O スケジューラーと仮想メモリーのオプションは SSD 使用に適しています。

SSD に関する詳細については、『Red Hat Enterprise Linux 7 ストレージ管理ガイド』のソリッドステートディスクの導入ガイドラインの章を参照してください。

SSD のチューニング時に考慮すべき点

SSD のパフォーマンスに影響を及ぼす可能性のある設定を行う場合には、以下の点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 スケジューラーを切り替える方法は、**/usr/share/doc/kernel-*version*/Documentation/block/switching-sched.txt** ファイルを参照してください。

Red Hat Enterprise Linuxnbsp 7.0 では、デフォルトの I/O スケジューラーは Deadline です (SATA ドライブの場合を除く)。SATA ドライブでは、CFQ がデフォルトの I/O スケジューラーです。高速ストレージの場合、Deadline のパフォーマンスは CFQ を上回り、特別なチューニングを実施しなくとも優れた I/O パフォーマンスが得られます。ただし、このデフォルトは一部のディスク (SAS の回転ディスクなど) には適さない場合があります。その場合には、I/O スケジューラーを CFQ に変更します。

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上、**vm_dirty_background_ratio** と **vm_dirty_ratio** の設定を下げようとしてください。書き出しのアクティビティーが増えても、通常はディスク上の他の操作のレイテンシーに悪影響を与えないためです。ただし、このチューニングで 総 **I/O** 処理が増加する場合があるため、ワークロードに固有のテストを実施しない限りこのチューニングは推奨できません。

スワップ

SSD はスワップデバイスとしても使用でき、通常ページアウトおよびページインに優れたパフォーマンスを示します。

8.4. 設定ツール

Red Hat Enterprise Linux ではストレージやファイルシステムの設定を行う際に役立つツールがいくつか用意されています。このセクションではこうしたツールを簡単に説明し、Red Hat Enterprise Linux 7 でそのツールを使用して I/O およびファイルシステム関連の問題を解決する例を紹介します。

8.4.1. ストレージパフォーマンス向けチューニングプロファイルの設定

Tuned サービスは、特定のユースケースのパフォーマンスを向上させるために設計されたプロファイルを複数提供します。次のプロファイルはストレージのパフォーマンス改善に特に役立ちます。

- latency-performance
- throughput-performance (デフォルト)

システムでプロファイルを設定するには次のコマンドを実行します。*name* には使用するプロファイル名を入力します。

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

tuned-adm recommend コマンドは、システムに適切なプロファイルを推奨します。

プロファイルの詳細および設定オプションなどについては、「tuned-adm」 を参照してください。

8.4.2. デフォルト I/O スケジューラーの設定

デフォルトの I/O スケジューラーとは、デバイスで他のスケジューラーを明示的に指定しなかった場合に使用されるスケジューラーです。

デフォルトのスケジューラーが指定されていない場合、**cfq** スケジューラーは SATA ドライブに使用され、**deadline** スケジューラーは他のすべてのドライブに使用されます。このセクションで説明する手順に従ってデフォルトのスケジューラーを指定した場合には、そのデフォルトスケジューラーがすべてのデバイスに適用されます。

デフォルトの I/O スケジューラーを設定するには、**Tuned** ツールを使用するか、**/etc/default/grub** ファイルを手動で変更します。

Red Hat は、**Tuned** ツールを使用して、起動したシステムでデフォルトの I/O スケジューラーを指定することを推奨します。***elevator*** パラメーターを設定するには、**disk** プラグインを有効にします。**disk** プラグインの詳細は、『Tuned』 の章の 「プラグイン」 を参照してください。

GRUB 2 を使用してデフォルトのスケジューラーを変更するには、起動時またはシステムの起動時に、カーネルコマンドラインに ***elevator*** パラメーターを追加します。手順8.1「GRUB 2 を使用したデフォルト I/O スケジューラーの設定」 で説明されているように、**Tuned** ツールを使用するか、手動で **/etc/default/grub** ファイルを変更できます。

手順8.1 GRUB 2 を使用したデフォルト I/O スケジューラーの設定

起動しているシステムのデフォルト I/O スケジューラーを設定し、再起動後も設定を維持するには、以下の手順を実施します。

1. **/etc/default/grub** ファイルの **GRUB_CMDLINE_LINUX** 行に ***elevator*** パラメーターを追加します。

```
# cat /etc/default/grub
...
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=vg00/lvroot rd.lvm.lv=vg00/lvswap
```

```
elevator=noop"
...
```

Red Hat Enterprise Linux 7 では、利用可能なスケジューラーは 期限、**noop**、および **cfq** です。詳細は、kernel-doc パッケージのインストール後に利用可能な、カーネルのドキュメントにある **cfq-iosched.txt** ファイルおよび **deadline-iosched.txt** ファイルを参照してください。

2. ***elevator*** パラメーターを追加して、新しい設定を作成します。

   BIOS ファームウェアを使用しているシステムと UEFI ファームウェアを使用しているシステムとでは、GRUB 2 設定ファイルの場所が異なります。以下のコマンドのいずれかを使用して、GRUB 2 設定ファイルを再作成します。

   - BIOS ファームウェアを使用している場合は、以下のコマンドを使用します。

     ```
     # grub2-mkconfig -o /etc/grub2.cfg
     ```

   - UEFI ファームウェアを使用している場合は、以下のコマンドを使用します。

     ```
     # grub2-mkconfig -o /etc/grub2-efi.cfg
     ```

3. システムを再起動して、変更を有効にします。

   GNU GRand Unified Bootloader バージョン 2 (GRUB 2) の詳細な情報については、『Red Hat Enterprise Linux 7 システム管理者のガイド』のGRUB 2 についてセクションを参照してください。

8.4.3. 汎用のブロックデバイスチューニングパラメーター

このセクション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/sd*X*/queue/** ディレクトリーにあります。リストされたチューニングパラメーターは I/O スケジューラーのチューニングと区別され、すべての I/O スケジューラーに適用できます。

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可能な場合は、このパラメーターを **0** に設定できます。

iostats

デフォルト値は **1** (***有効***)です。***iostats*** 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ます。これにより、I/O パスのオーバーヘッドがわずかに削除されます。***iostat*** 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する可能性があります。特定のストレージモデルにベンダーによって特に指定されていない限り、***iostat*** を有効にしたままにしておくことが推奨されます。

iostats を無効にすると、デバイスの I/O 統計は **/proc/diskstats** ファイルに表示されなくなります。**/sys/diskstats** の内容は、**sar** や ***iostats*** などの I/O ツールを監視するための I/O 情報のソースです。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*** の値によって決定されます。

一部のソリッドステートディスクは、I/O リクエストが内部消去ブロックサイズよりも大きいとパフォーマンスが悪化します。システムにアタッチするソリッドステートディスクモデルがこれに該当するかを判断するには、ハードウェアのベンダーに確認し、ベンダーの推奨事項に従います。Red Hat では、***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ことを推奨します。ストレージデバイスによって指定されていない場合は、いずれかのパラメーターに ***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、マージが有効になります。単純な 1 回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

一度にキュー待ちさせることができる読み取りと書き込み要求の最大数を指定します。デフォルト値は **128** です。つまり、読み取りまたは書き込みを要求する次のプロセスがスリープ状態になる前に、128 個の読み取り要求と 128 書き込み要求をキューに入れることができます。

遅延の影響を受けやすいアプリケーションの場合、ライトバック I/O が書き込み要求でデバイスキューを満杯にできないようにするため、このパラメーターの値を低くしてストレージのコマンドキューの深さを制限します。デバイスのキューが満杯になると I/O 動作を実行しようとしている他のプロセスはキューが使用できるようになるまでスリープ状態になります。要求はラウンドロビン方式で割り当てられ、1 つのプロセスが継続してキューのすべての領域を使用しないようにします。

I/O スケジューラー内の I/O 操作の最大数は **nr_requests*2** です。前述のように、***nr_requests*** は読み取りおよび書き込みに個別に適用されます。***nr_requests*** は I/O スケジューラー内の I/O 操作にのみ適用され、基盤となるデバイスに既にディスパッチされている I/O 操作には適用されないことに注意してください。したがって、デバイスに対する未処理の I/O 操作の上限は **(nr_requests*2)+(queue_depth)** で、***queue_depth*** は **/sys/block/sdN/device/queue_depth** で、LUN キューの深さとも呼ばれます。たとえば、この未処理の I/O 操作の合計数は **avgqu-sz** 列の **iostat** の出力で確認できます。

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、次のシーケンシャル読み取りに対し、必要となりそうな情報はカーネルページキャッシュ内にすでに存在するため、読み取りの I/O 操作が向上します。

多くの場合、デバイスマッパーは ***read_ahead_kb*** の値が高いという利点があります。マップする各デバイスに対して 128 KB の値が適切な開始点になりますが、***read_ahead_kb*** の値を 4-8 MB に増やすと、大きなファイルのシーケンシャル読み取りが行われるアプリケーション環境での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。ソリッドステートデバイスがこれを **0** に自動的に設定しない場合は、スケジューラーで不要な seek-reducing ロジックを無効にするように手動で設定します。

rq_affinity

デフォルトでは I/O 要求を発行したプロセッサーとは異なるプロセッサーで I/O の完了を処理することができます。***rq_affinity*** を **1** に設定してこの機能を無効にし、I/O 要求を発行したプロセッサーでのみ完了を実行します。これによりプロセッサーのデータキャッシングの効率性が改善されます。

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。

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

8.4.4. deadline スケジューラーのチューニング

期限 が使用中の場合、キューに入れられた I/O 要求は読み取りまたは書き込みのバッチに分類され、LBA の増加順に実行がスケジュールされます。アプリケーションは読み取り I/O でブロックする可能性の方が高いため、デフォルトでは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。バッチが処理されると、**deadline** は書き込み操作がプロセッサー時間を使い果たした時間を確認し、必要に応じて次の読み取りまたは書き込みバッチをスケジュールします。

以下のパラメーターは、**deadline** スケジューラーの動作に影響します。

fifo_batch

ひとつのバッチで発行する読み取りまたは書き込みの動作数です。デフォルト値は **16** です。値を高くするほど処理能力も高まりますが待ち時間も増加します。

front_merges

前方マージを生成しない場合は、このチューナブルを **0** に設定できます。ただし、このチェックのオーバーヘッドを測定していない限り、Red Hat はデフォルト値の **1** を推奨します。

read_expire

ミリ秒単位で指定します。この時間内に読み取り要求がスケジュールされます。デフォルト値は **500** (0.5 秒)です。

write_expire

ミリ秒単位で指定します。この時間内に書き込み要求がスケジュールされます。デフォルト値は **5000** (5 秒)です。

writes_starved

読み取りバッチ数を指定します。指定バッチ数を先に処理してから書き込みバッチをひとつ処理します。高い値を設定するほど読み取りバッチの方が多く優先して処理されます。

8.4.5. CFQ スケジューラーのチューニング

CFQ を使用するとプロセスはリアルタイム、ベストエフォート、アイドルの 3 種類いずれかのクラスに配置されます。リアルタイムのプロセスはすべてベストエフォートのプロセスより先にスケジュールされます。ベストエフォートのプロセスはすべてアイドルのプロセスより先にスケジュールされます。デフォルトではプロセスはベストエフォートのクラスに配置されます。**ionice** コマンドを使用すると、プロセスのクラスを手動で調整できます。

