# 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: 2020-10-05

# 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が、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お勧めしています。 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お勧めします。

# 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 を構成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が、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お勧めします。

## 本書の対象読者

『パフォーマンスチューニングガイド』は、主に以下の異なる (ただし、重複する点もあります) 2 種類の対象読者のために記述されました。

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一覧についてはナレッジベース https://access.redhat.com/ja/solutions/2819531 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階層で構成されるディレクトリーになります。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、システムのハードウェアおよび実行中のプロセスに関する情報も格納されています。これらファイルのほとんどは読み取り専用ですが、一部のファイル (主に /proc/sys 内のファイル) はユーザーやアプリケーションが操作することで、設定変更をカーネルに伝達できるものもあります。

**proc** ディレクトリー内のファイルの表示および編集に関する詳細は『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。

## 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
: システムのハードウェアおよびソフトウェアの基本情報を表示します。

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
: 現在の CPU 使用時間、メモリーおよび swap 領域の使用量、ネットワークの使用量を表示します。

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一覧、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一覧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
```

### 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** オプションを使って間隔を秒単位で設定することもできます。**sar -i 60** と指定すると sar は CPU の使用量を毎分チェックすることになります。

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。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っ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。電力管理のチューニング効果を確認する際にも使用できます。

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** の出力およびその解釈の仕方については「turbostat」を参照してください。

**turbostat** についての詳細情報は、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ロードを分散する方法について決定を行うのに役に立つようシステムの入出力デバイスロードの監視と報告を行います。**iostat** ツールは **iostat** を最後に実行した時点または起動した時点からのプロセッサーあるい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** 値とその値が多くなる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 値は何を示していますか?」 を参照してください。

## 2.7. IRQBALANCE

**irqbalance** はプロセッサー全体にハードウェア割り込みを分散しシステムのパフォーマンスを向上させるコマンドラインツールです。**irqbalance** の詳細は「irqbalance」または man ページをご覧ください。

```
$ 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。

よく使用する例の一つとして、TCP ソケット、メモリー使用量、ソケットを使用しているプロセスなどに関する詳細情報 (内部情報など) 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 ノード単位でのオペレーティングシステムおよびプロセス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 ヒットとミスのシステム統計をノードごとに表示します。最適なパフォーマンスは高い **numa_hit** 値と低い **numa_miss** 値で示されます。**Numastat** ではシステム内の NUMA ノード全体でシステムおよびプロセスメモリーがどのように分散されているかを表示するコマンドラインオプションも用意されています。

ノードごとの **numastat** 出力を CPU ごとの **top** 出力と相互参照し、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場合に便利です。

**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** ファイルシステム内の情報に定期的にアクセスしてノードごとに使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 (**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るため、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、システムの **/usr/share/doc/oprofile-*version*** に格納されているドキュメントをご覧いただくこともできます。

## 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の実行に伴う各プロセスをユーザー指定のファイル、ファイル記述子、ネットワークソケットなどに明確に識別する説明を出力します。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 を使っ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 パッケージをインストールすると付属ドキュメントが **/usr/share/doc/valgrind-*version*** にインストールされます。

## 2.14. PQOS

intel-cmt-cat パッケージから使用できる **pqos** ユーティリティーを使用すると、最近の Intel プロセッサーでの CPU キャッシュとメモリーの帯域幅を監視および管理することができます。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。

監視

- キャッシュ監視テクノロジー (CMT) を使用したラストレベルキャッシュ (LLC) の使用および競合監視。
- メモリー帯域監視 (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監視。

割り当て

- キャッシュ割り当てテクノロジー (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 領域サイズの制御。
- 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
```

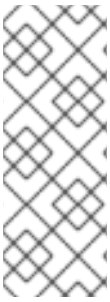
### 関連資料

- **pqos** の使用に関する詳細は、pqos(8) 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、選択されたプロファイルに従ってシステム設定を静的および動的にチューニングするデーモンです。**Tuned** 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複数の定義済みプロファイルとともに配布されます。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で行われたシステム設定の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 mode** で実行できます (常駐メモリーをまったく必要としません)。このモードでは、**tuned** は設定を適用し、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を含むたくさんの **tuned** 機能が使用できないため、**no-daemon mode** はデフォルで無効になります。**no-daemon mode** 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から構成されます。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とログイン時に頻繁に使用されますが、主に 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使う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、異なるタイミングと方法で使用されます。**Tuned** はこうしたコンポーネントの動作を監視して、それら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ー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ースはデフォルト設定のように常に最高速度で動作する必要はありません。**Tuned** には、ネットワークデバイスを監視してチューニングを行うプラグインがあり、ネットワークインターフェースの利用率が低くなると、それを検出して自動的にそのインターフェースの速度を低下させて電力の使用を削減することができます。たとえば、DVD イメージをダウンロードしたり、大きいファイルが添付された電子メールを開いたりしたためインターフェースのアクティビティーが長い時間にわたって増加すると、**tuned** はこれを検出して、アクティビティーレベルが非常に高い間に最良のパフォーマンスを提供できるようインターフェース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向けの他のプラグインにも使用されています。

動的チューニングは Red Hat Enterprise Linux ではグローバルで無効であり、**/etc/tuned/tuned-main.conf** ファイルを編集し、**dynamic_tuning** フラグを **1** に変更することによって有効にできます。

### 3.1.1. プラグイン

**Tuned** では、*監視プラグイン*と*チューニングプラグイン*の 2 つの種類のプラグインを使用します。監視プラグインは、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。現時点では、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ろうとし、パフォーマンスプロファイルで無効になります (個別プラグインの動的チューニングは **tuned** プロファイルで有効または無効にできます)。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クス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により、個別サブシステムがチューニングされ、**tuned** プロファイルから入力される複数のパラメーターが取得されます。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では、個別サブシステムのすべてのデバイスに一致するワイルドカードが使用されます (このプロファイルの変更方法の詳細については、「カスタムプロファイル」を参照)。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るようになります。ユーザーが行うのは、正しい **tuned** プロファイルを選択するだけです。

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小さい場合、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。また、レイテンシーは、特定の値に強制的に設定できます (動的に変更しない場合)。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により実現できます。

**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、一部のノートブックに存在し、Asus Super Hybrid Engine とも呼ばれます。CPU 負荷が ***load_threshold_powersave*** パラメーターで指定された値以下である場合は、プラグインによって FSB 速度が ***she_powersave*** パラメーターで指定された値に設定されます (FSB 周波数および対応する値の詳細については、カーネルのドキュメントを参照)。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は、FSB 速度が ***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、インターフェースの使用状況に応じてインターフェース速度が自動的に変更されます。

**sysctl**

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

なります。ここで、**name** は **sysctl** ツールにより提供されたものと同じ名前になります。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された値に設定します。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**

音声コーデックの 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。現時点では、**snd_hda_intel** と **snd_ac97_codec** がサポートされてます。値が **0** の場合は、autosuspend が無効になります。また、ブール値パラメーター ***reset_controller*** を **true** に設定することにより、コントローラーを強制的にリセットすることもできます。

**disk**

エレベーターを ***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、ASPM を ***aspm*** パラメーターで指定された値、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、ディスク spindown タイムアウトを ***spindown*** パラメーターで指定された値、ディスク readahead を ***readahead*** パラメーターで指定された値に設定し、現在のディスク readahead 値を ***readahead_multiply*** パラメーターで指定された定数で乗算できます。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*** により制御でき、デフォルトで有効になります。

**注記**

異なるディスクの readahead 値を指定する **tuned** プロファイルを適用すると、ディスク先読みの値の設定が **udev** ルールを使用して設定されている場合は、ディスク先読み値の設定が上書きされます。Red Hat では、**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。スクリプトは、**start** または **stop** のいずれかの引数 (スクリプトがプロファイルのロード時またはアンロード時に呼び出されるかによって決まります) によって呼び出されます。スクリプトファイル名は、***script*** パラメーターによって指定できます。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。このような操作を行わないと、ロールバックが機能しません。ユーザーの利便性のために、**functions** Bash ヘルパースクリプトがデフォルトでインストールされ、スクリプトで定義されたさまざまな機能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** 設定を指定します。構文は **name=value** となります。ここで、**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 **default**、**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*** オプションを使用すると、grub2 設定ファイルのカスタマイズされた標準的でない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 で以下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。パフォーマンスの最大化に興味がない場合は、**balanced** または **powersave** プロファイルに変更できます (下記参照)。

**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
```

マシンを起動する度に **tuned** を有効にするには、以下のコマンドを実行します。

```
systemctl enable tuned
```

プロファイルの選択などの他の **tuned** の制御の場合は、以下のコマンドを実行します。

```
tuned-adm
```

このコマンドを使用するには、**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
```

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、ロード中にそれらのプロファイル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** プロファイルでディスクに対して **high** スループットを設定し、同時に **spindown-disk** プロファイルでディスクスピンダウンを **low** 値に設定することが挙げられます。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され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提示するようにするには、以下のコマンドを実行します。

```
tuned-adm recommend
```

**Tuned** 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/** ディレクトリーに格納されます。各プロファイルには独自のディレクトリーがあります。プロファイルは **tuned.conf** という名前の主要設定ファイルとオプションのヘルパースクリプトなどの他のファイルから構成されます。

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

また、**/etc/tuned/** ディレクトリーで独自のプロファイルを作成して、特定のパラメーターのみが調整、または上書きされた **/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]** セクションは 1 つだけ存在します。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、以下のオプションがあります。

**include=*profile***

指定されたプロファイルがインクルードされます。たとえば、**include=powersave** の場合は、**powersave** プロファイルがインクルードされます。

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あり、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=*** を使用したときと同様です。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** で始まるすべての候補に一致し、それらの候補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** と **sda2** を除くすべての候補に一致します。

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

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

**include** オプションを使用して同じセクションが複数回指定された場合は、設定がマージされます。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 **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を持ちます。**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。

以下に、**balanced** プロファイルに基づき、すべてのデバイスの ALPM が最大の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を示します。

```
[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、**tuned-adm** ユーティリティーで簡単にアクティベートできる一般的なユースケース向けの定義済みプロファイルが複数含まれます。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一覧表示して、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だけを表示する場合は、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
```

以下に、基本パッケージでインストールされるプロファイルをリストします。

**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、および動画のプラグインが有効になり、**conservative** ガバナーがアクティブ化されます。***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 ホストアダプター向けの 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省電力がスケジュールされ、**ondemand** ガバナーがアクティブ化されます。さらに、AC97 音声省電力と、システムに応じて HDA-Intel 省電力 (10 秒のタイムアウト) が有効になります。KMS が有効なサポート対象 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。

**注記**

**powersave** プロファイルは、必ずしも最も効率的ではありません。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、一般的に **balanced** プロファイルが優れたオプションになる場合があります。

**throughput-performance**

高スループットに最適化されたサーバープロファイル。省電力メカニズムが無効になり、ディスクとネットワーク IO のスループットパフォーマンスを向上させる sysctl 設定が有効になり、**deadline** スケジューラーに切り替わります。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きます。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きます。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**

enterprise-storage プロファイルに基づく仮想ゲスト向けプロファイル。仮想メモリーの swappiness の減少やディスクの readahead 値の増加などが行われます。ディスクバリアは無効になりません。

**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 の減少、ディスクの readahead 値の増加、ダーティーページのよりアグレッシブな値の有効化などが行われます。

**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ロードされます。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 パッケージにより提供され、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ために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 にパーティションを設定します。分離された CPU でジッターおよび割り込みを減らすために、プロファイルは、ユーザー空間プロセス、移動可能なカーネルスレッド、割り込みハンドラー、およびカーネルタイマーから分離された CPU を消去します。

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** ファイルに **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。

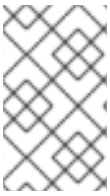
**isolated_cores=*cpu-list***

分離する CPU の一覧を表示します。分離されている CPU の一覧はコンマで区切られているか、ユーザーが範囲を指定できます。範囲は、ハイフンを使用して指定できます (例: **3-5**)。このオプションは必須です。ここに挙げられていない CPU は、自動的にハウスキーピング CPU と見なされます。

**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より考慮されない CPU の一覧を表示します。このオプションは任意です。これは、通常、**isolated_cores** と同じ一覧になります。

**cpu-partitioning** の詳細は、man ページの tuned-profiles-cpu-partitioning(7) 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中で省電力への影響度が最も低く、**tuned** の CPU プラグインとディスクプラグインのみが有効になります。

**desktop-powersave**

デスクトップシステム向けの節電プロファイル。SATA ホストアダプター用の ALPM 節電と **tuned** の CPU、イーサネット、およびディスクの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 ホストアダプター用の ALPM 省電力、WiFi 省電力、および **tuned** の CPU、イーサネット、ディスクのプラグインが有効になります。

**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になります。

**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、USB 自動サスペンド、および Bluetooth が無効になり、Wi-Fi 省電力が有効になり、ログ同期が無効になり、ディスクライトバック時間が増加し、ディスクの swappiness が減少します。すべてのパーティションは、**noatime** オプションを使用して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がアクティブ化され、readahead 設定値が乗算され、ルートパーティションと起動パーティション以外のパーティションでバリアが無効になります。

**注記**

物理マシンで **atomic-host** プロファイル、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 向けに **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 で以下のコマンドを実行します。

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

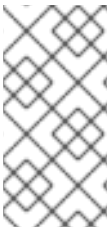
Red Hat Enterprise Linux Atomic Host 向けの **tuned** プロファイルには次の 2 つがあります。

**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)。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。



**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。カーネルコマンドラインの編集を可能にする 3 つの **tuned** プロファイル (**realtime**、**realtime-virtual-host**、および **realtime-virtual-guest**) が利用可能です。

**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** プロファイルと **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 **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、以下のコマンドを root で実行します。

-

```
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** プロファイルに基いて **/etc/tuned** ディレクトリー内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。これらのチューニングを有効にするには、**/etc/tuned/*profile*/tuned.conf** で興味があるチューニングをコメント解除します。**--enable** または **-e** オプションを使用して、有効な **PowerTOP** により提示された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ません。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 new_profile_name
```

**powertop2tuned** がサポートするオプションの完全な一覧を表示するには、以下のコマンドを実行します。

```
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負荷でのオペレーティングシステムのパフォーマンスを向上させることができます。**tuned-adm** コマンドラインツールを使用すると、異なるチューニングプロファイルをユーザーが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利用できる事前定義済みのプロファイルがいくつか用意されていますが、**tuned** を使用すると、カスタムのプロファイルを定義することもできます。事前に定義されているプロファイルをベースとしてカスタマイズすることも新規に作成することもできます。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 power-saving プロファイルと performance-boosting プロファイルの 2 種類のカテゴリーに分かれます。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処理能力
- 仮想マシンのパフォーマンス
- 仮想化ホストのパフォーマンス

### tuned ブートローダープラグイン

**tuned Bootloader plug-in** を使用すると、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たり、**tuned** を手動で停止すると、追加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、パラメーターは **grub.cfg** ファイルで永続化されます。

### 環境変数と展開 tuned 組み込み関数

GRUB 2 設定を更新したあとに **tuned-adm profile *profile_name*** と **grub2-mkconfig -o *profile_path*** を実行すると、Bash 環境変数を使用できます。Bash 環境変数は **grub2-mkconfig** の実行後に展開されます。たとえば、次の環境変数は **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。次の例では、**${isolated_cores}** は **1,2** に展開され、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}
```

次の例では、**${non_isolated_cores}** は **0,3-5** に展開され、**cpulist_invert** 組み込み関数が **0,3-5** 引数で呼び出されます。

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

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

**cpulist_invert** 関数は CPU の一覧を反転します。6 CPU のマシンでは、反転が **1,2** になり、カーネルは **isolcpus=1,2** コマンドラインパラメーターで起動されます。

**tuned** 環境変数を使用すると、必要な入力作業が減少します。また、**tuned** 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。組み込み関数で満足できない場合は、Python でカスタム関数を作成してプラグインという形で **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展開されます。

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

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

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

**isolated_cores=1,2** を **/etc/tuned/*my-variables.conf*** ファイルに追加する場合、カーネルは **isolcpus=1,2** パラメーターで起動されます。

### 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。1 つは新しい **tuned** プロファイルディレクトリーを作成する方法であり、もう 1 つは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する方法です。

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名前の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、**tuned.conf** という名前のファイルを作成し、次の行を先頭に含めます。

```
[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** の値がデフォルトの 10 ではなく 5 に設定された状態で、**throughput-performance** プロファイルから設定を使用するには、次の行を含めます。

```
[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** ファイルとともに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すべてのプロファイルの変更を保持できます。

または、システムプロファイルとともにディレクトリーを **/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ロードされることに注意してください。この方法の欠点は、**tuned** のアップグレード後にシステムプロファイルが更新される場合に、古くなった変更済みバージョンに変更が反映されなくなることです。

## リソース

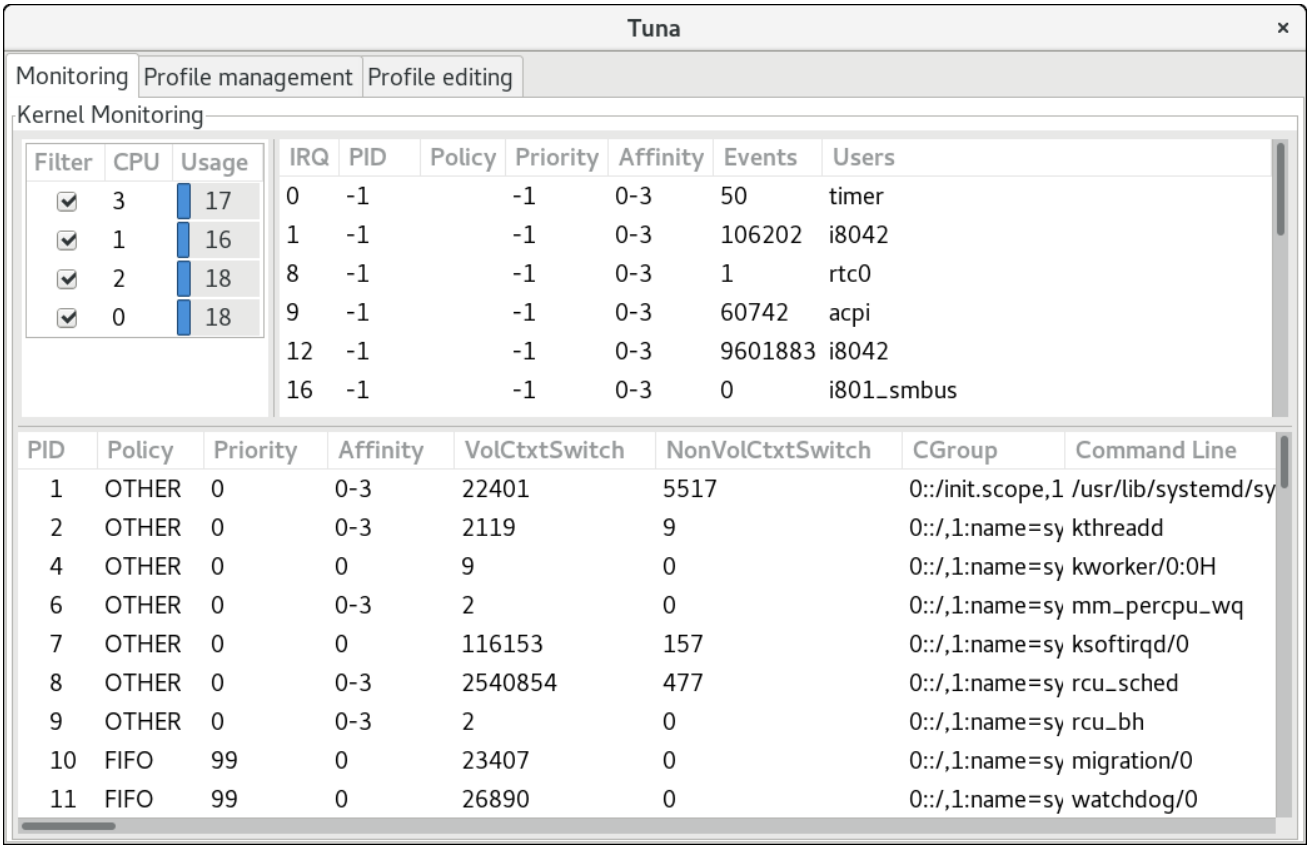
詳細については、「tuned」と「tuned-adm」を参照してください。**tuned** および **tuned-adm** 使用の詳細については、tuned(8) および tuned-adm(1) man ページを参照してください。

# 第4章 TUNA

**Tuna** ツールを使用すると、スケジューラーのチューニング可能機能、チューニングスレッドの優先度、および IRQ ハンドラーを調整し、CPU コアおよびソケットを分離できます。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 パッケージのインストール後、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ース (GUI) を起動します。**tuna -h** コマンドを使用して、使用できるコマンドラインインターフェース (CLI) の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*** コマンドと記述ファイル名を使用して、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) と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 をリストするには、Tuna GUI の Monitoring タブを確認するか、**/proc/cpuinfo** ファイルの詳細情報を確認します。

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 --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* は CPU 番号のコンマ区切りリストです (例: **--cpus=0,2**)。

## 4.3. TUNA を使用した IRQ のチューニング

現在システムで実行している IRQ のリストを確認するには、Tuna GUI の Monitoring タブまたは **/proc/interrpupts** ファイルを確認します。**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 --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* は CPU 番号のコンマ区切りリストです (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対象とし、2 つの CPU に分散するには、以下を行います。

```
# 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。
- ラウンドロビン形式の場合は、*policy* を **RR** に設定します。先入れ先出しの場合は **FIFO**、デフォルトのポリシーの場合は **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。

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ています。すべての **ssh** スレッドを CPU 0 および 1 で実行し、すべての **http** スレッドを CPU 2 および 3 で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、数字 (0 と 1)、16 進法マスク (**0x3**)、またはバイナリー (**11**) で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、数字 (2 と 3)、16 進法マスク (**0xC**)、またはバイナリー (**1100**) で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 では、システムレベルでのパフォーマンス測定値の監視、可視化、および分析を行う一連のツール、サービス、およびライブラリーである **Performance Co-Pilot** (**PCP**) がサポートされます。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 インターフェー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 インターフェー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および管理。
- 履歴データのログおよび取得。

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ます。**pminfo** や **pmstat** などのクライアントツールを使用すると、同じホストまたはネットワークを介してこのデータを取得、表示、アーカイブ、および処理できます。pcp パッケージはコマンドラインツールと基礎となる機能を提供します。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。

### リソース

- **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一覧、利用可能な設定オプションの説明、および関連する man ページの一覧が提供されます。デフォルトでは、包括的なドキュメンテーションは **/usr/share/doc/pcp-doc/** ディレクトリーにインストールされます (『Performance Co-Pilot User's and Administrator's Guide』や『Performance Co-Pilot Programmer's Guide』など)。
- 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」 (よくある質問の一覧) 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

PCP 設定の読み出しに XFS PMDA が記載されていない場合は、PMDA エージェントを手動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** ロールを指定すると、現在のシステムでパフォーマンスメトリックを収集できます。
- **monitor** ロールを指定すると、システムでローカルシステムまたはリモートシステム、あるいはローカルおよびリモートシステムの両方を監視できます。

デフォルトのオプションは、コレクターとモニターの両方である **both** で、XFS PMDA はほとんどの場合で適切に操作することができます。

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実行します。

```
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 をインストールし、XFS PMDA を有効にしたら、「XFS PMDA をインストールして PCP で XFS データを収集」 の手順を使用できます。PCP と XFS の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、利用可能なパフォーマンスメトリックに関する情報を表示する **pminfo** ツールを使用することです。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、各グループがルート XFS メトリックからの新しいリーフノードとなるようにする他の **XFS** メトリックのグループ があります。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 行で表示するには、以下を実行します。

```
# pminfo -t xfs.write_bytes

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

**xfs.write_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、以下を実行します。

```
# 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** メトリックスに対応するパフォーマンスの現在の読み取り値を取得します。

```
# pminfo -f xfs.read_bytes

xfs.read_bytes
    value 4891346238
```

### pmstore でのメトリックの設定

PCP を使用すると、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) に特定のメトリックの値を編集できます。メトリックの値を編集するには、**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、マウントされたファイルシステムの問題を特定し、パフォーマンスを評価することが簡単になります。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上で選択したメトリックのアーカイブされたログを作成します。

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。設定ファイルはプライマリーロギングインスタンスによってログに記録されるメトリックを指定します。

**pmlogger** で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、pmlogger の行が 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** サービスが実行されているとき、PCP はホスト上でメトリックのデフォルトセットをログに記録します。**pmlogconf** ユーティリティーを使用してデフォルト設定をチェックし、必要の応じて XFS ロギンググループを有効にすることができます。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** のプロンプトの従って、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し、有効な各グループのログ間隔を制御します。グループを選択するには、プロンプトの応答として **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編集し、一定の間隔の特定のメトリックを追加すると、必要に合ったロギング設定を作成することができます。

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**、または **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します。

- 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、グラフを使っ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、アーカイブに収集された **xfs.perdev.log** メトリックのデータを 5 秒間隔で表示し、すべてのヘッダーを表示する場合は以下を実行します。