次のパラメーターを使用すると CFQ スケジューラーの動作をさらに調整することができます。これらのパラメーターは、**/sys/block/*devname*/queue/iosched** ディレクトリー配下の指定されたファイルを変更することで、デバイスごとに設定されます。

back_seek_max

CFQ に後方シークを行わせる最長距離をキロバイトで指定します。デフォルト値は **16** KB です。後方シークは特にパフォーマンスを低下さるため、大きな値の使用は推奨していません。

back_seek_penalty

ディスクヘッドで前方または後方への移動を決定する際に後方シークに対して適用する乗数を指定します。デフォルト値は **2** です。ディスクヘッドの位置が 1024 KB で、システムに等距離の要求がある場合(1008 KB および 1040 KB など)、***back_seek_penalty*** が後方シーク距離に適用され、ディスクは前方に移動します。

fifo_expire_async

ミリ秒単位の長さで指定します。非同期 (バッファーされた書き込み) の要求を処理せず放置する長さです。この時間を過ぎると非同期の処理待ち要求が処理リストに移動されます。デフォルト値は **250** ミリ秒です。

fifo_expire_sync

同期（読み取りまたは **O_DIRECT** の書き込み）要求が処理されないままになる時間の長さ（ミリ秒単位）。この時間を過ぎると非同期の処理待ち要求が処理リストに移動されます。デフォルト値は **125** ミリ秒です。

group_idle

このパラメーターは、デフォルトで **0**（無効）に設定されます。**1**（有効）に設定すると、**cfq** スケジューラーは、コントロールグループで I/O を発行している最後のプロセスでアイドリングします。これは、比例加重 I/O コントロールグループを使用し、***slice_idle*** が **0**（高速ストレージ）に設定されている場合に役立ちます。

group_isolation

このパラメーターは、デフォルトで **0**（無効）に設定されます。**1**（有効）に設定すると、グループ間のより強力に分離されますが、ランダムなワークロードと連続的なワークロードの両方に適用されるため、スループットが低下します。***group_isolation*** が無効(**0**に設定)の場合、連続したワークロードにのみ公平性が提供されます。詳細は、**/usr/share/doc/kernel-doc-version/Documentation/cgroups/blkio-controller.txt** にインストールされているドキュメントを参照してください。

low_latency

このパラメーターはデフォルトで **1**（有効）に設定されます。有効にすると、**cfq** は、デバイスで I/O を発行する各プロセスに最大 **300** ミリ秒の待機時間を提供することで、スループットの公平性を優先します。このパラメーターを **0**（無効）に設定すると、ターゲットレイテンシーは無視され、各プロセスは完全なタイムスライスを受け取ります。

quantum

このパラメーターは、**cfq** が一度に 1 つのデバイスに送信する I/O 要求の数を定義します。基本的にはキューの深さを制限します。デフォルト値は **8** つの要求です。使用するデバイス側はより深いキューに対応している可能性はありますが、quantum の値を上げると待ち時間も増えます。特に大量の連続する書き込み作業負荷がある場合は顕著です。

slice_async

非同期の I/O 要求を発行している各プロセスに割り当てるタイムスライスの長さを指定します (ミリ秒単位)。デフォルト値は **40** ミリ秒です。

slice_idle

次の要求を待つあいだ cfq にアイドリングを行わせる長さをミリ秒単位で指定します。デフォルト値は **0** （キューまたはサービスツリーレベルではアイドリングなし）です。デフォルト値を使用すると外付け RAID ストレージでの処理能力が最適となりますが、シーク動作の総数が増加するため RAID ではない内蔵ストレージの場合には処理能力が低下します。

slice_sync

同期 I/O 要求を発行している各プロセスに割り当てるタイムスライスの長さをしていします (ミリ秒単位)。デフォルト値は **100** ミリ秒です。

8.4.5.1. 高速ストレージの CFQ のチューニング

高速な外部ストレージアレイやソリッドステートディスクなど、シークのペナルティーが大きいハードウェアでは **cfq** スケジューラーは推奨されません。このストレージで **cfq** を使用する必要がある場合、次の設定ファイルを編集する必要があります。

- **/sys/block/*devname*/queue/iosched/slice_idle** を **0**に設定します。
- **/sys/block/*devname*/queue/iosched/quantum** を **64**に設定します。
- **/sys/block/*devname*/queue/iosched/group_idle** を **1**に設定します。

8.4.6. noop スケジューラーのチューニング

noop I/O スケジューラーは、主に高速ストレージを使用する CPU バウンドシステムに役立ちます。また、一般的に **noop** I/O スケジューラーは、仮想ディスクに対して I/O 操作を実行するときに仮想マシンで使用されますが、排他的には使用されません。

noop I/O スケジューラーに固有の調整可能なパラメーターはありません。

8.4.7. パフォーマンス改善を目的としたファイルシステムの設定

本セクションでは Red Hat Enterprise Linux 7 で対応している各ファイルシステムに固有のパラメーターのチューニングについて説明しています。パラメーターはストレージデバイスのフォーマット時にパラメーターを設定する場合と、フォーマット化したデバイスのマウント時に設定する場合のいずれかに分けられます。

パフォーマンス低下の原因がファイルの断片化またはリソースの競合にある場合、一般的にはファイルシステムの再設定を行うことでパフォーマンスが改善されます。ただし、アプリケーションの変更を要する場合があります。このような場合にはカスタマーサポートに連絡してください。

8.4.7.1. XFS チューニング

本セクションではフォーマット時とマウント時に XFS ファイルシステムに使用できるチューニングパラメーターのいくつかについて説明します。

ほとんどの作業負荷で XFS のデフォルトフォーマットとマウント設定が適しています。この設定の変更は特定の作業負荷に必要な場合に限ってください。

8.4.7.1.1. フォーマットオプション

これらのフォーマットオプションの詳細については、man ページを参照してください。

```
$ man mkfs.xfs
```

ディレクトリーのブロックサイズ

ディレクトリーのブロックサイズは I/O 動作ごとに読み出したり修正したりするディレクトリー情報の量に影響を与えます。ディレクトリーブロックサイズの最小値はファイルシステムのブロックサイズになります (デフォルト 4 KB)。ディレクトリーブロックサイズの最大値は **64** KB です。

所定のブロックサイズの場合、サイズが大きいディレクトリーの方が小さいディレクトリーより多くの I/O を必要とします。またディレクトリーのブロックサイズが大きいシステムの方が小さいサイズより I/O 動作ごとの処理能力の消費量が高くなります。したがってディレクトリーのサイズ、ディレクトリーブロックサイズ共にできるだけ小さいサイズにすることを推奨しています。

Red Hat では 表8.1「ディレクトリーブロックサイズに対して推奨しているディレクトリーエントリーの最大数」 に示すディレクトリーブロックサイズで書き込みが多い作業負荷のエントリー数および読み取りが多い作業負荷のエントリー数を超えない設定を推奨しています。

表8.1 ディレクトリーブロックサイズに対して推奨しているディレクトリーエントリーの最大数

ディレクトリーのブロックサイズ	最大エントリー数 (読み取りが多い作業負荷)	最大エントリー数 (書き込みが多い作業負荷)
4 KB	100,000–200,000	1,000,000–2,000,000
16 KB	100,000–1,000,000	1,000,000–10,000,000
64 KB	>1,000,000	>10,000,000

ディレクトリーのブロックサイズによって異なる読み取りや書き込みの作業に与える影響については XFS 提供のドキュメントを参照してください。

ディレクトリーのブロックサイズを設定するには **mkfs.xfs -l** オプションを使用します。詳細は、man ページの **mkfs.xfs** を参照してください。

割り当てグループ

割り当てグループは独立した構造でファイルシステムのセクション全体に割り当てられている inode や空領域でインデックスを作成します。各割り当てグループは個別に修正できるため XFS による割り当て動作および割り当て解除の動作は影響するグループが異なる限り同時に行わせることが可能です。したがってファイルシステム内で実行可能な並列動作数は割り当てグループ数と同数になります。ただし、並列動作の実行は動作を行えるプロセッサー数によっても制限されるため割り当てグループ数はシステム内のプロセッサー数と同数またはそれ以上にすることを推奨しています。

ひとつのディレクトリーを複数の割り当てグループで同時に変更することはできません。したがって多数のファイルを作成したり削除するアプリケーションには一つのディレクトリーに全てのファイルを保存させないようにすることを推奨しています。

割り当てグループを設定するには **mkfs.xfs -d** オプションを使用します。詳細は、man ページの **mkfs.xfs** を参照してください。

増大に関する制約

フォーマットを行った後にファイルシステムのサイズを大きくする必要が生じた場合 (ハードウェアの追加やシンプロビジョニング)、一旦フォーマットを完了した割り当てグループのサイズは変更できないため初期のファイルレイアウトに十分注意してください。

割り当てグループのサイズ決定は初期のファイルシステム容量ではなく最終的な容量に応じて行ってください。完全に増大し切った状態のファイルシステムの割り当てグループ数はその最大サイズ (1 TB) に達しない限り 200 から 300 以内に収まるようにします。したがって、ほとんどのファイルシステムで増大可能な推奨最大サイズは初期サイズの 10 倍になります。

RAID アレイでファイルシステムを増大させる場合、新しい割り当てグループのヘッダーが追加したストレージに正しく合うようデバイスのサイズを割り当てグループサイズの倍数にする必要があるため更に注意が必要です。また、新しいストレージには既存ストレージと同じ配列を持たせる必要があります。フォーマット後は配列を変更することができないため、同じブロックデバイスに異なる配列のストレージがある場合は最適化が行えません。

inode とインライン属性

inode に十分な領域がある場合は inode への属性名と値の書き込みを XFS で直接行うことができます。こうしたインライン属性の読み出しや変更は余分な I/O が必要ないため別々の属性ブロックの読み出しに比べ 10 倍速くなります。

デフォルトの inode サイズは 256 バイトです。属性の格納に使用できるのはこのうちの約 100 バイトのみ、inode に格納されるデータエクステントポインター数により異なります。ファイルシステムをフォーマットする際に inode サイズを増やすと属性の格納に使用できる領域サイズが増加します。

属性名および属性値はいずれも最大サイズ 254 バイトに制限されています。属性名か属性値のいずれかの長さが 254 バイトを超えるとその属性は別の属性ブロックにプッシュされインラインには格納されなくなります。

inode パラメーターを設定するには **mkfs.xfs -i** オプションを使用します。詳細は、man ページの **mkfs.xfs** を参照してください。