```
$ 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。

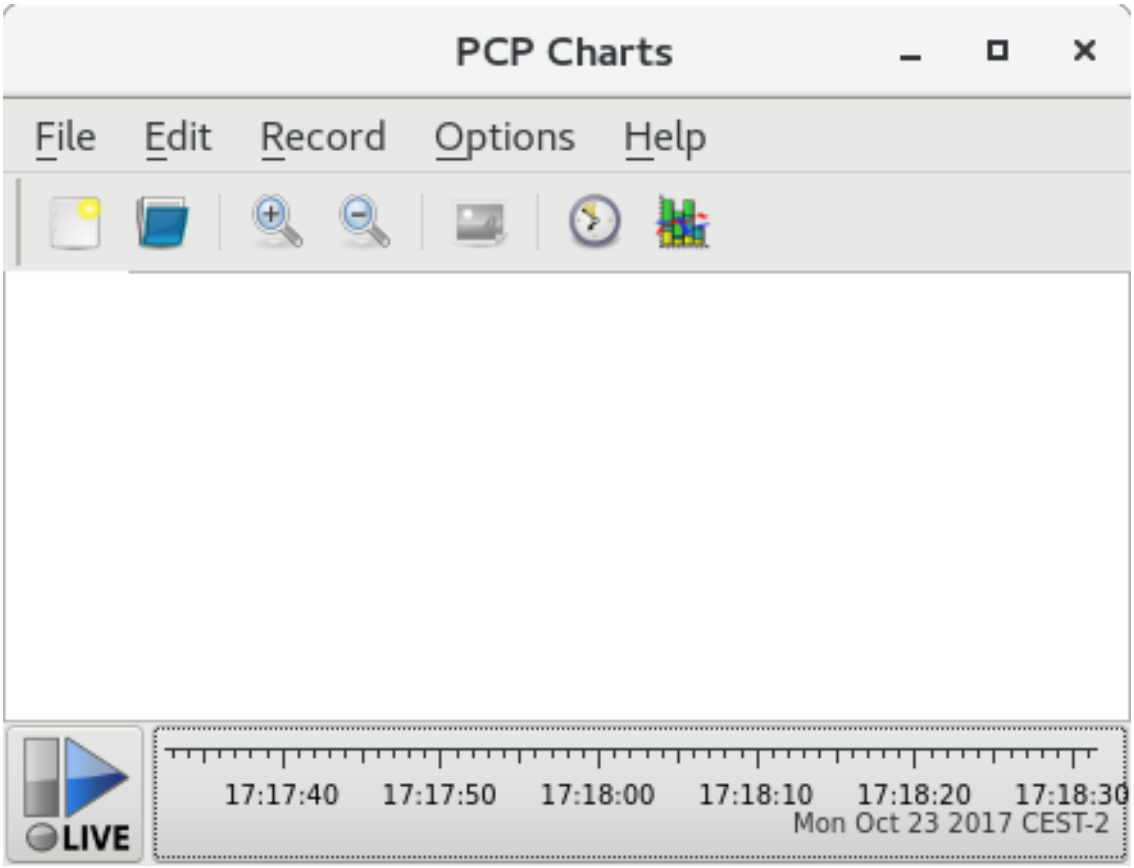
### 5.2.5. PCP Charts での視覚追跡

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 1 つ以上のライブホストから提供され、PCP ログアーカイブから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があります。PCP Charts をコマンドラインから実行するには、**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** 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、ローカルマシンとリモートマシンのホスト名またはアドレスを指定して、それらのローカルおよびリモートマシン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。詳細設定オプションには、チャートの軸の値を手動設定する機能や、色を手動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** をクリックして記録を停止します。記録の停止後、記録されたメトリックは後で閲覧するためにアーカイブ化されます。

PCP Charts インターフェー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、**view** 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することができます。このメタデータは、使用されるメトリックやチャートの列など、チャートのすべての側面を表します。カスタム **view** 設定を作成し、**File → Save View** をクリック

して保存して、**view** 設定を後でロードすることができます。**view** 設定ファイルとそれらの構文に関する詳細は、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 の view 設定ファイルの例では、XFS ファイルシステム **loop1** に対して読み書きされた合計バイト数を表す積み上げチャートグラフを示します。

```
#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。

**pmlogger** 出力の **tar.gz** アーカイブは、PCP Charts などの PCP ツールを使用して分析でき、他のソースのパフォーマンス情報と比較できます。

1. pcp パッケージをインストールします。

```
# yum install pcp
```

2. **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
```

3. **pmlogger** サービスを開始します。

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

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

4. XFS ファイルシステム上で操作を実行します。

5. **pmlogger** サービスを停止します。

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

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

6. 出力を収集し、ホスト名と現在の日時が名前に使用される **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、「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

この最初の項を読むと、以下のような要因がシステムおよびアプリケーションのパフォーマンスに与える影響について理解できます。

- プロセッサー同士およびメモリーなど関連リソースとの接続形態
- 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。

**Symmetric Multi-Processor (SMP) トポロジー**

SMP トポロジーの場合、すべてのプロセッサーが同じ時間量メモリーにアクセスすることができます。ただし、メモリーアクセスへの平等な共有は本質的に全 CPU からのメモリーアクセスを直列化することになるため、SMP システムでのスケーリングにおける制約が全般的に許容できないレベルと見られるようになってきました。こうした状況のため実際には最近のサーバーシステムではすべて NUMA マシンが使われています。

**Non-Uniform Memory Access (NUMA) トポロジー**

NUMA トポロジーは SMP トポロジーより後に開発されました。NUMA システムでは複数のプロセッサーがひとつのソケット上で物理的にまとめられています。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の場合、プロセッサー、メモリー、周辺デバイスの接続形態に関する情報は/**sys** ファイルシステムに格納されています。/**sys**/**devices**/**system**/**cpu** ディレクトリーにはプロセッサー同士の接続形態に関する詳細が格納されています。/**sys**/**devices**/**system**/**node** ディレクトリーには NUMA ノードおよびノード間の相対距離に関する情報が格納されています。

#### 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が表示されます。

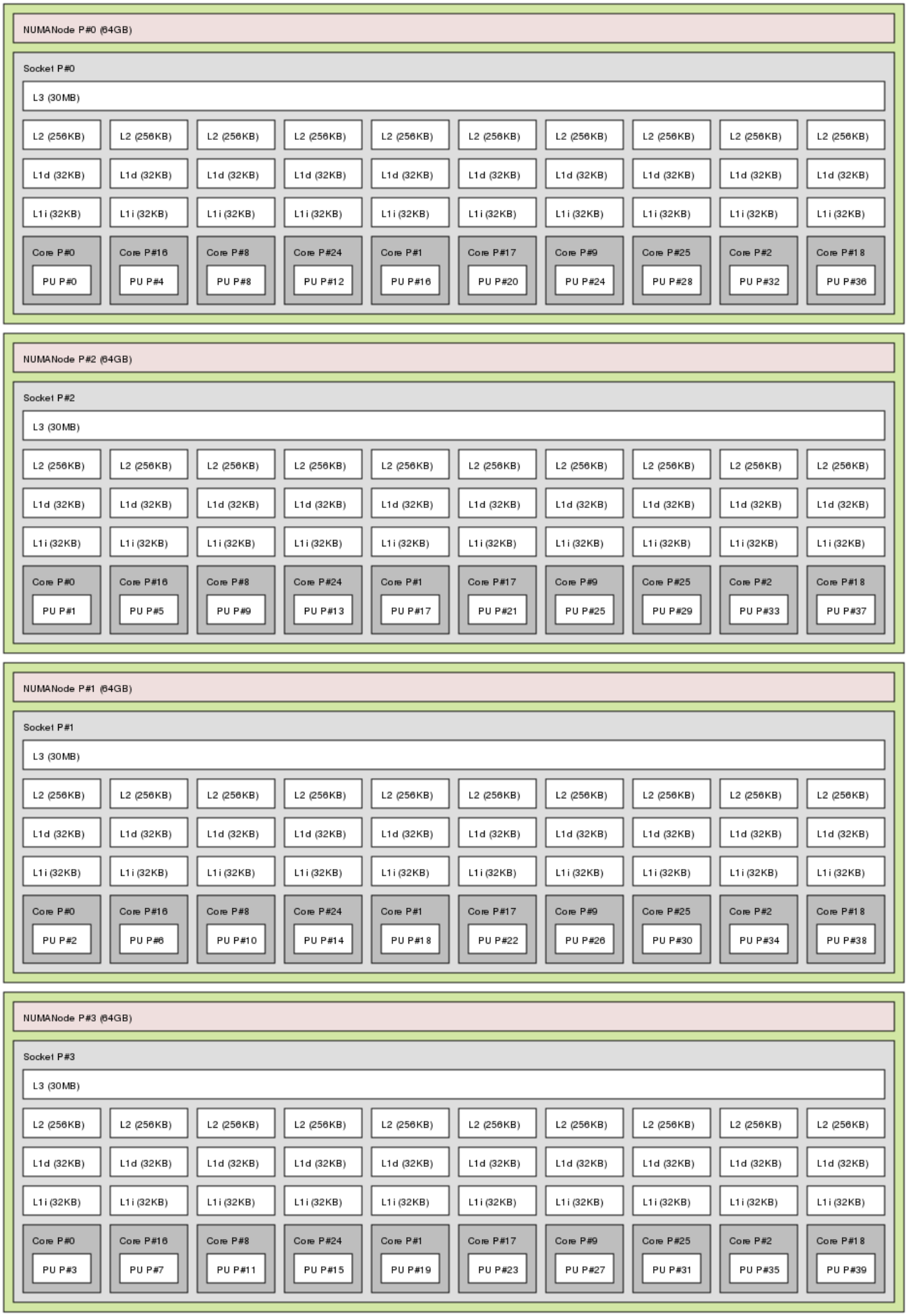
```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
node 1 cpus: 2 6 10 14 18 22 26 30 34 38
node 1 size: 65536 MB
node 1 free: 44321 MB
node 2 cpus: 1 5 9 13 17 21 25 29 33 37
node 2 size: 65536 MB
node 2 free: 44304 MB
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 のアーキテクチャーに関する情報を収集して表示します。

```
$ 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ます。システムをグラフィカルなイメージで表示します。**lstopo-no-graphics** コマンドを使用するとテキスト形式の出力になります。




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。システムのスケジューラーはスレッドを実行させるプロセッサーとそ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の場合、設計者または管理者側でスレッドを実行させる場所を決定した方がよい場合がよくあります。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い、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。オプションは ***nohz_full*** カーネルパラメーターを使用すると指定したコアで有効にすることができます。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

割り込み要求つまり IRQ とはハードウェアの一部からプロセッサーに早急な対応を求める信号を指します。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一つのバッチ (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 を使った 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って説明していきます。

### 6.2.1. turbostat

**Turbostat** は指定した間隔でカウンターの結果を出力するため、過剰な電力消費やディープスリープ状態に入れない問題、システム管理割り込み (SMI) が必要に作成される問題などサーバーでの予期しない動作を特定する場合に役立ちます。

**turbostat** ツールは kernel-tools パッケージの一部になります。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** ツールはプロセッサーやオペレーティングシステムのメモリー統計を NUMA ノード単位で表示し、プロセスのメモリーがシステム全体で分散されているのか、特定ノードに集中しているのかを表示します。

**numastat** 出力をプロセッサーごとの **top** 出力と相互参照し、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** 出力の詳細については 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数が記載されています。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一覧などになります。

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できる **pqos** ユーティリティーを使用すると、C最近の Intel プロセッサーで CPU キャッシュとメモリー帯域を監視することができます。

**pqos** ユーティリティーは、**top** utility. It monitors: と似たキャッシュおよびメモリー監視ツールを提供し、以下を監視します。

- IPC (サイクルごとの命令)。
- 最終レベルキャッシュミスの数。
- LLC で占有する 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 (MBL)。
- リモートメモリーへの帯域 (MBR)。

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。

**関連資料**

- **pqos** ユーティリティーと関連するプロセッサー機能の一般概要は、「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 でそのツールを使っ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。

動的ティックレスの動作を特定のコアで有効にするにはカーネルコマンドで ***nohz_full*** パラメーターを使ってそ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)。この動作は起動時に一時的に有効にすることも **/etc/default/grub** ファイルで永続的に有効にすることもできます。永続的に有効にする場合は **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*** パラメーターを使って特定のコアをユーザー領域タスクから隔離します。

- 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** 等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ではビジーな CPU で 1000 ティックになります。

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

動的テイクレスカーネルを設定すると 1 ティックになります。

```
# 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パフォーマンスと電力消費効率のバランスを指定することができます。パフォーマンスまたは電力消費効率どちらをどの程度犠牲にするかのオプションを選択すると、機能に対応するプロセッサーを動作させるためこの情報が使用されます。

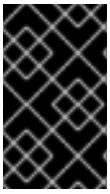
デフォルトではツールは全プロセッサー上で **performance** モードで動作します。プロセッサーのサポートを必要とします。**CPUID.06H.ECX.bit3** の表示があればサポートされています。実行する場合は root 権限で行ってください。

**x86_energy_perf_policy** は kernel-tools パッケージで提供されます。**x86_energy_perf_policy** の使い方については「x86_energy_perf_policy」を参照するか man ページをご覧ください。

```
$ 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** を使用すると実行中プロセスのプロセッサー親和性を読み出して設定したり、指定プロセッサー親和性でプロセスを起動することができるようになります。



**重要**

**taskset** はローカルの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りパフォーマンスを向上させる必要がある場合は **taskset** ではなく **numactl** を使用することを推奨しています。

**taskset** の詳細については「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

**numactl** を使用すると指定したスケジュールまたはメモリー配置ポリシーでプロセスを実行することができます。**Numactl** は共有メモリーセグメントやファイルに永続的なポリシーを設定したり、プロセスのプロセッサー親和性やメモリー親和性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。

マルチスレッド化したアプリケーションでパフォーマンス重視となるものは特定のプロッセッサーではなく特定の NUMA ノードで実行するよう設定すると効果を得られることがあります。この方法が適切かどうかはご使用のシステムおよび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データにアクセスする場合はスレッドが同じプロセッサーで実行されるよう設定すると良いかもしれません。ただし、別々のデータにアクセスしてキャッシュを行う複数のスレッドが同じプロセッサーで実行される場合、各スレッドは前のスレッドでアクセスされたキャッシュデータ消去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。キャッシュミスが過剰に発生しているかどうか確認する場合は「perf」の説明に従って **perf** ツールを使用すると行うことができます。