RAID

ソフトウェア RAID を使用している場合は、**mkfs.xfs** が、適切なストライプユニットと幅で、基盤となるハードウェアを自動的に設定します。しかし、ハードウェア RAID を使用している場合はハードウェア RAID が必ずしもすべてこの情報をエクスポートするとは限らないため手作業によるストライプ単位とストライプ幅の設定が必要な場合があります。ストライプユニットと幅を設定するには、**mkfs.xfs -d** オプションを使用します。詳細は、man ページの **mkfs.xfs** を参照してください。

ログサイズ

保留中の変更はログに書き込みが行われる同期イベントの発生までメモリー内に集められます。ログのサイズにより同時に処理できる並列の変更数が決定します。また、メモリー内に集めることができる変更の最大サイズも決定するため、ログ記録したデータがディスクに書き込まれる頻度も決定されます。ログが小さいほどデータのディスクへの書き込み頻度は多くなります。ただし、大きいログはそれだけ保留中の変更を記録するため大きくのメモリーを使用するため、メモリーに制限があるシステムの場合はログを大きくしても意味がありません。

ログはベースとなるストライブの単位と合わせるとパフォーマンスが良くなります。つまり、ログがストライプ単位の境界線で開始や終了を行うためです。ログをストライプユニットに合わせるには **mkfs.xfs -d** オプションを使用します。詳細は、man ページの **mkfs.xfs** を参照してください。

ログサイズを設定するには、**mkfs.xfs** オプションを使用します。*logsize* はログのサイズに置き換えます。

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

詳細は **mkfs.xfs** man ページをご覧ください。

```
$ man mkfs.xfs
```

ログのストライプ単位

RAID5 や RAID6 レイアウトを使用するストレージデバイスに書き込みを行うログの場合、ストライプ単位の境界線で書き込みの開始や終了を行わせるとパフォーマンスが良くなります (ベースとなるストライプ単位にあわせる)。**mkfs.xfs** は適切なログのストライプ単位を自動的に設定しようとしますが、この情報をエクスポートしている RAID デバイスにより異なります。

同期イベントが非常に頻繁に発生するような作業の場合、ログのストライプ単位を大きく設定するとパフォーマンスが低下する場合があります。書き込みが小さい場合、1 ログストライプのサイズを埋める必要があるため待ち時間が増えることがあるためです。ログ書き込みの待ち時間で制約を受ける作業の場合は、Red Hat では、ログのストライプ単位を 1 ブロックに設定して、アラインされていないログの書き込み開始を可能とすることを推奨しています。

対応しているログのストライプ単位の最大サイズはログのバッファーサイズの最大値です (256 KB)。これによりベースとなるストレージにログに設定できるストライプ単位より大きいストライプ単位を持たせることができます。この場合、**mkfs.xfs** は警告を発行し、ログストライプユニットを 32 KB に設定します。

ログのストライプ単位を設定するには以下のいずれかのオプションを使用します。*N* にはストライプ単位として使用するブロック数を入力します。*size* にはストライプ単位のサイズを KB で入力します。

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

詳細は **mkfs.xfs** man ページをご覧ください。

```
$ man mkfs.xfs
```

8.4.7.1.2. マウントオプション

Inode 割り当て

1 TB を超えるサイズのファイルシステムの場合、inode 割り当ての使用を強く推奨します。***inode64*** パラメーターは、ファイルシステム全体で inode とデータを割り当てるように XFS を設定します。これにより inode の多くがファイルシステムの先頭に割り当てられるのを防ぎ、データの多くがファイルシステムの後方に割り当てられるようになり、大きなファイルシステムのパフォーマンスが向上します。

ログのバッファーサイズと数

ログバッファーが大きいほどログにすべての変更を書き込む際に要する I/O 動作数が少なくなります。大きなログバッファーは I/O 使用が多い作業で非揮発性の書き込みキャッシュがないシステムのパフォーマンスを改善します。

ログのバッファーサイズは ***logbsize*** マウントオプションで設定され、ログバッファーに格納できる情報の最大量を定義します。ログのストライプ単位が設定されていない場合、バッファーの書き込みは最大値よりも短くなる可能性があるため、同期の多いワークロードのログバッファーサイズを縮小する必要はありません。ログバッファーのデフォルトサイズは 32 KB です。最大サイズは 256 KB です。これ以外に対応しているサイズは 64 KB、128 KB、または 32 KB から 256 KB のあいだのログストライプ単位の 2 の累乗の倍数です。

ログバッファーの数は、***logbufs*** マウントオプションで定義されます。デフォルト値は 8 ログバッファー (最大) ですが最小では 2 ログバッファーを指定することができます。余分なログバッファーにメモリーを割り当てる余裕のないメモリー制約のあるシステム以外、通常はログバッファー数を減らす必要はありません。ログバッファー数を減らすとログのパフォーマンスが低下する傾向があります。特にログの I/O 待ち時間に制約のある作業に顕著に見られます。

変更のログ記録の遅延

XFS にはログに変更を書き込む前にメモリーにその変更を集めておくことができるオプションがあります。***delaylog*** パラメーターを使用すると、頻繁に変更されるメタデータを、変更するたびにログに定期的に書き込みできます。このオプションを使用するとクラッシュで失う可能性のある動作数が増え、またメタデータの追跡に使用するメモリー量も増加します。ただし、メタデータの変更速度やスケーラビリティーが 10 倍向上されるため、**fsync**、**fdatasync**、または **sync** などを使用してデータとメタデータのディスクへの書き込みを確実に行わせる場合にデータやメタデータの整合性を損うことがありません。

マウントオプションの詳細は、**man xfs**を参照してください。

8.4.7.2. ext4 のチューニング

本セクションではフォーマットおよびマウント行う際に使用できる ext4 ファイルシステムのチューニングパラマーターについて説明します。

8.4.7.2.1. フォーマットオプション

Inode テーブルの初期化

ファイルシステム内のすべての inode を初期化するときに、非常に大きいファイルシステムではかなりの時間がかかる場合があります。デフォルトでは、初期化のプロセスは保留になります (レイジー inode テーブルの初期化が有効)。ただし、システムに ext4 ドライバーがない場合は、レイジー inode テーブルの初期化がデフォルトでは無効になります。**lazy_itable_init** を 1 に設定すると有効にできます。このような場合、カーネルのプロセスはファイルシステムがマウントされるとその初期化を続行します。

本セクションではフォーマット時に利用できる一部のオプションについてのみ説明しています。フォーマットパラメーターの詳細は **mkfs.ext4** の man ページを参照してください。

```
$ man mkfs.ext4
```

8.4.7.2.2. マウントオプション

Inode テーブル初期化の割合

レイジー inode テーブルの初期化が有効になっている場合は、***init_itable*** パラメーターの値を指定することで初期化が発生する速度を制御できます。バックグラウンドでの初期化にかかる時間はほぼ 1 をこのパラメーターの値で割った数値になります。デフォルト値は **10** です。

ファイルの自動同期

既存ファイル名の変更を行ったり、ファイルの短縮や書き直しをすると **fsync** が正しく動作しないアプリケーションがあります。ext4 の場合、デフォルトではこうした動作の後には必ず自動的にファイルの同期が行われます。ただしこのファイルの自動同期は時間がかかる場合があります。

ここまでの同期を必要としない場合はマウント時に **noauto_da_alloc** オプションを指定するとこの動作を無効にすることができます**noauto_da_alloc** を設定する場合、データの整合性を確保するにはアプリケーション側で明示的に fsync を使用する必要があります。

ジャーナル I/O の優先度

デフォルトでは、ジャーナル I/O の優先度は **3** で、これは通常の I/O よりも若干高いです。マウント時に ***journal_ioprio*** パラメーターを使用して、ジャーナル I/O の優先度を制御できます。***journal_ioprio*** の有効な値は **0** から **7** までで、**0** は最も優先度が高い I/O です。

本セクションではマウント時に使用できる一部のオプションのみを説明しています。マウントオプションの詳細については **mount** の man ページをご覧ください。

```
$ man mount
```

8.4.7.3. Btrfs のチューニング

Red Hat Enterprise Linux 7.0 以降、Btrfs がテクノロジープレビューとして提供されています。チューニングは、現在の負荷に基づいてシステムを最適化するために常に行う必要があります。作成およびマウントのオプションについては、Red Hat Enterprise Linux 7 ストレージ管理ガイドのBtrfsの章を参照してください。

データ圧縮

デフォルトの圧縮アルゴリズムは zlib ですが、特定の負荷がある場合は、圧縮アルゴリズムを変更することが推奨されます。たとえば、ファイル I/O が大きいシングルスレッドの場合は、lzo アルゴリズムを使用することが推奨されます。マウント時オプションは次のとおりです。

- **compress=zlib**: 圧縮率が高く、古いカーネルに安全なデフォルトオプション。
- **compress=lzo**: 圧縮は高速ですが、zlib よりも低速です。
- **compress=no**: 圧縮を無効にします。
- **compress-force=*method***: 動画やディスクイメージなどの圧縮率が高くないファイルにも圧縮を有効にします。利用可能な方法は **zlib** および **lzo** です。

マウントオプションの追加後に作成または変更されたファイルのみが圧縮されます。既存のファイルを圧縮するには、以下で *method* を **zlib** または **lzo** に置き換えたあとでこのコマンドを実行します。

```
$ btrfs filesystem defragment -cmethod
```

lzo を使用してファイルを再圧縮するには、次のコマンドを実行します。

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

8.4.7.4. GFS2 のチューニング

本セクションではフォーマットおよびマウント行う際に使用できる GFS2 ファイルシステムのチューニングパラマーターについて説明します。

ディレクトリーの間隔

GFS2 マウントポイントの最上位レベルのディレクトリー内に作成されるディレクトリーはすべて自動的に間隔が置かれ、断片化を低減すると共にディレクトリー内での書き込み速度を速めていま

す。最上位レベルのディレクトリーなどの別のディレクトリーを空けるには、以下のようにそのディレクトリーに ***T*** 属性でマークします。*dirname* は、スペースを空けるディレクトリーへのパスに置き換えます。

```
# chattr +T dirname
```

chattr は e2fsprogs パッケージの一部として提供されます。

競合を減らす

GFS2 はクラスター内のノード間で通信を必要とする可能性があるグローバルロックのメカニズムを使用します。複数のノード間でファイルやディレクトリーの競合が起きるとパフォーマンスの低下を招きます。複数のノード間で共有するファイルシステムの領域をできるだけ小さくすることでキャッシュ間の無効化を招く危険性を最小限に抑えることができます。

第9章 ネットワーク

ネットワークサブシステムは、精度の高い接続で多くの異なるパーツから形成されています。したがって Red Hat Enterprise Linux 7 のネットワークはほとんどの作業に最適なパフォーマンスを提供し、またそのパフォーマンスを自動的に最適化するよう設計されています。このため通常は手作業によるネットワークパフォーマンスのチューニングは必要ありません。本章では実用的なネットワーク設定のシステムに適用可能な最適化について説明しています。

ネットワークパフォーマンスの問題は、ハードウェアの誤作動やインフラストラクチャーの障害が原因であることがあります。これらの問題の解決については、本書では扱いません。

9.1. 留意事項

チューニングに関して適切な決定をするには Red Hat Enterprise Linux のパケット受信について十分に理解しておく必要があります。本セクションではネットワークパケットの受信と処理について、また障害が発生しやすい箇所について説明しています。

Red Hat Enterprise Linux システムへ送信されるパケットはネットワークインターフェイスカード (NIC) で受信され、内蔵ハードウェアバッファーまたはリングバッファーのいずれかに置かれます。次に NIC によりハードウェア割り込み要求が送信され、その割り込み要求を処理するソフトウェア割り込み動作の作成が求められます。

ソフトウェア割り込み動作の一部としてパケットがバッファーからネットワークスタックへ転送されます。パケットおよびネットワークの設定に応じてパケットは転送または破棄されるかアプリケーションのソケット受信キューに渡され、ネットワークスタックから削除されます。このプロセスは、NIC ハードウェアバッファーにパケットがなくなるか、特定のパケット数(**/proc/sys/net/core/dev_weight**で指定)が転送されるまで継続されます。

Red Hat カスタマーポータルで利用可能な Red Hat Enterprise Linux Network Performance Tuning Guide には、Linux カーネルでのパケット受信に関する情報が含まれ、NIC チューニングの SoftIRQ ミス(netdev budget)、**tuned** チューニングデーモン、**numad** NUMA デーモン、CPU 電源状態、割り込み分散、一時停止フレーム、割り込み結合などを説明します。 アダプターキュー(**netdev** バックログ)、アダプター RX および TX バッファー、アダプター TX キュー、モジュールパラメーター、アダプターオフロード、Jumbo Frames、TCP および UDP プロトコルのチューニング、および NUMA 局所性。

9.1.1. チューニングを行う前に

ネットワークパフォーマンス関連の問題はほとんどの場合ハードウェアが正常に機能していないかったり、インフラストラクチャーが不完全であることが原因で発生します。ネットワークスタックのチューニングを行う前に、まずハードウェアおよびインフラストラチャーが期待通りに動作しているか必ず確認されることを強く推奨しています。

9.1.2. パケット受信におけるボトルネック

ネットワークスタックは大幅に自己最適化されていますが、ネットワークパケットの処理中にはボトルネックとなり、パフォーマンスが低下する可能性のあるポイントが多数あります。

NIC ハードウェアバッファーまたはリングバッファー

大量のパケットがドロップされるとハードウェアバッファーがボトルネックとなる場合があります。ドロップされたパケットを監視する方法については、「ethtool」 を参照してください。

ハードウェアまたはソフトウェアの割り込みキュー

割り込みにより、レイテンシーやプロセッサーの競合が増大する可能性があります。プロセッサー

で割り込みがどのように処理されるかについては、「IRQ (Interrupt Request - 割り込み要求) の処理」 を参照してください。システムで割り込み処理を監視する方法については、「/proc/interrupts」 を参照してください。割り込み処理に影響を与える設定オプションについては、「AMD64 および Intel 64 での割り込み親和性の設定」 を参照してください。

アプリケーションのソケット受信キュー

アプリケーションの受信キューのボトルネックは、要求元のアプリケーションにコピーされない多数のパケット、または **/proc/net/snmp** の UDP 入力エラー(**InErrors**)の増加によって示されます。こうしたエラーを監視する方法については、「ss」 および 「/proc/net/snmp」 を参照してください。

9.2. パフォーマンスの問題の監視と診断

Red Hat Enterprise Linux 7 ではシステムパフォーマンスの監視およびネットワークを設定しているサブシステムに関連するパフォーマンスの問題の診断などを行う際に便利なツールをいくつか用意しています。本セクションでは使用できるツールについて簡単に説明し、ネットワーク関連のパフォーマンス問題の監視と診断を行う方法について例を用いて説明していきます。

9.2.1. ss

ss はソケットに関する統計情報を出力するコマンドラインユーティリティーで、管理者が時間とともにデバイスパフォーマンスを評価することができます。デフォルトでは、**ss** は、確立された接続があるオープンでリッスンしていない TCP ソケットを一覧表示しますが、特定のソケットに関する統計を管理者が除外するのに役立つオプションが多数あります。

Red Hat は、Red Hat Enterprise Linux 7 の **netstat** よりも **ss** を推奨しています。

ss は、iproute パッケージで提供されます。詳細は man ページをご覧ください。

```
$ man ss
```

9.2.2. ip

ip ユーティリティーを使用すると、管理者はルート、デバイス、ルーティングポリシー、およびトンネルを管理および監視できます。**ip monitor** コマンドは、デバイス、アドレス、およびルートの状態を継続的に監視できます。

ip は、iproute パッケージで提供されます。ip の使い方については man ページをご覧ください。

```
$ man ip
```

9.2.3. dropwatch

dropwatch は、カーネルによって ドロップされたパケットを監視 および記録するインタラクティブなツールです。

詳細は、**dropwatch** の man ページを参照してください。

```
$ man dropwatch
```

9.2.4. ethtool

ethtool ユーティリティーを使用すると、管理者はネットワークインターフェイスカードの設定を表示および編集できます。特定デバイスでドロップされたパケット数などの統計情報を監視する場合に便利です。

ethtool -S と監視するデバイスの名前を使用して、指定したデバイスのカウンターの状態を表示できます。

```
$ ethtool -S devname
```

詳細については man ページをご覧ください。

```
$ man ethtool
```

9.2.5. /proc/net/snmp

/proc/net/snmp ファイルは、IP、ICMP、TCP、UDP の監視と管理のために snmp エージェントが使用するデータを表示します。このファイルを定期的にチェックして普段とは異なる値がないかを確認、パフォーマンス関連の問題となる可能性が潜んでいないか確認することができます。たとえば、**/proc/net/snmp** の UDP 入力エラー(**InErrors**)が増えると、ソケット受信キューのボトルネックを示している可能性があります。

9.2.6. SystemTap を使用したネットワークの監視

『Red Hat Enterprise Linux 7 SystemTap ビギナーズガイド』ではネットワークパフォーマンスのプロファイリングと監視に役立つサンプルスクリプトをいくつか紹介しています。

以下の **SystemTap** のサンプルスクリプトはネットワークに関連し、ネットワークパフォーマンスの問題を診断する際に役に立つ場合があります。デフォルトでは、これらは **/usr/share/doc/systemtap-client/examples/network** ディレクトリーにインストールされます。

nettop.stp

5 秒ごとにプロセスリスト (プロセス ID とコマンド)、送受信したパケット数、その間にプロセスによって送受信されたデータ量を出力します。

socket-trace.stp

Linux カーネルの **net/socket.c** ファイル内の各関数をインストルメント化し、トレースデータを出力します。

dropwatch.stp

5 秒ごとにカーネル内の場所で解放されたソケットバッファー数を出力します。**--all-modules** オプションを使用してシンボリック名を表示します。

latencytap.stp スクリプトは、異なるタイプのレイテンシーが 1 つ以上のプロセスに与える影響を記録します。レイテンシータイプのリストを 30 秒ごとに出力し、待機しているプロセスまたはプロセスの合計時間で降順でソートします。これは、ストレージとネットワークレイテンシーの両方の原因を特定するのに役立ちます。Red Hat は、このスクリプトに **--all-modules** オプションを使用して、レイテンシーイベントのマッピングをより有効化することを推奨します。デフォルトでは、このスクリプトは **/usr/share/doc/systemtap-client-*バージョン*/examples/profiling** ディレクトリーにインストールされます。

詳細は、Red Hat Enterprise Linux 7 SystemTap ビギナーズガイド を参照してください。

9.3. 設定ツール

Red Hat Enterprise Linux ではシステムの設定に役立つツールをいくつか提供しています。本セクションでは利用できるツールを簡単に説明し、Red Hat Enterprise Linux 7 でネットワーク関連のパフォーマンスの問題を解決する場合のツールの使用例を紹介しています。

ただし、ネットワークパフォーマンスに関連する問題はハードウェアが正常に機能していない、またはインフラストラクチャーが不完全であることが原因で発生する場合もありますので注意が必要です。ツールを使用してネットワークスタックのチューニングを行う前に、まずハードウェアおよびインフラストラチャーが期待通りに動作しているか必ず確認されることを強く推奨しています。

また、ネットワークパフォーマンスの特定の問題についてはネットワークのサブシステムを再設定するのではなくアプリケーション側を変更することで解決するものもあります。一般的にはアプリケーションが posix 呼び出しを頻繁に行うよう設定すると解決することがよくあります。アプリケーション領域にデータをキュー待ちさせることになってもデータを柔軟に格納して必要に応じメモリーにスワップインしたりスワップアウトすることができるようになるためです。

9.3.1. ネットワークパフォーマンス向けの Tuned プロファイル

Tuned サービスは、特定のユースケースでパフォーマンスを向上させるために、さまざまなプロファイルを提供します。以下のプロファイルはネットワークパフォーマンの改善に役立つプロファイルになります。

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

プロファイルについての詳細は、「tuned-adm」 を参照してください。

9.3.2. ハードウェアバッファーの設定

ハードウェアバッファーによって大量のパケットがドロップされている場合は、複数の解決策が考えられます。

入力トラフィックの速度が遅い

キューを満たす速度を下げるには、受信トラフィックのフィルター、参加するマルチキャストグループ数の削減、またはブロードキャストトラフィック量の削減を行います。受信トラフィックのフィルター方法に関する詳細はRed Hat Enterprise Linux 7 セキュリティーガイドを参照してください。マルチキャストグループの詳細は Red Hat Enterprise Linux 7 のクラスタリング関連のドキュメントを参照してください。ブロードキャストのトラフィックの詳細はRed Hat Enterprise Linux 7 システム管理者のガイドまたは設定するデバイスに関するドキュメントを参照してください。

ハードウェアバッファーキューのサイズ

キューのサイズを大きくすることでドロップされてしまうパケット数を減らしオーバーフローを防ぎます。ethtool コマンドでネットワークデバイスの rx/tx パラメーターを修正します。

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

キューのドレイン (解放) レートの変更

デバイス加重はデバイスで一度 (スケジュールされているプロセッサーのアクセス 1 回) に受信できるパケット数を参照します。***dev_weight*** パラメーターで制御されるデバイスの重みを増やすことで、キューが排出される速度を増やすことができます。**/proc/sys/net/core/dev_weight** ファイルの