**Numactl** はプロセッサーとメモリー親和性を管理する場合に役立つオプションがいくつか用意されています。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** ライブラリーが収納され、カーネル対応の NUMA ポリシーに対するシンプルなプログラミングインターフェースを提供しています。**numactl** アプリケーションに比べより高度な調整を行うことができます。詳細については 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** は自動で NUMA の親和性を管理するデーモンです。NUMA トポロジーとシステム内のリソース使用を監視して NUMA によるリソース管理と管理を動的に改善します。

また、**numad** では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されています。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割り込みなしで完了しなければならない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指定タイムスライスが経過した後もリソースへのアクセスを中断する必要はありません。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でもノーマルポリシーのスレッドより先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 を使っ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) はリアルタイムポリシーで各スレッドに固定の優先度を指定します。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** 全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順位が高いスレッドをスケジュールに入れます。**SCHED_FIFO** スレッドの優先レベルは 1 から 99 で指定することができ、99 が最も高い優先度になります。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。プロセッサーがひとつのシステムの場合は最終的にハングすることになります。

管理者側で **SCHED_FIFO** の帯域幅を制限してリアルタイムのアプリケーションプログラマーがプロセッサーを独占するリアルタイムのタスクを開始しないよう阻止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** µs もしくは 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** µs もしくは 0.95 秒です。

#### 6.3.6.1.2. SCHED_RR を使っ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になります。複数のスレッドを同じ優先レベルで実行しなければならない場合に便利です。

**SCHED_FIFO** と同様、**SCHED_RR** は各スレッドに固定の優先度を指定するリアルタイムポリシーになります。スケジューラーは **SCHED_RR** 全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順位が高いスレッドをスケジュールに入れます。ただし、**SCHED_FIFO** とは異なり複数のスレッドが同じ優先度を持つ場合は特定の時間内でラウンドロビン方式にスケジュールが行われます。

この時間枠は ***sched_rr_timeslice_ms*** カーネルパラメーターを使って秒単位で設定することができます (**/proc/sys/kernel/sched_rr_timeslice_ms**)。最も小さい値は 1 ミリ秒になります。

#### 6.3.6.1.3. SCHED_OTHER を使った通常のスケジュール

Red Hat Enterprise Linux 7 では **SCHED_OTHER** がデフォルトのスケジューリングポリシーになります。CFS (Completely Fair Scheduler) を使ってこのポリシーでスケジュールされているスレッドすべてに対してプロセッサーへの公平なアクセスを提供します。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。

このポリシーを使用す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*** 起動パラメーターを使用するとスケジューラーから CPU を切り離すことができます。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 コマンドを使ってその 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
```

また CPU の分離は **Tuna** ツールを使っても行うことができます。**Tuna** を使用すると起動時だけでなく

いつでも CPU の分離を行うことができます。この方法は ***isolcpus*** パラメーターを使用する場合とは若干異なり、***isolcpus*** で得られるようなパフォーマンスの改善は現時点では期待できません。ツールの詳細については「Tuna を使った 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

割り込み要求には関連づけられた親和性プロパティー **smp_affinity** があり、割り込み要求を処理するプロセッサーはこのプロパティーによって指定されます。アプリケーションのパフォーマンスを改善す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複数のプロセッサーに割り当てます。これにより指定し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の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** ドライバーを使用したデバイス。
   - 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。

   - このような場合には、以下の手順に従って割り込みの配信を設定できます。

     さらに、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ます。BIOS の中には、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。どの CPU を使用していても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。

   非物理フラットモード (**flat**) のみが複数の CPU への割り込みの分散に対応します。このモードは、9 個以上の CPU を持つシステムでは利用できません。

   ```
   $ 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する場合は、**Setting APIC routing to physical flat** と同様の行を確認できます。
- このようなメッセージが表示されない場合、システムは **flat** モードを使用します。

システムが **x2apic** モードを使用する場合は、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これを無効にできます。

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 進数のビットマスクとして保存されます。各ビットは別の CPU を設定します。最下位ビットは CPU 0 です。

マスクのデフォルト値は **f** で、割り込み要求がシステムのどのプロセッサーでも処理可能であることを意味します。この値を **1** に設定すると、プロセッサーの 0 のみが割り込みを処理できることになります。

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'
```

32 個以上のプロセッサーが搭載されているシステムでは、32 ビットグループにそれぞれ **smp_affinity** 値を区切って設定する必要があります。たとえば、64 個のプロセッサーを持つシステムで最初の 32 個のプロセッサーのみが割り込み要求を処理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する **/proc/irq/*irq_number*/smp_affinity** ファイルに格納されます。

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
```

**関連資料**

- 割り込みステアリングに対応するシステムでは、割り込み要求の **smp_affinity** プロパティーを修正するとハードウェアが設定され、カーネルを介入させることなくハードウェアレベルで特定プロセッサーに割り込みを処理させる決定が行われるようになります。

  割り込みステアリングの詳細は、9章 *ネットワーク*を参照してください。

### 6.3.8. Tuna を使った 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、および割り込み親和性を制御できます。また、制御できる各種のエンティティーに複数のアクションも提供します。**Tuna** に関する情報は 4章*Tuna* を参照してください。

# 第7章 メモリー

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、「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 大規模ページの設定」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いページサイズを使用することにより恩恵を受けることができます。たとえば、数百メガバイトまたは数十ギガバイトの大きな固定データセットを使用するアプリケーションでは、4 KB ページを使用した場合にパフォーマンスの問題が発生することがあります。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** とも呼ばれます) と **Transparent Huge Page** 機能の 2 つの異なる大規模ページ機能を使用できます。

### 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 でキャッシュできるアドレスマッピング数は限られています。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の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、そして大量のメモリーを必要とするアプリケーションの関係は最小限のメモリーで動作するアプリケーションに比べ TLB ミスによるパオーマンス低下を被りやすくなります。このためできるだけ TLB ミスが起こらないようにすることが重要になります。

Both HugeTLB および Transparent Huge Page 機能を使用すると、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。また、TLB に格納されたアドレスがより大きなメモリーを参照できるため、TLB のミスが少なくなり、アプリケーション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って説明していきます。

### 7.2.1. vmstat を使っ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、CPU アクティビティーに関する報告を出力します。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 を使っ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** はユーザー領域のバイナリーに対するインストルメンテーションを提供するフレームワークになります。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している **valgrind** ツールは、初期化されていないメモリーの使用、不適切なメモリー割り当て、割り当て解除などのメモリーに関するエラーの検出に役立ちます。

**valgrind** や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 を使ったメモリー使用量のプロファイリング

**Memcheck** は **valgrind** のデフォルトツールです。検出や診断が難しい次のようなメモリーエラーを検出、報告します。

- 発生してはいけないメモリーアクセス
- 未定義または未初期化の値の使用
- ヒープメモリーの不正な解放
- ポインターの重複
- メモリーリーク

**注記**

**Memcheck** が行うのはエラーの報告だけでありそ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** により違反が発生する直前にエラーメッセージがログ記録されるようになります。

memcheck はインストルメンテーションを使用するため memcheck を付けて実行されるアプリケーションの実行速度は通常に比べ 10 倍から 30 倍ほど遅くなります。

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

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

次のオプションを使用すると特定の問題タイプの memcheck 出力に集中させることもできます。

--leak-check

アプリケーションの実行が完了すると memcheck によりメモリーリークの検索が行われます。デフォルト値は検出したメモリーリーク数を出力する **--leak-check=summary** になります。**--leak-check=yes** や **--leak-check=full** を指定するとリークそれぞれの詳細を出力させることができます。無効にする場合は **--leak-check=no** を指定します。

--undef-value-errors

デフォルト値は未定義の値が使用された場合にエラーを報告する **--undef-value-errors=yes** です。**--undef-value-errors=no** を指定するとこの報告を無効にしてメモリーチェックの速度を若干早めることができます。

--ignore-ranges

**--ignore-ranges=0xPP-0xQQ,0xRR-0xSS** などのようにメモリーアクセスを確認する際 memcheck に無視させる範囲を指定します。

memcheck オプションの全一覧は **/usr/share/doc/valgrind-version/valgrind_manual.pdf** に収納されているドキュメントをご覧ください。

#### 7.2.2.2. Cachegrind を使っ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を使ってアプリケーションのやりとりをシュミレーションします。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。このため、Cachegrind で実行されたアプリケーションは通常の実行に比べて 20 倍 から 100 倍の時間がかかります。

アプリケーション実行中の統計を収集しコンソールに要約を出力します。cachegrind をアプリケーションで実行する場合は次のコマンドを実行します。

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

cachegrind の出力を特定の問題に集中させる場合は次のオプションを使用することもできます。

--I1

**--I1=*size*,*associativity*,*line_size*** などのように第 1 レベルの命令キャッシュのサイズ、結合性、行サイズを指定します。

--D1

**--D1=*size*,*associativity*,*line_size*** などのように第 1 レベルのデータキャッシュのサイズ、結合性、行サイズを指定します。

--LL

**--LL=*size*,*associativity*,*line_size*** などのように最後のレベルのキャッシュのサイズ、結合性、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。デフォルトでは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。デフォルトでは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 はプロファイリングの詳細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* はプロセスの識別子になります。この詳細情報は付随する cg_annotateツールでさらに以下のように処理することができます。

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

また、Cachegrind ではコード変更の前と後のプログラムパフォーマンスをより簡単に図表にすることができる cg_diff ツールも用意しています。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
```

結果の出力ファイルの詳細は cg_annotate ツールで表示させることができます。

cachegrind のオプション全一覧は **/usr/share/doc/valgrind-version/valgrind_manual.pdf** に収納されているドキュメントを参照してください。

### 7.2.2.3. Massif を使ったヒープとスタックの領域プロファイリング

Massif は指定アプリケーションが使用す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を抑え実行速度を高めながらアプリケーションが swap 領域を使い切ってしまう可能性を低減する方法を理解する場合に役立ちます。massif を付けてアプリケーションを実行すると実行速度が通常より約 20 倍遅くなります。

アプリケーションで massif を実行するには次のコマンドを実行します。

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

次のオプションを使用すると massif の出力を特定の問題に集中させることもできます。

--heap

massif にヒープのプロファイリングを行わせるかどうかを指定します。デフォルト値は **--heap=yes** です。**--heap=no** に設定するとヒープのプロファイリングが無効になります。

--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 にスタックのプロファイリングを行わせるかどうかを指定します。スタックのプロファイリングにより massif の動作がかなり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場合は **--stack=yes** に設定します。プロファイリングする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わかりやすく示すため massif はメインスタックの開始時のサイズはゼロであると仮定している点に注意してください。

--time-unit

massif で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** (命令を実行) です。**ms** (ミリ秒数またはリアルタイム) や **B** (ヒープおよびスタックで割り当てられたバイト数または割り当て解除されたバイト数) 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。*pid* は指定アプリケーションのプロセス識別子になります。ms_print ツールはこのプロファイリングデータを図式化しアプリケーション実行中のメモリー消費量を表示すると共に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*** ファイルのデータを図式化するには次のコマンドを実行します。

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

Massif のオプション全一覧は **/usr/share/doc/valgrind-version/valgrind_manual.pdf** に収納されているドキュメントを参照してください。

## 7.3. HUGETLB 大規模ページの設定

Red Hat Enterprise Linux 7.1 以降、大規模ページを予約するには起動時と実行時の 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 ノードごとに大規模ページを予約できます。実行時の予約が起動プロセスでできるだけ早いタイミングで行われる場合は、メモリーの断片化の可能性が低くなります。

### 7.3.1. 起動時の大規模ページの設定

起動時に大規模ページを設定するには、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

起動時にカーネルで設定する永続大規模ページ数を定義します。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大規模ページ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、**/proc/sys/vm/nr_hugepages** ファイルの値を変更することで調整可能です。

NUMA システムの場合、このパラメーターで割り当てた大規模ページはノード間で平等に分割されます。実行中、大規模ページを特定ノードに割り当てる場合はそのノードの **/sys/devices/system/node/*node_id*/hugepages/hugepages-1048576kB/nr_hugepages** ファイルの値を変更します。

詳細についてはデフォルトで **/usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt** にインストールされるカーネル関連ドキュメントをお読みください。

**hugepagesz**

起動時にカーネルで設定する永続大規模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**

起動時にカーネルで設定する永続大規模ページのデフォルト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と値のリスト」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 の大規模ページと 1 GB の巨大ページがサポートされます。

1. **/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 root として追加して、1 GB ページ向けの 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. **/usr/lib/systemd/system/hugetlb-gigantic-pages.service** という名前のファイルを次の内容で作成します。

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
DefaultDependencies=no
Before=dev-hugepages.mount
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/lib/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

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