内容を変更するか、procps-ng パッケージで提供される **sysctl** を使用して永続的に変更することで、このパラメーターを一時的に変更できます。

キューのドレインレートの変更は、通常、ネットワークのパフォーマンス低下を軽減する最も簡単な方法です。ただし、デバイスが一度に受信できるパケット数を増やすと、追加のプロセッサー時間が使用され、他のプロセスがスケジュールできません。そのため、パフォーマンス上の問題が発生する可能性があります。

9.3.3. 割り込みキューの設定

分析でレイテンシーが高いと、割り込みベースのパケット受信ではなく、ポーリングベースでメリットが得られます。

9.3.3.1. ビジーポーリングの設定

ビジーポーリングはソケット層のコードにネットワークデバイスの受信キューをポーリングさせ、ネットワークの割り込みは無効にすることでネットワーク受信パス内の待ち時間を低減します。これにより、割り込みおよび結果として生じるコンテキストスイッチによって生じる遅延が削除されます。ただし、CPU 使用率も増加します。ビジーポーリングは、CPU がスリープ状態にならないようにします。これにより、追加の電力消費が発生する可能性があります。

ビジーポーリングはデフォルトでは無効になっています。以下の手順で特定のソケットでのビジーポーリングを有効にします。

- ***sysctl.net.core.busy_poll*** を **0** 以外の値に設定します。このパラメーターはソケットのポーリングと選択用デバイスキューでパケットを待機する時間を管理します (マイクロ秒単位)。Red Hat は、**50** の値を推奨します。
- **SO_BUSY_POLL** ソケットオプションをソケットに追加します。

ビジーポーリングをグローバルに有効にするには、***sysctl.net.core.busy_read*** を **0** 以外の値に設定する必要があります。このパラメーターはソケットの読み取り用デバイスキューでパケットを待機する時間を管理します (マイクロ秒単位)。また、**SO_BUSY_POLL** オプションのデフォルト値も設定します。Red Hat では、ソケット数が少ない場合は **50**、ソケット数が多い場合は **100** の値を推奨します。ソケット数が非常に多くなる場合は (数百以上) 代わりに **epoll** を使用します。

ビジーポーリング動作は次のドライバーで対応しています。また、Red Hat Enterprise Linux 7 でも対応のドライバーになります。

- bnx2x
- be2net
- ixgbe
- mlx4
- myri10ge

Red Hat Enterprise Linux 7.1 からは次のコマンドを実行するとデバイスかビジーポーリングに対応するどうかをチェックすることもできるようになります。

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

これが **[fixed]** で **busy-poll:** を返す場合、デバイスでビジーポーリングを使用できます。

9.3.4. ソケット受信キューの設定

分析により、ソケットキューのドレインレートが遅すぎるためにパケットがドロップされていることが示唆された場合、結果として生じるパフォーマンスの問題を軽減する方法がいくつかあります。

受信トラフィックの速度を下げる

キューに到達する前にパケットをフィルターまたは破棄したり、デバイスのウェイトを下げたりすることで、キューがいっぱいになる速度を低減します。

アプリケーションのソケットキューの深さを増やす

ソケットキューがバーストで制限されたトラフィック量を受信した場合は、ソケットキューの深さをトラフィックのバーストサイズに一致するように増やすと、パケットがドロップされるのを防ぐことができます。

9.3.4.1. 受信トラフィックの速度を下げる

受信トラフィックの速度を下げるには、受信トラフィックにフィルターをかけたり、ネットワークインターフェイスカードのデバイス加重を減らします。受信トラフィックのフィルター方法に関する詳細はRed Hat Enterprise Linux 7 セキュリティーガイドを参照してください。

デバイス加重はデバイスで一度 (スケジュールされているプロセッサーのアクセス 1 回) に受信できるパケット数を参照します。デバイスの重みは、***dev_weight*** パラメーターによって制御されます。**/proc/sys/net/core/dev_weight** ファイルの内容を変更するか、procps-ng パッケージで提供される **sysctl** を使用して永続的に変更することで、このパラメーターを一時的に変更できます。

9.3.4.2. キューの深さを増やす

アプリケーションソケットキューの深さを増やすことは、通常、ソケットキューのドレインレートを改善する最も簡単な方法ですが、長期的な解決策になる可能性は低いです。

キューの深さを増やすには、以下の変更のいずれかを行ってソケット受信バッファーのサイズを増やします。

/proc/sys/net/core/rmem_default の値を増やす

このパラメーターは、ソケットが使用する受信バッファーのデフォルトサイズを制御します。この値は、**/proc/sys/net/core/rmem_max** の値以下である必要があります。

setsockopt を使用して **SO_RCVBUF** に大きな値を設定する

このパラメーターにより、ソケットの受信バッファーの最大サイズを制御します (バイト単位)。**getsockopt** システムコールを使用して、バッファーの現在の値を判断します。詳細については、socket(7) の man ページをご覧ください。

9.3.5. RSS (Receive-Side Scaling) の設定

マルチキュー受信とも呼ばれる Receive Side Scaling (RSS) は、ネットワーク受信処理を複数のハードウェアベースの受信キューに分散し、受信ネットワークトラフィックを複数の CPU で処理できるようにします。RSS を使用すると、1 つの CPU をオーバーロードして発生する受信中断プロセッシングにおけるボトルネックを緩和し、ネットワークレイテンシーを低減させることができます。

お使いのネットワークインターフェイスカードが RSS に対応しているかどうかを確認するには、複数の割り込み要求キューが **/proc/interrupts** 内のインターフェイスに関連付けられているかどうかを確認します。たとえば、**p1p1** インターフェイスに関心がある場合は、以下のようになります。

```
# egrep 'CPU|p1p1' /proc/interrupts
   CPU0    CPU1    CPU2    CPU3    CPU4    CPU5
89:  40187       0       0       0       0       0   IR-PCI-MSI-edge   p1p1-0
90:      0     790       0       0       0       0   IR-PCI-MSI-edge   p1p1-1
91:      0       0     959       0       0       0   IR-PCI-MSI-edge   p1p1-2
92:      0       0       0    3310       0       0   IR-PCI-MSI-edge   p1p1-3
93:      0       0       0       0     622       0   IR-PCI-MSI-edge   p1p1-4
94:      0       0       0       0       0    2475   IR-PCI-MSI-edge   p1p1-5
```

上記の出力は、NIC ドライバーが **p1p1** インターフェイスに 6 つの受信キューを作成したことを示しています(**p1p1-0** から **p1p1-5**)。また、各キューで処理された割り込みの数と、割り込みをした CPU の数も表示されます。この場合、デフォルトではこの特定の NIC ドライバーは CPU ごとに 1 つのキューを作成するため 6 つのクエリーがあります。また、このシステムには CPU が 6 つあります。これは、NIC ドライバー間で非常に一般的なパターンです。

または、ネットワークドライバーの読み込み後に **ls -1 /sys/devices/*/*/*device_pci_address*/msi_irqs** の出力を確認することもできます。たとえば、PCI アドレスが **0000:01:00.0** のデバイスを確認する場合は、以下のコマンドでそのデバイスの割り込み要求キューを一覧表示できます。

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

RSS はデフォルトで有効になっています。RSS のキューの数 (または、ネットワークアクティビティーを処理する CPU 数) は該当するネットワークデバイスドライバーで設定されます。**bnx2x** ドライバーの場合は、***num_queues*** で設定されます。**sfc** ドライバーの場合は、***rss_cpus*** パラメーターで設定されます。通常、これは **/sys/class/net/*デバイス*/queues/*rx-queue*/** で設定されます。*device* はネットワークデバイスの名前(**eth1** など)で、*rx-queue* は適切な受信キューの名前です。

RSS を設定する場合、Red Hat は、物理 CPU コアごとにキュー数を 1 つに制限することを推奨します。ハイパースレッディングは、分析ツールで個別のコアとして表されますが、ハイパースレッディングなどの論理コアを含むすべてのコアにキューを設定することは、ネットワークパフォーマンスにあまり良い影響はありません。

有効にすると、RSS は各 CPU のがキューした処理量に基づいて、利用可能な CPU 間でネットワーク処理を均等に分散します。ただし、**ethtool *--show-rxfh-indir*** パラメーターおよび ***--set-rxfh-indir*** パラメーターを使用して、ネットワークアクティビティーの分散方法を変更し、特定タイプのネットワークアクティビティーを他のタイプよりも重要なものとして重み付けすることができます。

irqbalance デーモンを RSS と組み合わせて使用すると、ノード間のメモリー転送やキャッシュラインバウンスの可能性を減らすことができます。これにより、ネットワークパケット処理のレイテンシーが短縮されます。

9.3.6. 受信パケットステアリング (RPS) の設定

RPS (Receive Packet Steering) は RSS と似ていますが、パケットを処理するための特定の CPU に転送するのに使用されます。ただし、RPS はソフトウェアレベルで実装されており、単一のネットワークインターフェイスカードのハードウェアキューがネットワークトラフィックのボトルネックとなるのを防

ぐのに役立ちます。

RPS は、ハードウェアベースの RSS と比較していくつかの利点があります。

- RPS は、どのネットワークインターフェイスカードでも使用できます。
- 新しいプロトコルに対応するために、ソフトウェアフィルターを RPS に簡単に追加できます。
- RPS は、ネットワークデバイスのハードウェア割り込みレートを増やしません。ただし、プロセッサー間割り込みは導入されます。

RPS は、**/sys/class/net/** デバイス **/queues/*rx-queue*/rps_cpus** ファイルでネットワーク***デバイス***と受信キュー ごとに設定されます。*device* はネットワークデバイスの名前(**eth0**など)、*rx-queue* は適切な受信キューの名前です(**rx**-0 など)。

rps_cpus ファイルのデフォルト値は **0** です。この値の場合は RPS は無効です。したがってネットワーク割り込みを処理する CPU がパケットも処理します。

RPS を有効にするには、指定されたネットワークデバイスからのパケットを処理し、キューを受信する CPU で適切な **rps_cpus** ファイルを設定します。

rps_cpus ファイルはコンマ区切りの CPU ビットマップを使用します。したがって、CPU がインターフェイスの受信キューの割り込みを処理できるようにするには、プレースホルダー内のその位置の値を 1 に設定します。たとえば、CPU 0、1、2、および 3 で割り込みを処理するには、**rps_cpus** の値を **f** (16 進数の場合は 15)に設定します。バイナリー表現では、15 は **00001111** (1+2+4+8)です。

単一送信キューを持つネットワークデバイスには、同じメモリードメインの CPU を使用するように RPS を設定することで、最善のパフォーマンスを実現できます。NUMA 以外のシステムでは、利用可能な CPU がすべて使用できることを意味します。ネットワーク割り込みレートが非常に高い場合には、ネットワーク割り込みを処理する CPU を除くと、パフォーマンスが向上することがあります。

複数キューのネットワークデバイスでは、通常、RPS と RSS の両方を設定する利点はありません。これは、RSS はデフォルトで CPU を各受信キューにマップするように設定されるためです。ただし、CPU よりもハードウェアキューの方が少ない場合、および同一メモリードメインの CPU を使用するように RPS が設定されている場合は、RPS が利点ももたらす可能性があります。

9.3.7. RFS (Receive Flow Steering) の設定

Receive Flow Steering (RFS) は RPS の動作を拡張し、CPU キャッシュのヒットレートを増やし、ネットワークレイテンシーを低減します。RPS はキューの長さのみに基づいてパケットを転送し、RFS は RPS バックエンドを使用して最も適切な CPU を算出します。次にパケットを消費するアプリケーションの場所に基づいてパケットを転送します。これにより、CPU キャッシュの効率が上がります。

RFS はデフォルトで無効になっています。RFS を有効にするには、以下の 2 つのファイルを編集する必要があります。

/proc/sys/net/core/rps_sock_flow_entries

このファイルの値を同時にアクティブになるであろう接続の予測最大数に設定します。中程度のサーバー負荷には **32768** の値が推奨されます。入力した値はすべて直近の 2 の累乗で切り上げられます。

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

device を、設定するネットワークデバイスの名前(**eth0**など)に置き換え、*rx-queue* を設定する受信キュー(**rx-0** など)に置き換えます。

このファイルの値を、**rps_sock_flow_entries** の値を **N** で割った値に設定します。ここでの **N** は、

デバイスの受信キューの数です。たとえば、**rps_flow_entries** が **32768** に設定され、16 の設定済み受信キューがある場合は、**rps_flow_cnt** を **2048** に設定する必要があります。単一キューデバイスの場合、**rps_flow_cnt** の値は **rps_sock_flow_entries** の値と同じです。

1 つの送信者から受信したデータは複数の CPU に送信されません。1 つの送信元から受信するデータ量が 1 つの CPU が処理できるものよりも大きい場合は、割り込みの数や CPU の処理量を減らすために、より大きなフレームサイズを設定します。NIC オフロードオプション または高速 CPU について検討してください。

numactl または **taskset** を RFS と併用して、アプリケーションを特定のコア、ソケット、または NUMA ノードに固定することを検討してください。これにより、パケットが順番に処理されるようにします。

9.3.8. アクセラレート RFS の設定

アクセラレート RFS は、ハードウェア補助を追加することで、RFS の速度を強化します。RFS と同様に、パケットはパケットを使用するアプリケーションの場所に基づいて転送されます。ただし、従来の RFS とは異なり、パケットはデータを消費するスレッドに対してローカルとなる CPU に直接送信されます。つまりアプリケーションを実行している CPU、またはキャッシュ階層内のその CPU にローカルとなる CPU のいずれかになります。

RFS は、以下の条件が満たされている場合にのみ利用できます。

- アクセラレート RFS がネットワークインターフェイスカード対応になっていること。アクセラレート RFS は、**ndo_rx_flow_steer() netdevice** 関数をエクスポートするカードでサポートされています。
- **ntuple** フィルターを有効にする必要があります。

これらの条件が満たされると、従来の RFS 設定に基づいて、キューマッピングへの CPU が自動的に消費されます。つまり、キューマッピングへの CPU は各受信キューに対してドライバーによって設定された IRQ 機関に基づいて重複排除されます。従来の RFS の設定方法については、「RFS (Receive Flow Steering) の設定」 を参照してください。

Red Hat では RFS を使用すべき状況でネットワークインターフェイスカードがハードウェアアクセラレートに対応している場合は常にアクセラレート RFS の使用を推奨しています。

付録A ツールについて

パフォーマンスの調整に使用できる Red Hat Enterprise Linux 7 の各種ツールについて簡単に説明しています。詳細および最新情報についてはツールの man ページをご覧ください。

A.1. IRQBALANCE

irqbalance は、システムパフォーマンスを向上させるために、プロセッサー全体にハードウェア割り込みを分散するコマンドラインツールです。デフォルトではデーモンとして実行されますが、**--oneshot** オプションで 1 回きりの実行も可能です。

パフォーマンス改善には、以下のパラメーターが便利です。

--powerthresh

CPU が省電力モードになる前にアイドル状態になることが可能な CPU 数を設定します。しきい値を超える CPU 数が平均の **softirq** ワークロードよりも多い CPU で、平均を上回る CPU が複数の標準差を超えておらず、複数の **irq** をそれらに割り当てると、CPU は省電力モードに配置されます。省電力モードでは、CPU は **irq** バランシングの一部ではないため、不必要に陥ることはありません。

--hintpolicy

irq カーネルアフィニティーヒントの処理方法を決定します。有効な値は **exact** (**irq** アフィニティーヒントは常に適用)、サブセット (**irq** はバランスが取れますが、割り当てられたオブジェクトはアフィニティーヒントのサブセット)、または **ignore** (**irq** アフィニティーヒントは完全に無視されます)。

--policyscript

各割り込み要求に対して実行するスクリプトの場所を定義します。デバイスパスと **irq** 番号を引数として渡し、**irqbalance** が想定する終了コードをゼロにします。定義されるスクリプトでは、渡された **irq** の管理で **irqbalance** をガイドするために、ゼロまたはそれ以上のキーと値のペアを指定できます。

有効な鍵の値のペアとして認識されるのは、以下のものです。

ban

有効な値は **true** （渡された **irq** を分散から除外）または **false** （この **irq**のバランシングによる分散）です。

balance_level

ユーザーが渡された **irq** のバランスレベルを上書きすることを許可します。デフォルトでは、バランスレベルは **irq** を所有するデバイスの PCI デバイスクラスに基づいています。有効な値は、**none**、**package**、**cache**、または **core** です。

numa_node

渡された **irq** に対してローカルとみなされる NUMA ノードのユーザーオーバーライドを許可します。ローカルノードについての情報が ACPI で指定されていない場合は、デバイスはすべてのノードから等距離とみなされます。有効な値は、特定の NUMA ノードを識別する整数(0 から)と、すべてのノードから **irq** を同等と見なすことを指定する **-**1 です。

--banirq

指定割り込み要求番号を持つ割り込みが禁止割り込みリストに追加されます。

IRQBALANCE_BANNED_CPUS 環境変数を使用して、**irqbalance** で無視される CPU のマスクを指定することもできます。

詳細は、以下の man ページを参照してください。

```
$ man irqbalance
```

A.2. ETHTOOL

ethtool ユーティリティーを使用すると、管理者はネットワークインターフェイスカードの設定を表示および編集できます。特定デバイスでドロップされたパケット数などの統計情報を監視する場合に便利です。

ethtool、そのオプション、およびその使用方法は、の man ページに包括的な説明があります。

```
$ man ethtool
```

A.3. SS

ss はソケットに関する統計情報を出力するコマンドラインユーティリティーで、管理者が時間とともにデバイスパフォーマンスを評価することができます。デフォルトでは、**ss** は、確立された接続があるオープンでリッスンしていない TCP ソケットを一覧表示しますが、特定のソケットに関する統計を管理者が除外するのに役立つオプションが多数あります。

一般的に使用されるコマンドの 1 つは s **-tmpie** です。すべての TCP ソケット(**t**、内部 TCP 情報(**i**)、ソケットのメモリー使用量(**m**)、ソケットを使用したプロセス（()、および詳細なソケット情報(**i**)を表示します。

Red Hat は、**Red Hat Enterprise Linux 7** の **netstat** よりも **ss** を推奨しています。

ss は、**iproute** パッケージで提供されます。詳細は **man** ページをご覧ください。

```
$ man ss
```

A.4. TUNED

Tuned は、チューニングプロファイルを設定することで、特定のワークロードでより良く動作するようにオペレーティングシステムを調整するチューニングデーモンです。また、**CPU** やネットワーク使用の変化に反応するよう設定し、動作しているデバイスではパフォーマンスを改善し動作していないデバイスでは電力消費を抑えるよう設定を調整することもできます。

動的チューニングの動作を設定するには、**/etc/tuned/tuned-main.conf** ファイルの ***dynamic_tuning*** パラメーターを編集します。その後、**Tuned** は定期的にシステム統計を分析し、それらを使用してシス

テムチューニング設定を更新します。***update_interval*** パラメーターを使用して、これらの更新の間隔を秒単位で設定できます。

tuned の詳細については **man** ページをご覧ください。

```
$ man tuned
```

A.5. TUNED-ADM

tuned-adm は、**Tuned** プロファイルに切り替えて、特定のユースケースでパフォーマンスを向上させることができるコマンドラインツールです。また、システムを評価し、推奨されるチューニングプロファイルを出力する **tuned-adm recommend** サブコマンドも提供します。

Red Hat Enterprise Linux 7 では、**Tuned** に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追加されました。これにより、**Tuned** に統合されていない機能を使用して **Tuned** プロファイルを拡張できます。

Red Hat Enterprise Linux 7 では、プロファイル定義ファイルに ***include*** パラメーターも提供されるため、既存のプロファイルで独自の **Tuned** プロファイルをベースにすることができます。

Tuned では、以下のチューニングプロファイルが提供され、**Red Hat Enterprise Linux 7** でサポートされています。

throughput-performance

処理能力の改善に焦点をあてたサーバープロファイルになります。デフォルトのプロファイルでほとんどのシステムに推奨となります。

このプロファイルは、 **intel_pstate** と **min_perf_pct=100** を設定して節電によりパフォーマンスを重視します。透過的な **Huge Page** を有効にし、**cpupower** を使用して **performance cpufreq** ガバナーを設定します。また、***kernel.sched_min_granularity_ns*** を **10 DHEs**、***kernel.sched_wakeup_granularity_ns*** を **15 DHEs** に、***vm.dirty_ratio*** を **40%** に設定します。

latency-performance

待ち時間の短縮に焦点をあてたサーバープロファイルです。**c-state** チューニングや **Transparent Huge Page** の **TLB** 効率性の改善を目的とする待ち時間に制約のある作業負荷に推奨のプロファイルです。