```
#!/bin/sh
```

```
nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
 echo "ERROR: $nodes_path does not exist"
 exit 1
fi

reserve_pages()
{
 echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages 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** 上に 2 つの 1GB ページを予約し、 **node1** 上に 1 つの 1GB ページを予約するには、最後の行を以下に置き換えます。

```
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. 初期時のブート予約を有効にします。

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

**注記**

任意のタイミングで **nr_hugepages** に書き込みを行うことにより、1GB を超えるページを実行時に予約できます。ただし、このような予約はメモリーの断片化により失敗することがあります。起動初期時に実行されるこのスクリプトを使用するのが最も信頼できる 1GB ページの予約方法になります。

### 7.3.2. 実行時の大規模ページの設定

次のパラメーターを使用して実行時の大規模ページ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大規模ページ数を定義します。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 の大規模ページを 20 ページ **node2** に割り当てています。

```
# 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**

メモリーのオーバーコミット中、システムで作成、使用が可能な追加の大規模ページ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、永続大規模ページ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大規模ページ数が取得されます。この余剰大規模ページ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大規模ページがプロセスに割り当てられるため、大規模ページ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、カーネルは大規模ページを任意のプロセスに割り当てようとします (大規模ページ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な場合、カーネルは **madvise()** システムコールで指定された個別プロセスのメモリー領域のみに大規模ページを割り当てます。

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。透過的な大規模ページ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
```

透過的な大規模ページを有効にするには、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、大規模ページ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大規模ページを有効にします。

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

透過的な大規模ページを無効にするには、次のコマンドを実行します。

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

場合によっては、存続期間が長い割り当てで最良のパフォーマンスをすぐに実現するよりも存続期間が短い割り当てに低レイテンシーを提供する方が優先度が高いことがあります。このような場合は、THP を有効にしたままで **direct compaction** を **disabled** にします。

直接圧縮は、大規模ページの割り当て時に行われる同期的なメモリー圧縮です。直接圧縮を無効にすると、メモリーの節約は保証されませんが、ページフォールトが頻繁に発生するときにレイテンシーが高くなる可能性が低くなります。THP により負荷が大幅に改善される場合は、パフォーマンスが低下すること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
```

透過的な大規模ページ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該当ファイルの値を変更し、テストとして一時的に設定することができます。そのパラメーターで使用環境に適したパフォーマンスが得られることを確認したら **sysctl** コマンドを使って今度は永続的にパラメーターを設定します。

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** で **sysctl vm.overcommit_memory=1** を追加し、次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お勧めします。

### 7.5.1. 仮想メモリーのパラメーター

本セクションのパラメーターは特に記載のない限り **/proc/sys/vm** にあります。

**dirty_ratio**

パーセント値です。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** % です。

**dirty_background_ratio**

パーセント値です。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** % 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要求されるメモリー量が大き過ぎるため失敗する要求数を推測し経験則的なメモリーオーバーコミット処理を行います。ただし、正確なアルゴリズムではなく経験則を使ってメモリーの割り当てを行うため、この設定の場合はメモリーがオーバーロードする可能性があります。

このパラメーターを **1** に設定するとカーネルはメモリーのオーバーコミット処理を行いません。このためメモリーのオーバーロードの可能性が高くなりますが、メモリー集約型のタスクパフォーマンスは向上します。

**2** に設定するとカーネルは利用可能な swap 領域と ***overcommit_ratio*** で指定されている物理 RAM の割合との合計と同等もしくはそれ以上のメモリー要求は拒否します。メモリーのオーバーコミットに対するリスクは低減しますが swap 領域が物理メモリーより大きいシステムにしか推奨していません。

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

プロセスで使用可能なメモリーマップ領域の最大数を定義します。ほとんどの場合デフォルト値の **65530** が適した値になります。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。これを使って各低メモリ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。

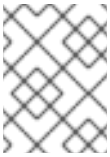
**警告**

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*** の設定が低すぎると、システムによるメモリの回収が妨げられます。これによりシステムがハングし、メモリ不足によってプロセスが強制終了される可能性があります。ただし、***min_free_kbytes*** の設定が高すぎると (システムメモリー合計の 5–10% など) システムがすぐにメモリー不足に陥ってしまうためメモリーの回収に非常に時間がかかることになります。

oom_adj

***panic_on_oom*** パラメーターが **0** に設定されている状態でメモリーを使い切ってしまうと、**oom_killer** 関数はシステムが復元できるようになるまでプロセスを強制終了し、プロセスを最も高い **oom_score** で起動します。

***oom_adj*** パラメーターはプロセスの ***oom_score*** を確定する際に役立ちます。このパラメーターはプロセスの識別子ごとに設定されています。**-17** の値を設定するとそのプロセスの **oom_killer** は無効になります。これ以外にも **-16** から **15** までの値を使用することができます。



**注記**

調整されたプロセスによって生成されたプロセスは、プロセスの **oom_score** を継承します。

スワップ

swappiness 値 (**0**〜**100**) は、システムが匿名メモリーやページキャッシュを優先する度合いを制御します。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** に設定すると非常に強力にスワップを避けるため、メモリーおよび I/O に圧力がかかりメモリー不足によるプロセスの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

本セクションのパラメーターは特に記載のない限り **/proc/sys/fs** にあります。

**aio-max-nr**

アクティブな非同期の全入出力コンテキスト内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値である **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 1/10 です。

この値を上げるとファイル処理数の不足が原因のエラーを解決できるようになります。

### 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されます。

**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、以下のコマンドを使用します。

```
# sysctl kernel.msgmax
```

**msgmnb**

1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、以下のコマンドを使用します。

```
# 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、以下のコマンドを使用します。

```
# sysctl kernel.msgmni
```

**shmall**

システム上で一度に使用可能な共有メモリーの総ページ数を定義します。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、以下のコマンドを使用します。

```
# 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、以下のコマンドを使用します。

```
# 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。いずれのシステムでもデフォルト値は **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、以下のコマンドを使用します。

```
# 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、「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に影響を与える要因を以下に示します。

- データの書き込みと読み取りのパターン
- ベースとなる配列でデータを整列
- ブロックサイズ
- ファイルシステムのサイズ
- ジャーナルサイズと場所
- レコーディングのアクセスタイム
- データ信頼性の確保
- 事前読み出しのデータ
- 事前割り当てのディスク領域
- ファイル断片化
- リソース競合

本章を読むと、ファイルシステムの処理能力、スケーラビリティ、応答性、リソース使用量、可用性などに影響を与えるフォーマットとマウントのオプションについて理解できます。

### 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** により、要求が I/O スケジューラーに到達した時点からの要求のレイテンシーが保証されます。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、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。1バッチが処理されると **deadline** は書き込み動作が待機している長さを確認して次の読み取りバッチまたは書き込みバッチを適宜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。**cfq** (Completely Fair Queueing) スケジューラー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 種類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** は過去データを使ってアプリケーションで発行される今後の I/O 要求数の増減を予測します。I/O の増加を予測した場合は他のプロセスの I/O が処理を待機している場合であってもアイドリングを行い新しい I/O を待ちます。

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 スケジューラーはシンプルな FIFO (first-in first-out) のスケジューリングアルゴリズムを実装しています。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、技術プレビュー 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。
- Redundant Array of Inexpensive Disks (RAID) の複数のバージョンのサポート。
- I/O エラーとファイルシステムオブジェクトのマップの逆参照。
- 透過的な圧縮 (パーティション上のすべてのファイルは自動的に圧縮されます)。
- データおよび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

デバイスのフォーマットを一旦行うと設定により決定される事項を変更できないファイルシステムがあります。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。

サイズ

負荷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

ブロックとはファイルシステムの作業単位であり、ブロックサイズとは1ブロック内に格納できるデータ量です。つまり1度に書き込まれる、または読み取られる最小限のデータ量になります。

一般的な使用例のほとんどでデフォルトのブロックサイズが適したサイズになります。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。複数ファイルが複数ブロックをまたいで使用することがありますが、実行時に余分なオーバーヘッドが生まれる可能性があります。また、ブロック数に制限のあるファイルシステムがあるため、この場合はファイルシステムの最大サイズも制限されることになります。

デバイスを **mkfs** コマンドでフォーマット化する際にファイルシステムのオプションとしてブロックサイズを指定します。ブロックサイズを指定するパラメーターはファイルシステムにより異なります。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

ファイルシステムの配列はファイルシステム全体にデータを配分することに関係しています。RAIDなどのストライプ型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ベースとなるストレージの配列でデータやメタデータを揃えることでパフォーマンスを改善できます。

多くのデバイスは特定のファイルシステムでフォーマットを行う際に自動的に設定される推奨配列をエクスポートします。使用するデバイスが推奨配列をエクスポートしない、または推奨されている設定を変更したい場合などは **mkfs** でデバイスのフォーマットを行う際に手作業で配列を指定する必要があります。

ファイルシステムの配列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。詳細は使用するファイルシステムの **mkfs** man ページをご覧ください。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

ファイルシステムのジャーナル機能は書き込み動作が実行される前にその書き込み動作で加えられる予定の変更をジャーナルファイルに記録します。これによりシステムクラッシュや停電などが発生した場合のデバイスの破損の可能性を低減し、復元プロセスをスピード化します。

メタデータの処理率が高い負荷の場合はジャーナルへの更新がかなり頻繁に必要になります。ジャーナルが大きいほどメモリー使用量も高くなりますが書き込み動作の頻度は減ります。また、メタデータの処理率が高い負荷を処理するデバイスのシーク時間を改善するには、そのジャーナルをより高速な処理が行えるプライマリーストレージとは別の専用ストレージに配置します。

**警告**

外部ジャーナルが信頼できるものであることを確認してください。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てください。ジャーナルデバイスはマウント時に指定されたものを使用します。詳細については **mkfs** の man ページおよび **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

本セクションではほとんどのファイルシステムに適用でき、デバイスのマウント時に指定することができるチューニングについて説明しています。

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。

アクセス時間

ファイルが読み込まれる度、アクセスが起きた時間 (**atime**) でそのメタデータが更新されます。これによりさらに書き込み I/O が発生します。デフォルトでは Red Hat Enterprise Linux 7 は前回のアクセス時間が最後の変更 (**mtime**) または状態変更 (**ctime**) の時間より古い場合にのみ **atime** フィールドを更新するため、ほとんどの場合、このオーバーヘッドが最小になります。

ただし、このメタデータの更新に時間がかかり、また正確なアクセス時間のデータは必要としない場合、**noatime** マウントオプションを付けてファイルシステムをマウントすることができます。このオプションを付けるとファイルの読み取り時のメタデータへの更新が無効になります。ま

た、**nodiratime** 動作が有効になるためディレクトリーの読み取り時のメタデータへの更新も無効になります。

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。先読みの値が高いほどより多くのデータを先に取得します。

Red Hat Enterprise Linux は検出対象に応じて適切な先読み値の設定を試行します。ただし、正確な検出が常に可能なわけではありません。たとえば、ストレージアレイを単一の LUN としてシステムに提示すると、システム側はそれを単一の LUN として検出するためアレイに適切な先読み値を設定しません。

連続 I/O による多量のストリーミングを必要とする作業負荷の場合は先読み値を高くすると役に立つことがよくあります。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

SSD およびシンプロビジョンのストレージいずれの場合もファイルシステムで使用されていないブロックは定期的に破棄することを推奨しています。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。指定した基準と一致するファイルシステム内の未使用ブロックをすべて破棄します。

Red Hat Enterprise Linux 7 では、物理的な破棄動作に対応している XFS と ext4 フォーマットのデバイスでバッチ破棄がサポートされます (つまり、**/sys/block/*devname*/queue/discard_max_bytes** の値がゼロ以外の HDD デバイスおよび **/sys/block/*devname*/queue/discard_granularity** の値が **0** 以外の 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** オプションを使って設定します。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

事前割り当てではディスク領域にデータを書き込むことなくその領域がファイルに割り当てられたという印を付けます。データの断片化および読み取り能力の低下などを制限する場合に便利です。Red Hat Enterprise Linux 7 では XFS、ext4、GFS2 のデバイスでのマウント時の領域事前割り当て

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 ページをご覧ください。**fallocate(2) glibc** 呼び出しを使用するとアプリケーションに対しても領域の事前割り当てが有効に働きます。

## 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って説明していきます。

### 8.2.1. vmstat を使っ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。I/O サブシステムがパフォーマンスの問題に関連していないかどうかを判断する際に役に立ちます。

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、書き込み動作のブロック (KB 単位)

**bo**
ブロックアウト、読み取り動作のブロック (KB 単位)

**wa**
I/O 動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** での分析で I/O サブシステムがパフォーマンスの低下に関連していることが示されている場合は **iostat** を使ってその I/O デバイスを確定することができます。

**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 を使った 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** での分析で I/O サブシステムがパフォーマンスの低下に関連していることが示されている場合は **iostat** を使ってその I/O デバイスを確定することができます。

**iostat** 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** の報告出力を特定のデバイスに集中させることができます。

```
$ man iostat
```

#### 8.2.2.1. blktrace を使った詳細な I/O 分析

**Blktrace** は I/O サブシステム内で費やされた時間配分に関する詳細情報を提供します。付属ユーティリティーの **blkparse** は **blktrace** から生の出力を読み取り **blktrace** で記録される入力および出力の動作の概要を読みやすい形で生成します。

このツールの詳細については、blktrace(8) and 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 を使った 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** の出力を分析して I/O スタックの各エリアでデータが消費する時間を表示するため I/O サブシステム内の障害を発見しやすくなります。