このプロファイルは、**intel_pstate** と **max_perf_pct=100** を設定して節電によりパフォーマンスを重視します。透過的な **Huge Page** を有効にし、**cpupower** を使用して **performance cpufreq** ガバナーを設定し、***cpu_dma_latency*** 値 **1** を要求します。

network-latency

ネットワークの待ち時間短縮に焦点をあてたサーバープロファイルです。

このプロファイルは、 **intel_pstate** と **min_perf_pct=100** を設定して節電によりパフォーマンスを重視します。透過的な巨大ページと **NUMA** 自動負荷分散が無効になります。また、**cpupower** を使用して **performance cpufreq** ガバナーを設定し、***cpu_dma_latency*** 値 **1** を要求します。また、***busy_read*** および ***busy_poll*** 時間を **50 DHEs** に設定し、***tcp_fastopen*** を **3** に設定します。

network-throughput

ネットワーク処理能力の改善に焦点をあてたサーバープロファイルです。

intel_pstate と **max_perf_pct=100** を設定しカーネルのネットワークバッファーサイズを大きくして節電よりパフォーマンスを重視します。透過的な **Huge Page** を有効にし、**cpupower** を使用して **performance cpufreq** ガバナーを設定します。また、***kernel.sched_min_granularity_ns*** を **10 DHEs**、***kernel.sched_wakeup_granularity_ns*** を **15 DHEs** に、 ***vm.dirty_ratio*** を **40%** に設定します。

virtual-guest

Red Hat Enterprise Linux 7 仮想マシンと **VMware** ゲストでのパフォーマンスの最適化に焦点をあてたプロファイルです。

このプロファイルは、**intel_pstate** と **max_perf_pct=100** を設定して節電によりパフォーマンスを重視します。また仮想マシンの **swap** を低減します。透過的な **Huge Page** を有効にし、**cpupower** を使用して **performance cpufreq** ガバナーを設定します。また、***kernel.sched_min_granularity_ns*** を **10 DHEs**、***kernel.sched_wakeup_granularity_ns*** を **15 DHEs** に、***vm.dirty_ratio*** を **40%** に設定します。

virtual-host

Red Hat Enterprise Linux 7 仮想ホストでのパフォーマンスの最適化に焦点をあてたプロファイルです。

このプロファイルは、**intel_pstate** と **max_perf_pct=100** を設定して節電によりパフォーマンスを重視します。また仮想マシンの **swap** を低減します。**Transparent Huge Page** を有効にしダー

ティーなページをより頻繁にディスクに書き戻します。**cpupower** を使用して **performance cpufreq** ガバナーを設定します。また、***kernel.sched_min_granularity_ns*** を **10 DHEs**、***kernel.sched_wakeup_granularity_ns*** を **15 DHEs**、***kernel.sched_migration_cost*** を **5 DHEs** に、***vm.dirty_ratio*** を **40%** に設定します。

cpu-partitioning

cpu-partitioning プロファイルは、システム **CPU** を分離されたハウスキーピング **CPU** に分割します。分離された **CPU** のジッターと割り込みを減らすために、プロファイルは分離された **CPU** を、ユーザー空間プロセス、可動カーネルスレッド、割り込みハンドラー、およびカーネルタイマーから削除します。

ハウスキーピング **CPU** は、すべてのサービス、シェルプロセス、およびカーネルスレッドを実行できます。

/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。設定オプションは以下のようになります。

isolated_cores=*cpu-list*

分離する **CPU** をリスト表示します。分離された **CPU** のリストはコンマで区切るか、ユーザーが範囲を指定できます。**3-5** などのダッシュ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。

no_balance_cores=*cpu-list*

システム全体のプロセスの負荷分散時に、カーネルに考慮されない **CPU** のリストを表示します。このオプションは任意です。通常、これは **isolated_cores** と同じリストです。

cpu-partitioning の詳細は、**tuned-profiles-cpu-partitioning(7)** の **man** ページを参照してください。

tuned-adm で提供される省電力プロファイルの詳細は、 **Red Hat Enterprise Linux 7 Power Management Guide** を参照してください。

tuned-adm の使用方法は、の **man** ページを参照してください。

```
$ man tuned-adm
```

A.6. PERF

perf ツールは便利なコマンドを複数提供します。その一部は本セクションに記載されています。**perf** の詳細は、**Red Hat Enterprise Linux 7 開発者ガイド** を参照してください。または、**man** ページを参照してください。

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。デフォルトの測定イベント以外のイベントに関する統計を収集する場合はオプションフラグを使用することができます。**Red Hat Enterprise Linux 6.4** では、**perf stat** を使用して、**1** つ以上の指定されたコントロールグループ**(cgroup)**に基づいて監視をフィルターできるようになりました。

詳細については **man** ページをご覧ください。

```
$ man perf-stat
```

perf record

このコマンドは、パフォーマンスデータをファイルに記録し、後で **perf report** を使用して分析できます。詳細については **man** ページをご覧ください。

```
$ man perf-record
```

perf report

ファイルからパフォーマンスデータを読み取り、記録されたデータの分析を行います。詳細については **man** ページをご覧ください。

```
$ man perf-report
```

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、システムのソフトウェア設定とパフォーマンス監視ハードウェアによって異なります。詳細については **man** ページをご覧ください。

```
$ man perf-list
```

perf top

このコマンドは、**top** ツール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。詳細については **man** ページをご覧ください。

```
$ man perf-top
```

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。追跡するターゲットを増やすこともできます。全リストは **man** ページをご覧ください。

```
$ man perf-trace
```

A.7. PERFORMANCE CO-PILOT (PCP)

Performance Co-Pilot (PCP) は数多くのコマンドラインツール、グラフィカルツール、ライブラリーなどを備えています。これらのツールに関する詳細については、それぞれの **man** ページを参照してください。

表**A.1 Red Hat Enterprise Linux 7** で **Performance Co-Pilot** により配布されるシステムサービス

名前	説明
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。
pmmgr	ゼロ以上の設定ディレクトリーに従って、PMCD (Performance Metric Collector Daemon) を実行している、検出された一連のローカルおよびリモートのホストで PCP デーモンのコレクションを管理します。
pmproxy	PMCD (Performance Metric Collector Daemon) プロキシーサーバー
pmwebd	HTTP プロトコルを使用して、Performance Co-Pilot クライアント API のサブセットを RESTful Web アプリケーションにバインドします。

表**A.2 Red Hat Enterprise Linux 7** で **Performance Co-Pilot** により配布されるツール

名前	説明
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。

名前	説明
pmatop	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。
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。
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。
pmcollectl	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システムレベルデータを収集して表示します。
pmconfig	設定パラメーターの値を表示します。
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。
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。
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。
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。
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。
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。
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の I/O 統計を報告します(- x dm** オプションを使用します)。
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。

名前	説明
pmlogconf	**pmlogger** 設定ファイルを作成および変更します。
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。
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。
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	トレースの PMDA (Performance Metrics Domain Agent) に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。

表**A.3 XFS** 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.*	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_tree.*	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数のカウンター。同期および非同期トランザクション数、および空のトランザクション数が含まれます。

メトリックグループ	提供されたメトリック
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ってフラッシュされたファイルデータのバイト数と、ディスクの連続したスペースおよび連続していないスペースにフラッシュされたバッファー数のカウンター。
xfs.attr.*	すべての XFS ファイルシステム上での属性 get、set、remove、list 操作の数。
xfs.quota.*	XFS ファイルシステム上の quota 操作のメトリック。クオータの再要求数、クオータキャッシュのミス数、キャッシュのヒット数、およびクオータデータの再要求数の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。

表**A.4** デバイスごとの **XFS** 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_tree.*	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。

メトリックグループ	提供されたメトリック
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。

A.8. VMSTAT

vmstat は 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** アクティビティーに関するレポートを出力します。最後にマシンを起動した時点または最後の報告を行った時点からのイベント平均を瞬時に報告します。

-a

アクティブおよび非アクティブなメモリーを表示します。

-f

起動時からのフォーク数を表示します。これには **fork**、**vfork**、**clone** のシステムコールが含まれ、作成されたタスク数の合計と同じになります。各プロセスはスレッドの使用により **1** タスクまたは複数のタスクで表されます。表示は繰り返されません。

-m

スラブ情報を表示します。

-n

ヘッダーが定期的に表示されるのではなく、**1** 回表示されるように指定します。

-s

各種イベントのカウンターとメモリー統計の表を表示します。表示は繰り返されません。

delay

リポート間の遅延を秒単位で指定します。遅延を指定しない場合はマシンを最後に起動した時点からの平均値のレポートが一つのみ出力されます。

count

システムに関するレポートの回数を指定します。**count** が指定されておらず、**delay** が定義されている場合、**vmstat** は無期限に報告します。

-d

ディスク統計を表示します。

-p

パーティション名を値として取り、そのパーティションの詳細な統計をレポートします。

-S

レポートで出力される単位を指定します。有効な値は、**k (1000** バイト**)**、**K (1024** バイト**)**、**m (1,000,000** バイト**)**、または **M (1,048,576** バイト**)**です。

-D

ディスクアクティビティーに関する要約統計をレポートします。

各出力モードで提供される出力に関する詳細は **man** ページをご覧ください。

```
$ man vmstat
```

A.9. X86_ENERGY_PERF_POLICY

x86_energy_perf_policy ツールを使用すると、管理者はパフォーマンスとエネルギー効率の相対的な重要度を定義できます。これは、**kernel-tools** パッケージで提供されます。

現行ポリシーを表示するには、以下のコマンドを実行します。

```
# x86_energy_perf_policy -r
```

新たなポリシーを設定するには、以下のコマンドを実行します。

```
# x86_energy_perf_policy profile_name
```

profile_name を次のプロファイルのいずれかに差し替えます。

performance

プロセッサーは省エネのためパフォーマンスを犠牲にはしません。これはデフォルト値です。

normal

大幅な省エネとなる可能性がある場合はマイナーなパフォーマンス低下を許容します。ほとんどのサーバーおよびデスクトップで妥当な設定になります。

powersave

最大限の省エネを目的とし大幅なパフォーマンス低下の可能性を受け入れます。

x86_energy_perf_policy の使い方については **man** ページをご覧ください。

```
$ man x86_energy_perf_policy
```

A.10. TURBOSTAT

turbostat ツールは、システムがさまざまな状態で費やした時間に関する詳細情報を提供します。**Turbostat** は **kernel-tools** パッケージで提供されます。

デフォルトでは、**turbostat** はシステム全体のカウンター結果の概要を出力し、続いて次の見出しの下に **5** 秒ごとにカウンター結果を出力します。

pkg

プロセッサーのパッケージ番号。

コア

プロセッサーのコア番号。

CPU

Linux の **CPU (**論理プロセッサー**)** 番号。

%c0

CPU が命令をリタイアした間隔の割合。

GHz

この数値が **TSC** の値よりも大きい場合、**CPU** はターボモードです。

TSC

間隔全体の平均クロック速度。

%c1、**%c3**、**%c6**

プロセッサーがそれぞれ **c1**、**c3**、または **c6** 状態であった間隔の割合。

%pc3 または **%pc6**

プロセッサーが **pc3** または **pc6** の各状態だったあいだの間隔の割合

-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** 秒ごとに結果を出力します。

注記

次回の **Intel** プロセッサーには **c** 状態が追加される可能性があります。**Red Hat Enterprise Linux 7.0** 以降、**turbostat** は **c7**、**c8**、**c9**、および **c10** の状態に対応します。

A.11. NUMASTAT

numastat ツールは **numactl** パッケージで提供され、**NUMA** ノードごとにプロセスおよびオペレーティングシステムのメモリー統計（割り当てヒットやミスなど）を表示します。**numastat** コマンドのデフォルトの追跡カテゴリーを以下に概説します。