**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**)
- I/O イベントのドライバーイベントへのディスパッチ (**D**)
- 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ション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した **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が含まれるカーネル I/O スタックの下部で費やされた合計時間 (**Q2C**) を待機期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
[...]
```

デバイスがリクエストに対応するのに長時間かかる場合 (**D2C**)、デバイスが過負荷の状態であったり、

デバイスの送られたワークロードが最適ではないことがあります。ストレージデバイスがディスパッチされる前 (**Q2G**) に、ブロック I/O が長時間キューに置かれた場合、使用中のストレージは I/O の負荷に対応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、ブロックレイヤーに送信されたリクエストの間隔 (**Q2Q**) がリクエストがブロックレイヤーで費やす時間の合計 (**Q2C**) よりも大きいと **btt** が示す場合、I/O リクエスト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に関係しない可能性があることを意味します。

I/O 前後の **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。

このツールについての詳細は、btt(1) の man ページを参照してください。

```
$ man btt
```

#### 8.2.2.3. seekwatcher を使った blktrace 出力の分析

**seekwatcher** ツールは **blktrace** の出力を使って I/O の時間経過を図式化することができます。ディスク I/O の LBA (論理ブロックアドレス)、処理能力 (MB/秒)、毎秒のシーク数、毎秒の I/O 動作数を示します。いつデバイスの毎秒の動作数の制限に達しているかを確認する際に役立ちます。

このツールの詳細については man ページをご覧ください。

```
$ man seekwatcher
```

### 8.2.3. SystemTap を使っ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。

次の **SystemTap** のスクリプト例はストレージのパフォーマンスに関連し、ストレージやファイルシステムパフォーマンス関連の問題を診断する際に役立ちます。デフォルトでは **/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

SSD は円盤状の磁気記憶媒体を回転させるのではなく NAND フラッシュチップを使って永続データを格納します。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。円盤状の磁気記憶媒体のようにアクセスする際に生じるシーク時間がありません。HDD と比べてストレージ容量 1 ギガバイトあたりのコストが高く記憶密度は低いですが、待ち時間が短く処理能力に非常に優れています。

SSD 上の使用ブロックがディスクの最大容量に近づくにつれパフォーマンスは低下していきます。低下の度合いは製造元により異なりますが、いずれのデバイスであってもパフォーマンスの低下は見られます。破棄動作を有効にすることでこの低下を緩和することができる場合があります。詳細は「メンテナンス」をご覧ください。

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

いずれの I/O スケジューラーでもほとんどの SSD で正しく動作するはずです。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お勧めしています。I/O スケジューラーの切り替え方法については、**/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) サブシステムにも特別なチューニングは必要ありません。SSD での I/O が高速であることを考慮すると、書き込みアクティビティーが増加しても通常はディスク上の他の動作の待ち時間に悪影響を与えないはずです。したがっ

て、**vm_dirty_background_ratio** および **vm_dirty_ratio** の設定を低くすることができるでしょう。ただし、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お勧めできません。

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 でそのツールを使って 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** サービスには特定の使用事例でパフォーマンスを改善できるようデザインされたプロファイルが用意されています。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薦します。

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を指定しない場合、SATA ドライブでは **cfq** スケジューラーが使用され、その他すべてのドライブでは **deadline** スケジューラーが使用されます。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 スケジューラーは、**Tuned** ツールを使用するか、手動で **/etc/default/grub** ファイルを変更して設定することができます。

起動しているシステムのデフォルト I/O スケジューラーを指定する場合は、**Tuned** ツールの使用を推奨します。***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** プラグインの詳細については、『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修正するには、起動時またはシステムが起動している時に、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。**Tuned** ツールを使用するか、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」 で説明するように手動で **/etc/default/grub** ファイルを修正することができます。

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 で使用可能なスケジューラーは、**deadline**、**noop**、および **cfq** です。詳細については、お使いのカーネルのドキュメントに含まれている **cfq-iosched.txt** および **deadline-iosched.txt** ファイルを参照してください。これらは kernel-doc パッケージのインストールにより利用することができます。

2. 追加した ***elevator*** パラメーターを含む新しい設定ファイルを作成します。

   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

ここで取り上げる汎用チューニングパラメーターは、**/sys/block/sd*X*/queue/** ディレクトリーないで利用できます。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**

いくつかの I/O イベントが **/dev/random** のエントロピープールに貢献しています。これらの貢献のオーバーヘッドが計測可能になる場合、このパラメーターを **0** に設定することができます。

**iostats**

デフォルト値は **1** (***enabled***) です。***iostats*** を **0** に設定すると、デバイスに対する I/O 統計の収集が無効化され、I/O パスのオーバーヘッドが若干削減されます。一部の NVMe ソリッドステートストレージデバイスなどのパフォーマンスが非常に高いデバイスの場合、***iostats*** を **0** に設定すると、パ

フォーマンスが若干向上します。ストレージモデルに対してベンダーの指定がない限り、***iostats*** を有効の状態にする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** ファイル内に存在しなくなります。**/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。そのため、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 監視ツールの出力から除外されます。

max_sectors_kb

I/O 要求の最大サイズをキロバイト単位で指定します。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確定されます。パラメーターの最大値は ***max_hw_sectors_kb*** の値で確定されます。

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 は、常に ***max_sectors_kb*** を最適な I/O サイズと内部消去ブロックサイズの倍数にするよう推奨しています。これらの値がゼロであったり、ストレージデバイスの指定がない場合は、いずれかのパラメーターに ***logical_block_size*** の値を使用します。

nomerges

リクエストのマージはほとんどのワークロードに対して有用ですが、デバッグの目的で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、マージが有効になっています。簡単な 1 度のマージを無効にするには、***nomerges*** を **1** に設定します。あらゆる種類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の要求に対する次の処理をスリープ状態にするまでそれぞれ 128 個の読み取り要求と 128 個の書き込み要求をキュー待ちにすることができます。

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。***nr_requests*** は読み取りと書き込みで別々に適用されます。***nr_requests*** は I/O スケジューラー内の I/O 操作のみに適用され、基盤のデバイスにすでにディスパッチされた I/O 操作には適用されません。そのため、1 つのデバイスに対する未処理の I/O 操作の最大数は **(nr_requests*2)+(queue_depth)** になります (***queue_depth*** は **/sys/block/sdN/device/queue_depth** で LUN キューの深さと呼ばれることもあります)。たとえば、未処理の I/O 操作の合計数は **avgqu-sz** 列の **iostat** の出力に表示されます。

optimal_io_size

このパラメーターを使用すると最適な I/O サイズを報告するストレージデバイスがあります。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

シーケンシャル読み取り操作中にオペレーティングシステムが先に読み取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。

デバイスマッパーでは、多くの場合で ***read_ahead_kb*** の値を大きくすると効果的です。開始点として、各デバイスに対して 128 KB の値にするとよいでしょう。***read_ahead_kb*** の値を 4–8 MB に上げると、大型の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 SSD はそのソリッドステート状態を正しく通知しないため、従来の回転ディスクとしてマウントされます。ご使用の SSD デバイスで、この値が自動的に **0** に設定されない場合は、この値を手作業で設定し、スケジューラーで不要なシーク時間短縮ロジックを無効にします。

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** に設定するとこの機能を無効にして I/O の完了はその I/O 要求を発行したプロセッサーでしか行わないようにします。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

スケジューラーや特定ストレージデバイスのスケジューラー優先順位を設定するには、**/sys/block/*devname*/queue/scheduler** ファイルを編集します (*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

**deadline** を使用するとキュー待ちの 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。1 バッチが処理されると **deadline** は書き込み動作が待機している長さを確認して次の読み取りバッチまたは書き込みバッチを適宜スケジュールします。

**deadline** スケジューラーの動作に影響を与えるパラメーターです。

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

前方マージを一切生成しない作業負荷の場合、**0** に設定することは可能です。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 ではデフォルト値の **1** の使用を推奨しています。

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。プロセスのクラスを手作業で調整する場合は **ionice** コマンドを使って行います。

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。パラメーターは **/sys/block/*devname*/queue/iosched** ディレクトリー配下にある指定ファイルでデバイスごとに設定を変更します。

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 と 1040 KB など)、***back_seek_penalty*** が後方シークの距離に対して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

ミリ業単位の長さで指定します。同期 (読み取りまたは **O_DIRECT** 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 **125** ミリ秒です。

group_idle

このパラメーターはデフォルトでは **0** (無効) に設定されます。**1** (有効) に設定すると **cfq** スケジューラーは制御グループ内で 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。比例加重 I/O 制御グループを使用し、***slice_idle*** を **0** に設定している (高速ストレージで) 場合に便利です。

group_isolation

このパラメーターはデフォルトでは **0** (無効) に設定されます。**1** (有効) に設定するとグループ間をより強力に分離しますが、ランダムな作業負荷および連続した作業負荷の両方に対して公平性が適用されるため処理能力は低減されます。***group_isolation*** を無効にすると (**0** の設定) 公平性は連続した作業負荷にのみ適用されます。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ご覧ください。

low_latency

このパラメーターはデフォルトでは **1** (有効) に設定されます。有効の場合、**cfq** はデバイスで I/O を

発行している各プロセスごと最大 **300** ミリ秒の待ち時間を与え処理能力より公平性を優先させます。**0** (無効) に設定するとターゲットの待ち時間は無視され、各プロセスは完全なタイムスライスを受け取ります。

quantum

**cfq** がひとつのデバイスに一度に送信できる I/O 要求数を指定します。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** 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を行わない) 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

高速な外付けストレージアレイや SSD などの、シークによる影響をさほど受けないハードウェアの場合、**cfq** スケジューラーの使用は推奨しません。こうしたストレージで **cfq** を使用しなければならない場合は、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、主に高速のストレージを使用する CPU バウンドシステムで使用すると便利です。一般的に、**noop** I/O スケジューラーは仮想マシンが仮想ディスクに 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、この使用に限定されません。

**noop** I/O スケジューラーに固有するチューニング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

フォーマットオプションの詳細については man ページをご覧ください。

```
$ 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)。また最大値は **64** KB になります。

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 **mkfs.xfs** man ページをご覧ください。

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 **mkfs.xfs** の man ページをご覧ください。

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 **mkfs.xfs** の man ページをご覧ください。

RAID

ソフトウェア RAID を使用している場合は **mkfs.xfs** で自動的にベースとなるハードウェアが適切なストライプ単位とストライプ幅に設定されます。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。設定を行う場合は **mkfs.xfs -d** オプションを使用します。詳細は **mkfs.xfs** の man ページをご覧ください。

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。ログをストライプ単位に合わせるには **mkfs.xfs -d** オプションを使用します。詳細は **mkfs.xfs** の man ページをご覧ください。

ログサイズを設定するには以下の **mkfs.xfs** オプションを使用します。*logsize* にはログのサイズを入力します。

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

詳細については **mkfs.xfs** の 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** を使用すると自動的に適切なログのストライブ単位への設定が試行されます。ただし、この情報をエクスポートしている RAID デバイスによります。

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** により警告が発せられログのストライプ単位には 32 KB が設定されます。

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
```

詳細については **mkfs.xfs** の 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*** パラメーターを使用すると XFS が inode とデータをファイルシステム全体に割り当てるよう設定されます。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ます。

ログのバッファサイズと数

ログバッファ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。

ログのバッファサイズは ***logbsize*** マウントオプションで設定、ログバッファに格納できる情報量の最大値を定義します。ログのストライプ単位を設定していない場合はバッファの書き込みは最大値より短くなるため、同期の多い作業に対してはログのバッファサイズを小さくする必要はありません。ログバッファ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。

ログバッファ数は ***logbufs*** マウントオプションで指定します。デフォルト値は 8 ログバッファ (最大) ですが最小では 2 ログバッファを指定することができます。余分なログバッファ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数を減らす必要はありません。ログバッファ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*** パラメーターを使うと頻繁に変更されるメタデータは変更の度にログに書き込むのではなく、定期的なログへの書き込みを行わせることができるようになります。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**、**sync** などを使っ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なります。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。フォーマットのパラメーターの詳細については **mkfs.ext4** の man ページをご覧ください。

```
$ 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起こる割合を制御することができます。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

ジャーナル I/O の優先度はデフォルトでは通常の I/O より若干高い **3** です。マウント時に ***journal_ioprio*** パラメーターを使ってジャーナル I/O の優先度を制御することができます。***journal_ioprio*** に指定できる値は **0** から **7** の範囲です。**0** が最も優先度の高い I/O になります。

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** と **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時およびマウント時に GFS2 ファイルシステムに対して使用できるチューニングパラメーターの一部について説明しています。

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同様別のディレクトリーノ間隔も空ける場合は以下に示すように ***T*** 属性でそのディレクトリーに印を付けます。*dirname* には間隔を空けるディレクトリーのパスを入力します。

```
# 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構成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。

一部の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となることがあります。こうした問題の解決については本ガイドの範疇を超えるため説明の対象としていません。

## 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ースカード (NIC) で受信され、内蔵ハードウェアバッファまたはリングバッファ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構成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にパケットがなくなる、または一定のパケット数 (**/proc/sys/net/core/dev_weight** で指定) が転送されるまで続けられます。

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」は、Linux カーネルでのパケット受信に関する情報を含み、NIC チューニングの次の領域をカバーします。SoftIRQ ミス (netdev budget)、**tuned**、チューニングデーモン、**numad** NUMA デーモン、CPU の電源状態、割り込みの分散、一時停止フレーム、割り込みの統合、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、ジャンボフレーム、TCP および UDP プロトコルチューニング、ならびに NUMA ローカリティー。

### 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その大部分が自己最適化を行いますが、ネットワークパケットの処理中にボトルネックとなってパフォーマンスの低下を招くポイントがいくつかあります。

**NIC** ハードウェアバッファまたはリングバッファ

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

割り込みにより待ち時間が増大しプロセッサーの競合を招く場合があります。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

要求しているアプリケーションに対して多数のパケットがコピーされない場合、**/proc/net/snmp** の UDP 入力エラー (**InErrors**) が増加する場合などはアプリケーションの受信キューでのボトルネックを示しています。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構成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長期間に渡りデバイスパフォーマンスの評価を行うことができます。デフォルトでは接続を確立していてリスニングを行っていないオープンの TCP ソケットが表示されますが、特定のソケットに関する統計を出力から除去できる便利なオプションがいくつか用意されています。

Red Hat Enterprise Linux 7 では **netstat** 経由での **ss** を推奨しています。

**ss** は iproute パッケージで提供されます。詳細については 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ルート、デバイス、ルーティングポリシー、トンネルなどの管理と監視を行うことができます。**ip monitor** では引き続きデバイス、アドレス、ルートなどの状態を監視することができます。

**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** はカーネルでドロップされたパケットの監視と記録ができるインテラクティブなツールになります。

詳細については **dropwatch** の man ページをご覧ください。

```
$ man dropwatch
```

### 9.2.4. ethtool

**ethtool** ユーティリティーを使用するとネットワークインターフェースカードの設定を表示したり編集したりすることができます。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** に監視したいデバイス名を付けて使用するとそのデバイスのカウンターの状態を表示させることができます。

```
$ 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** には IP、ICMP、TCP、UDP などの監視と管理のため snmp エージェントで使用されるデータが表示されます。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**) が増えている場合はソケット受信キューでの障害を示している可能性があります。

### 9.2.6. SystemTap を使っ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。

ネットワーク構成に関連し、ネットワークパフォーマンスの問題を診断する際に役立つ **SystemTap** のサンプルスクリプトを以下に示します。デフォルトでは **/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 秒ごとにプロセス一覧 (プロセス ID とコマンド)、送受信したパケット数、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て、追跡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数を出力します。シンボリック名を表示させる場合は **--all-modules** オプションを使用します。

**latencytap.stp** スクリプトは異なるタイプの待ち時間が 1 つ以上のプロセスに与える影響を記録します。30 秒ごとに待ち時間タイプの一覧がプロセスの待機した合計時間の降順 (待機時間が長い順) に分類されて出力されます。ストレージやネットワーク両方の待ち時間の原因を特定する際に役に立ちます。Red Hat では、待ち時間イベントの原因が特定できるようこのスクリプトには **--all-modules** オプションを使用することを推奨しています。デフォルトではこのスクリプトは **/usr/share/doc/systemtap-client-*version*/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。ツールを使っ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** サービスではいくつかの使用例でパフォーマンスを改善させるプロファイルを数種類用意しています。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

多数のパケットがハードウェアバッファーによってドロップされる場合に考えられるソリューションがいくつかあります。

入力トラフィックの速度を落とす

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キューのサイズ変更

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
```

キューの排出の割合を変更する

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*** パラメーターで管理されているデバイス加重を増やすことでキューが排出される割合を増加させることができます。一時的に変更する場合は **/proc/sys/net/core/dev_weight** ファイルの内容を編集します。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** を使って変更します。

キューの排出の割合を変更するのがネットワークパフォーマンスの低下を軽減する最もシンプルな方法になります。ただし、デバイスで一度に受信できるパケット数を増やすと余分なプロセッサー時間を使用することになります。この間、他のプロセッサーはスケジュールに入れられないため、別のパフォーマンスの問題を引き起こす可能性があります。

### 9.3.3. 割り込みキューの設定

分析結果が待ち時間の高さを示している場合はパケットの受信を割り込みベースではなくポーリングベースに変更すると改善される可能性があります。

#### 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。このため割り込みおよびその結果となるコンテキストスイッチによる遅延が解消されます。ただし CPU の使用率も増大します。ビジーポーリングでは CPU がスリープ状態になるのが阻止されるため余分な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 では **50** の設定を推奨しています。
- **SO_BUSY_POLL** ソケットオプションをソケットに追加します。

グローバルにポーリングを有効にする場合は ***sysctl.net.core.busy_read*** のパラメーターも **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されます。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** の設定を推奨しています。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"
```

上記のコマンドを実行して **busy-poll: on [fixed]** が返された場合はそのデバイスでビジーポーリングを使用することができます。

### 9.3.4. ソケット受信キューの設定

分析結果がソケットキューの排出の割合が遅すぎるためパケットがドロップされてしまうことを示している場合は、このことが原因とな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

パケットがキューに到達する前にフィルターをかけたりドロップする、またはデバイスの加重を低くするなどの方法でキューが満杯になる割合を低下させます。

アプリケーションのソケットキューの深さを増やす

限られたトラフィック量を集中的に受信するようなソケットキューの場合、その集中的なトラフィック量に合うようソケットキューの深さを増やすとパケットがドロップされなくなる場合があります。

#### 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ー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。デバイス加重は ***dev_weight*** パラメーターで管理します。一時的に変更する場合は **/proc/sys/net/core/dev_weight** ファイルの内容を編集します。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** を使って変更します。

#### 9.3.4.2. キューの深さを増やす

ソケットキューの排出の割合を改善させる場合、アプリケーションのソケットキューの深さを増やすのが一般的には最も簡単な方法になりますが長期的なソリューションにはならない場合が多くあります。

キューの深さを増やすには次のような変更を行うことでソケット受信バッファのサイズを増やします。

**/proc/sys/net/core/rmem_default** の値を増やす

このパラメーターにより、ソケットで使用され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** の値以下にする必要があります。

**setsockopt** を使って **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ってバッファーの現在の値を決定します。この詳細については、socket(7) man ページをご覧ください。

### 9.3.5. RSS (Receive-Side Scaling) の設定

RSS (Receive-Side Scaling) はマルチキュー受信とも呼ばれ、ネットワーク受信プロセスを複数のハードウェアベースの受信キューに分散させることで、インバウンドのネットワークトラフィックを複数の CPU で処理させることができるようになります。RSS は単一 CPU のオーバーロードで発生する受信割

り込み処理におけるボトルネックを緩和し、ネットワークの待ち時間を短くすることができます。

ご使用のネットワークインターフェースカードが RSS に対応しているか確定するには、複数の割り込み要求キューが **/proc/interrupts** 内でそのインターフェースに関連付けられているかどうかチェックします。たとえば、**p1p1** インターフェイスを確認する場合は以下を実行します。

```
# 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 ドライバーにより **p1p1** インターフェースに 6 つの受信キューが作成されたことを示しています (**p1p1-0** から **p1p1-5**)。また、各キューでいくつの割り込みが処理されたか、どの CPU が割り込みを処理したかも示しています。この例の場合、システムには 6 つの CPU があり、この NIC ドライバーはデフォルトで 1 CPU ごと 1 つのキューを作成するためキューが 6 つ作成されています。NIC ドライバーでは一般的なパターンです。

別の方法では、ネットワークドライバーが読み込まれた後に **ls -1 /sys/devices/*/*/*device_pci_address*/msi_irqs** の出力をチェックすることもできます。たとえば、PCI アドレスが **0000:01:00.0** のデバイスをチェックするには、以下のコマンドを実行すると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** ドライバーは ***num_queues***、**sfc** ドライバーは ***rss_cpus*** パラメーターで設定されます。いずれにしても、通常は **/sys/class/net/*device*/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)、*rx-queue* は該当する受信キューの名前になります。

RSS を設定する場合はキューの数の設定は物理 CPU コアごと 1 キューに限定することを推奨しています。分析ツール上では複数のハイパースレッドが別々のコアとして表示されることがよくありますが、ハイパースレッド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のがネットワークパフォーマンスの改善につながるとは証明されていません。

RSS が有効になっていると、ネットワーク処理は各 CPU がキューに登録した処理量に基づいて CPU 間で均等に分散されます。ただし、**ethtool *--show-rxfh-indir*** および ***--set-rxfh-indir*** パラメーターを使ってネットワークアクティビティーの配分方法を修正したり、特定タイプのネットワークアクティビティーを他より重要なアクティビティーとして加重することもできます。

**irqbalance** デーモンを RSS と併用するとノード間メモリー転送やキャッシュラインバウンスの可能性が低減されます。ネットワークパケット処理の待ち時間が少なくなります。

### 9.3.6. 受信パケットステアリング (RPS) の設定

受信パケットステアリング (Receive Packet Steering - RPS) はパケットを処理のため特定の CPU にダイレクトするという点では RSS と似ています。しかし、RPS はソフトウェアレベルで実装されるため単一ネットワークインターフェー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でボトルネックにならないよう阻止する際に役立ちます。

RPS には ハードウェアベースの RSS と比較して、以下のような利点があります。

- RPS はすべてのネットワークインターフェースカードで使用できます。
- 新たなプロトコルに対応するようソフトウェアフィルターを簡単に追加することができます。
- ネットワークデバイスのハードウェア割り込み率は上がりません。ただし、プロセッサー間の割り込みが生まれます。

RPS は **/sys/class/net/*device*/queues/*rx-queue*/rps_cpus** ファイル内でネットワークデバイスおよび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* は該当する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および受信キューからのパケットを処理する CPU で該当ファイル **rps_cpus** を設定します。

**rps_cpus** ファイルはコンマで区切った CPU ビットマップを使用します。つまり、CPU にインターフェース上の受信キューの割り込みを処理させるには、ビットマップ上の CPU 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場合は **rps_cpus** の値を **f** (15 の 16 進数表記) に設定します。2 進数表記では、15 は **00001111** (1+2+4+8) で表されます。

単一の送信キューのネットワークデバイスでは、同一メモリードメイン内の CPU を使用するように RPS を設定すると、最善のパフォーマンスが達成できます。これは、NUMA 以外のシステム上では、利用可能な CPU すべてが使用されることを意味します。ネットワーク割り込み率が非常に高い場合は、ネットワーク割り込みを処理する CPU を除外するとパフォーマンスが改善する場合もあります。

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

CPU キャッシュのヒット率を高めネットワークの待ち時間を減らすため RPS の動作を拡張したのが RFS (Receive Flow Steering) です。RPS はキューの長さのみに基づいてパケットを転送する一方、RFS は RPS のバックエンドを使用して最適な CPU を計算してからパケットを消費するアプリケーションの場所に応じてそのパケットを転送します。これにより CPU のキャッシュ効率が高まります。

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。適度なサーバー負荷の場合は **32768** の設定値を推奨しています。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* は設定するネットワークデバイス名 (**eth0** など)、*rx-queue* は設定する受信キューになります (**rx-0** など)。

**rps_sock_flow_entries** を **N** で割った値をこのファイルの値に設定します。**N** はデバイス上の受信キューの数です。たとえば、**rps_flow_entries** が **32768** に設定されていて設定済みの受信キューが 16 ある場合、**rps_flow_cnt** は **2048** に設定します。単一キューデバイスの場合は **rps_flow_cnt** の値は **rps_sock_flow_entries** の値と同じになります。

一人の送信者から受け取ったデータは複数の CPU には送られません。一人の送信者から受け取ったデータ量が単一 CPU で処理できる量を越えている場合はフレームサイズを大きくして割り込み数および CPU の処理量を減らします。あるいは NIC オフロードオプションまたは高速な CPU の導入を考慮してください。

RFS と併せて **numactl** または **taskset** を使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みてください。これにより、パケットが間違った順番で処理されることを防ぐことができます。

### 9.3.8. アクセラレート RFS の設定

アクセラレート RFS はハードウェアに対する支援が加わり RFS の速度が更に高まります。RFS 同様、パケットはパケットを消費するアプリケーションの位置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。

アクセラレート RFS は、以下の条件が満たされた場合にのみ、利用可能になります。

- アクセラレート RFS がネットワークインターフェー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** フィルタリングが有効になっていること。

条件が満たされるとキューマッピングに対する CPU が従来の RFS 設定に応じて自動的に推測されます。つまり、キューマッピングに対する CPU は各受信キューのドライバーで設定される IRQ 親和性に応じて推測されるということになります。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ー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** はコマンドラインツールです。ハードウェアの割り込みをプロセッサー間で配分してシステムのパフォーマンスを改善します。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 度だけの実行も可能です。

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 **softirq** ワークロードを 1 標準偏差以上下回り、平均値から 1 標準偏差を超える CPU がなく、複数の **irq** がこれらに割り当てられている場合、CPU は省電力モードに入ります。このモードでは、CPU は **irq** バランシングの一部ではないので、不必要に再開されることがありません。

--hintpolicy

**irq** カーネル親和性ヒントの処理方法を決定します。**exact** (**irq** 親和性ヒントを常に適用)、**subset** (**irq** は均等に分散されるが割り当てオブジェクトは親和性ヒントのサブセット)、または **ignore** (**irq** 親和性ヒントを完全に無視) の値を使用できます。

--policyscript

各割り込み要求に実行するスクリプトの場所を定義します。引数として渡される **irq** 番号とデバイスパス、**irqbalance** で期待されるゼロ終了コードを付けます。定義されたスクリプトでは、渡された **irq** の管理で **irqbalance** をガイドするために、ゼロもしくはそれ以上の鍵と値のペアを指定することができます。

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** をバランシングから除外) もしくは **false** (この **irq** でバランシングを実施) です。

balance_level

渡された **irq** のバランスレベルをユーザーが上書きできるようにします。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きます。有効な値は **none**、**package**、**cache**、または **core** になります。

numa_node

渡された **irq** にはローカルとみなされる NUMA ノードを、ユーザーが上書きできるようにします。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)、および **irq** が全ノードから等距離であるとみなすことを指定する **-1** になります。

--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一覧に追加されます。

また、***IRQBALANCE_BANNED_CPUS*** 環境変数を使って **irqbalance** に無視される 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ネットワークインターフェースカードの設定を表示したり編集したりすることができます。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**、ethtool のオプション、使い方などについての詳細は man ページの記載をご覧ください。