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。各 **numa_miss** イベントには、対応する **numa_foreign** イベントが別のノードにあります。

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

別のノード上でのプロセスによって当該ノードへの割り当てに成功したページ数

以下のオプションのいずれかを指定すると、表示される単位がメモリーのメガバイトに変更され（およそ **2** 桁に丸められます）、以下のように他の特定の **numastat** 動作が変更されます。

-c

表示情報の表を横方向に縮小します。これは、**NUMA** ノード数が多いシステムでは有用ですが、コラムの幅とコラム間の間隔はあまり予測可能ではありません。このオプションが使用されると、メモリー量は一番近いメガバイトに切り上げ/下げられます。

-m

/proc/meminfo にある情報と同様に、ノードごとにシステム全体のメモリー使用量情報を表示します。

-n

元の **numastat** コマンドと同じ情報を表示します **(numa_hit**、**numa_miss**、**numa_foreign**、**interleave_hit**、**local_node**、および **other_node)**。測定単位としてメガバイトを使用して、更新されたフォーマットを表示します。

-p pattern

指定されたパターンのノードごとのメモリー情報を表示します。**pattern** の値が数字で設定される場合、**numastat** は数値プロセス識別子であると想定します。それ以外の場合は、**numastat** は指定されたパターンをプロセスコマンドラインで検索します。

-p オプションの値の後に入力されるコマンドライン引数は、フィルターにかける追加のパターンとみなされます。追加のパターンは、フィルターを絞り込むのではなく拡張します。

-s

表示データを降順に並び替え、**(total** コラムの**)** メモリー消費量の多いものが最初に表示されます。

オプションでは、**node** を指定すると、表はその **node** のコラムにしたがって並び替えられます。このオプション使用時には、以下のように **node** の値は **-s** オプションの直後に続けます。

```
numastat -s2
```

このオプションと値の間に空白スペースを入れないでください。

-v

詳細情報を表示します。つまり、複数プロセスのプロセス情報が各プロセスの詳細情報を表示します。

-V

numastat のバージョン情報を表示します。

-z

情報情報から値が **0** の行と列のみを省略します。表示目的で **0** に切り下げられている **0** に近い値は、表示出力から省略されません。

A.12. NUMACTL

numactl を使用 すると、管理者は指定されたスケジューリングまたはメモリー配置ポリシーでプロセスを実行できます。**numactl** は、共有メモリーセグメントまたはファイルに永続ポリシーを設定し、プロセスのプロセッサーアフィニティーとメモリーアフィニティーを設定することもできます。

numactl は 便利なオプションを多数提供します。ここでは、それらオプションのいくつかを紹介し、使用方法を提示していますが、完全なものではありません。

--hardware

ノード間の相対距離を含む、システムで使用可能なノードのインベントリーを表示します。

--membind

メモリーが特定のノードからのみ割り当てられるようにします。指定された場所に利用可能なメモリーが十分にない場合は、割り当ては失敗します。

--cpunodebind

指定されたコマンドとその子プロセスが、指定されたノードでのみ実行されるようにします。

--phycpubind

指定されたコマンドとその子プロセスが、指定されたプロセッサーでのみ実行されるようにします。

--localalloc

メモリーが常にローカルノードから割り当てられるように指定します。

--preferred

メモリーを割り当てる元となるノードを指定します。この指定ノードからメモリーが割り当てられない場合は、別のノードがフォールバックとして使用されます。

これらおよび他のパラメーターについての詳細は、以下の **man** ページを参照してください。

```
$ man numactl
```

A.13. NUMAD

numad は自動 **NUMA** アフィニティー管理デーモンです。**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。

numad が有効な場合は、その動作により自動 **NUMA** バランシングのデフォルト動作が上書きされる点に注意してください。

A.13.1. コマンドラインからの numad の使用

numad を実行可能ファイルとして使用するには、単に以下を実行します。

```
# numad
```

numad の実行中に、アクティビティーは **/var/log/numad.log** に記録されます。アクティビティーは、以下のコマンドで停止されるまで継続されます。

```
# numad -i 0
```

numad を停止しても、**NUMA** アフィニティーを改善するために行った変更は削除されません。システムが大幅に変更される場合、**numad** を再度実行すると、新しい条件下でパフォーマンスを向上させるためにアフィニティーが調整されます。

numad 管理を特定プロセスに制限するには、以下のオプションで開始します。

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

-p *pid*

指定 ***pid*** を明示的な対象リストに加えます。指定されたプロセスは **numad** プロセスの重要度しきい値に達するまで管理されません。

-S 0

これにより、プロセススキャンのタイプが **0** に設定され、**numad** 管理が明示的に含まれるプロセスに制限されます。

利用可能な **numad** オプションの詳細は **numad** の **man** ページを参照してください。

```
$ man numad
```

A.13.2. numad のサービスとしての使用

numad はサービスとして実行されますが、現在のシステムのワークロードに基づいてシステムを動的にチューニングしようとします。アクティビティーは **/var/log/numad.log** に記録されます。

サービスを開始するには、以下のコマンドを実行します。

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

起動後もサービスを維持する場合は以下のコマンドを実行します。

```
# chkconfig numad on
```

利用可能な **numad** オプションの詳細は **numad** の **man** ページを参照してください。

```
$ man numad
```

A.13.3. プレプレースメントアドバイス

numad は、さまざまなジョブ管理システムでクエリーできるプレプレースメントアドバイスサービスを提供し、プロセスに対する **CPU** およびメモリーリソースの初期バインディングを提供します。**numad** が実行可能ファイルとして実行しているかサービスとして実行しているかに関係なく、このプレプレースメントアドバイスを利用することができます。

A.13.4. KSM をともなう numad の使用

KSM を **NUMA** システムで使用している場合は、**/sys/kernel/mm/ksm/merge_nodes** パラメーターの値を **0** に変更し、**NUMA** ノード全体でページのマージを回避します。これを実行しないと、**KSM** はノードにまたがってページをマージするので、リモートメモリーアクセスが増大します。また、カーネルメモリーが計算した統計情報は、ノード間での大量のマージ後にはそれぞれの間で相反する場合があります。そのため、**KSM** デーモンが多くのメモリーページをマージした後に、**numad** は利用可能なメモリーの正確な量と場所について混乱する可能性があります。**KSM** は、システムにメモリーをオーバーコミットしている場合にのみ、有用なものです。システムに未使用のメモリーが大量にあると、**KSM** デーモンをオフにして無効にすることでパフォーマンスが高まる場合があります。

A.14. OPROFILE

OProfile は、**oprofile** パッケージで提供されるオーバーヘッドコストの少ない、システム全体のパフォーマンス監視ツールです。プロセッサー上にあるパフォーマンス監視ハードウェアを使用して、メモリーの参照時期、第 **2** レベルのキャッシュ要求の回数、及び受け取ったハードウェア割り込みの回数など、システム上のカーネルと実行可能ファイルに関する情報を取得します。**OProfile** は、**Java Virtual Machine (JVM)** で実行されるアプリケーションのプロファイリングも実行できます。

OProfile は以下のツールを提供します。従来の **opcontrol** ツールと新しい **operf** ツールは相互に排他的であることに注意してください。

ophelp

システムプロセッサーで使用可能なイベントとその簡単な説明を表示します。

opimport

サンプルデータベースファイルをシステム用に外部のバイナリー形式からネイティブの形式に変換します。異なるアーキテクチャーからのサンプルデータベースを解析する場合にのみこのオプションを使用してください。

opannotate

アプリケーションがデバッグシンボルでコンパイルされている場合は、実行可能ファイル用の注釈付きのソースを作成します。

opcontrol

プロファイリング実行で収集するデータを設定します。

operf

opcontrol を置き換えることを目的としています。**operf** ツールは **Linux Performance Events** サブシステムを使用するため、**1** つのプロセスまたはシステム全体としてプロファイリングをより正確に対象にし、**OProfile** がシステム上のパフォーマンス監視ハードウェアを使用する他のツールと適切に共存できるようにします。**opcontrol** とは異なり、初期設定は必要ありません。**--system-wide** オプションが使用されていない限り、**root** 権限なしで使用できます。

opreport

プロファイリングデータを取得します。

oprofiled

デーモンとして実行して定期的にサンプルデータをディスクに書き込みます。

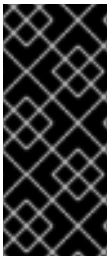
レガシーモード**(opcontrol**、**oprofiled** ツール、および後プロセッシングツール**)**は引き続き利用できますが、推奨されるプロファイリング方法はなくなりました。

これらコマンドの詳細情報については、**OProfile man** ページを参照してください。

```
$ man oprofile
```

A.15. TASKSET

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。



重要

taskset はローカルメモリー割り当てを保証しません。ローカルメモリー割り当てによる追加のパフォーマンス上の利点が必要な場合、**Red Hat** は **taskset** ではなく **numactl** を使用することを推奨します。

実行中のプロセスの **CPU** 親和性を設定するには、以下のコマンドを実行します。

```
# taskset -pc processors pid
```

processors を、コンマ区切りの ***プロセッサー*** 一覧またはプロセッサーの範囲に置き換えます（例：**1,3,5-7**）。***pid*** を再設定するプロセスのプロセス識別子で置き換えます。

特定の親和性のプロセスを開始するには、以下のコマンドを実行します。

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

processors をコンマ区切りのプロセッサーリストまたはプロセッサーの範囲で置き換えます。***application*** を実行するアプリケーションのコマンド、オプション、引数で置き換えます。

taskset の詳細は、の **man** ページを参照してください。

```
$ man taskset
```

A.16. SYSTEMTAP

SystemTap は、独自のガイド**(Red Hat Enterprise Linux 7** バージョンの **SystemTap** ビギナーズガイド および **SystemTap Tapset Reference)**に記載されています。

付録B 改訂履歴

改訂 10.13-59 非同期の更新	Mon May 21 2018	Marek Suchánek
改訂 10.14-00 7.5 GA 公開用ドキュメントの準備。	Fri Apr 6 2018	Marek Suchánek
改訂 10.13-58 新しいセクション: pqos.	Fri Mar 23 2018	Marek Suchánek
改訂 10.13-57 非同期の更新	Wed Feb 28 2018	Marek Suchánek
改訂 10.13-50 7.4 GA 公開用ドキュメントバージョン	Thu Jul 27 2017	Milan Navrátil
改訂 10.13-44 非同期の更新	Tue Dec 13 2016	Milan Navrátil
改訂 10.08-38 7.2 GA リリース向けのバージョン	Wed Nov 11 2015	Jana Heves
改訂 0.3-23 RHEL 7.1 GA 向けにビルド。	Tue Feb 17 2015	Laura Bailey
改訂 0.3-3 RHEL 7.0 GA 用に再構築。	Mon Apr 07 2014	Laura Bailey