```
$ 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長期間に渡りデバイスパフォーマンスの評価を行うことができます。デフォルトでは接続を確立していてリスニングを行っていないオープンの TCP ソケットが表示されますが、特定のソケットに関する統計を出力から除去できる便利なオプションがいくつか用意されています。

よく使用されるコマンドの一つに **ss -tmpie** があります。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**)、ソケットを使用しているプロセス (**p**)、ソケット情報の詳細 (**i**) などを表示します。

Red Hat Enterprise Linux 7 では **netstat** 経由での **ss** を推奨しています。

**ss** は iproute パッケージで提供されます。詳細については 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作業負荷環境でパフォーマンスが向上するようオペレーティングシステムを適合させるチューニングデーモンになります。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。

動的なチューニング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** がシステム統計を周期的に分析し、その統計を使用してシステムのチューニング設定を更新します。***update_interval*** パラメーターを使用すると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** のプロファイルを切り替えて、特定の使用事例でパフォーマンスの向上を可能にするコマンドラインツールです。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 より、**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あらゆるシェルコマンドを実行でき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た **Tuned** の拡張が可能になります。

また、Red Hat Enterprise Linux 7 ではプロファイル定義ファイルに ***include*** パラメーターが用意されるため、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 HugePage を有効にし、**cpupower** を使って **performance** cpufreq ガバナーを設定します。また、***kernel.sched_min_granularity_ns*** を **10** µs に ***kernel.sched_wakeup_granularity_ns*** を 15 µs に ***vm.dirty_ratio*** を **40**% にそれぞれ設定します。

**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。

**intel_pstate** と **max_perf_pct=100** を設定して節電よりパフォーマンスを重視します。透過的な大規模ページを有効にし、**cpupower** を使って **performance** cpufreq ガバナーを設定、***cpu_dma_latency*** 値に **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。

このプロファイルでは、**intel_pstate** と **min_perf_pct=100** を設定することにより、節電よりパフォーマンスが優先されます。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 ガバナーが設定され、**1** の ***cpu_dma_latency*** 値が要求されます。さらに、***busy_read*** と ***busy_poll*** の時間が **50** µs、***tcp_fastopen*** が **3** に設定されます。

**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。透過的な大規模ページを有効にし、**cpupower** を使って **performance** cpufreq ガバナーを設定します。また、***kernel.sched_min_granularity_ns*** を **10** µs に ***kernel.sched_wakeup_granularity_ns*** を 15 µs に ***vm.dirty_ratio*** を **40**% にそれぞれ設定します。

**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。

**intel_pstate** と **max_perf_pct=100** を設定して節電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。透過的な大規模ページを有効にし、**cpupower** を使って **performance** cpufreq ガバナーを設定します。***kernel.sched_min_granularity_ns*** を **10** µs に ***kernel.sched_wakeup_granularity_ns*** を 15 µs に ***vm.dirty_ratio*** を **40**% にそれぞれ設定します。

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。

**intel_pstate** と **max_perf_pct=100** を設定して節電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。透過的な大規模ページ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** を使って **performance** cpufreq ガバナーを設定します。***kernel.sched_min_granularity_ns*** を **10** µs に ***kernel.sched_wakeup_granularity_ns*** を 15 µs に ***kernel.sched_migration_cost*** を **5** µs に ***vm.dirty_ratio*** を **40**% にそれぞれ設定します。

**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 にパーティションを設定します。分離された CPU でジッターおよび割り込みを減らすために、プロファイルは、ユーザー空間プロセス、移動可能なカーネルスレッド、割り込みハンドラー、およびカーネルタイマーから分離された CPU を消去します。

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** ファイルに **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 の一覧を表示します。分離されている CPU の一覧はコンマで区切られているか、ユーザーが範囲を指定できます。範囲は、ハイフンを使用して指定できます (例: **3-5**)。このオプションは必須です。ここに挙げられていない CPU は、自動的にハウスキーピング CPU と見なされます。

**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より考慮されない CPU の一覧を表示します。このオプションは任意です。これは、通常、**isolated_cores** と同じ一覧になります。

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

**tuned-adm** で提供される節電プロファイルの詳細は『Red Hat Enterprise Linux 7 電力管理ガイド』を参照してください。

**tuned-adm** の使用に関する詳細は man ページをご覧ください。

```
$ man tuned-adm
```

## A.6. PERF

**perf** ツールは便利なコマンドを複数提供し、その一部を本セクションで説明しています。**perf** に関する詳細は、Red Hat Enterprise Linux 7 開発者ガイド または man ページを参照してください。

**perf stat**

実行された指示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総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 からは **perf stat** を使って 1 コントロールグループ (cgroups) または複数のコントロールグループに応じてモニタリングにフィルターをかけることができるようになります。

詳細については man ページをご覧ください。

```
$ man perf-stat
```

**perf record**

このコマンドでパフォーマンスデータをファイルに記録し、後で **perf report** を使って分析を行うことができます。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**

特定のマシン上で利用可能なイベントを表示します。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**

**top** ツールとよく似た機能を実行します。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、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**

**strace** ツールとよく似た機能を実行します。指定スレッドまたはプロセスで使用されるシステムコールとそのアプリケーションで受信され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。全一覧は 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 Inference Engine。 |
| **pmlogger** | パフォーマンスメトリックスロガー。 |
| **pmmgr** | ゼロ以上の設定ディレクトリーに応じて、Performance Metric Collector Daemon (PMCD) が実行されている検出済みローカルおよびリモートホストセットの 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。 |
| **pmatop** | パフォーマンスの観点から、もっと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) のシステムレベル占有状態を表示します。 |
| **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ス値をプロットします。 |
| **pmclient** | Performance Metrics Application Programming Interface (PMAPI) を使用して高レベルシステムパフォーマンスメトリックスを表示します。 |
| **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、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** | 所定の時間ウィンドウで 1 つまたは 2 つのアーカイブの各メトリックの平均値を比較します (この変化はパフォーマンスの低下の原因を探すときに重要になります)。 |
| **pmdumplog** | Performance Co-Pilot アーカイブファイルから制御、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。 |

| 名前 | 説明 |
| --- | --- |
| **pmdumptext** | ライブで収集された、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スの値を出力します。 |
| **pmerr** | 利用可能な Performance Co-Pilot エラーコードとその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** | 算術、論理式、およびルール式のセットを定期的に評価する推論エンジン。メトリックスは、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スに関する情報を表示します。メトリックスは、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 **device-mapper** デバイス (**-x dm** オプションを使用) の I/O 統計を報告します。 |
| **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。 |
| **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、または修正します。 |
| **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ス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スの可用性を決定します。 |
| **pmrep** | 選択されたメトリックス値、簡単にカスタマイズ可能なメトリックス値、パフォーマンスメトリックス値について報告します。 |
| **pmsocks** | ファイアウォールを介した 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スの値を変更します。 |

| 名前 | 説明 |
| --- | --- |
| **pmtrace** | トレース Performance Metrics Domain Agent (PMDA) にコマンドラインインターフェースを提供します。 |
| **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を数えるカウンター、クラスター化のカウンター、およびクラスターへの失敗を数えるカウンターも含まれます。 |
| xfs.allocs.*<br>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。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比較、および btree からのエクステントの記録作成および削除も含まれます。 |
| xfs.block_map.*<br>xfs.bmap_tree.* | メトリックにはブロックマップの読み書き数とブロック削除の数や、挿入、削除、および検索のエスクテントリスト操作が含まれます。ブロックマップからの比較、検索、挿入、および削除操作の操作カウンターも含まれます。 |
| xfs.dir_ops.* | 作成、エントリー削除、「getdent」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。 |
| xfs.inode_ops.* | オペレーティングシステムが inode キャッシュで異なる結果の XFS inode を検索した回数のカウンター。キャッシュのヒット数、キャッシュのミス数などをカウントします。 |
| xfs.log.*<br>xfs.log_tail.* | XFS ファイルシステム上でのログバッファー書き込み数のカウンター。ディスクへ書き込まれたブロック数が含まれます。また、ログフラッシュおよびピン留めの数のメトリックも含まれます。 |
| 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。 |

| メトリックグループ | 提供されたメトリック |
| --- | --- |
| 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。カウンターには要求されたバッファー呼び出しの数、成功したバッファーロック数、待機したバッファーロック数、miss_locks、miss_retries およびページ検索時のバッファーのヒット数が含まれます。 |
| 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。制御メトリックは pmstore ツールによっ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を数えるカウンター、クラスター化のカウンター、およびクラスターへの失敗を数えるカウンターも含まれます。 |
| xfs.perdev.allocs.*<br>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。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比較、および btree からのエクステントの記録作成および削除も含まれます。 |
| xfs.perdev.block_map.*<br>xfs.perdev.bmap_tree.* | メトリックにはブロックマップの読み書き数とブロック削除の数や、挿入、削除、および検索のエスクテントリスト操作が含まれます。ブロックマップからの比較、検索、挿入、および削除操作の操作カウンターも含まれます。 |
| xfs.perdev.dir_ops.* | 作成、エントリー削除、「getdent」操作の数に対する XFS ファイルシステムのディレクトリー操作のカウンター。 |
| xfs.perdev.transactions.* | メタデータトランザクション数のカウンター。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。 |
| xfs.perdev.inode_ops.* | オペレーティングシステムが inode キャッシュで異なる結果の XFS inode を検索した回数のカウンター。キャッシュのヒット数、キャッシュのミス数などをカウントします。 |
| xfs.perdev.log.*<br>xfs.perdev.log_tail.* | XFS ファイルシステム上でのログバッファー書き込み数のカウンター。ディスクへ書き込まれたブロック数が含まれます。また、ログフラッシュおよびピン留めの数のメトリックも含まれます。 |
| xfs.perdev.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.perdev.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。 |

| メトリックグループ | 提供されたメトリック |
| --- | --- |
| xfs.perdev.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.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。カウンターには要求されたバッファー呼び出しの数、成功したバッファーロック数、待機したバッファーロック数、miss_locks、miss_retries およびページ検索時のバッファーのヒット数が含まれます。 |
| 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、CPU アクティビティーに関する報告を出力します。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**

アクティブなメモリーと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**

ヘッダーの出現を定期的ではなく 1 度のみに指定します。

**-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 を指定せず delay を指定すると **vmstat** は無期限で報告を行います。

**-d**

ディスクの統計情報を表示します。

**-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情報を報告します。

**-S**

レポートで出力される単位を指定します。**k** (1000 バイト)、**K** (1024 バイト)、**m** (1,000,000 バイト)、**M** (1,048,576 バイト) の値が使用できます。

**-D**

ディスクの動作に関する概要統計を報告します。

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パフォーマンスと電力消費効率のバランスを指定することができます。このツールは 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。これがデフォルト値です。

**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** ツールは別々の状態でシステムが消費する時間量についての詳細な情報を提供します。**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と以下の見出しの下に各カウンター結果を 5 秒ごとに出力します。

**pkg**
プロセッサーのパッケージ番号

**core**
プロセッサーのコア番号

**CPU**
Linux CPU (論理プロセッサー) 番号

**%c0**
CPU が指示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 はターボモードになります。

**TSC**
間隔全体を通じたクロック平均速度

**%c1、%c3、および %c6**
プロセッサーが c1、c3、または c6 の各状態だったあいだの間隔の割合

**%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出力の間隔を指定します。結果を 10 秒ごとに出力させる場合は **turbostat -i 10** を実行します。

**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 からは **turbostat** で c7、c8、c9、c10 の各状態に対応するようになります。

## A.11. NUMASTAT

**numastat** ツールは numactl パッケージで提供されます。NUMA ノードベースでオペレーティングシステムとプロセッサーのメモリー統計情報 (割り当てヒットとミスなど) 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簡単に説明します。

**numa_hit**
当該ノードへの割り当てに成功したページ数

**numa_miss**
当初の意図されたノードがメモリー不足だったために当該ノードに割り当てられたページ数、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上にある

numa_foreign
: 当初は当該ノードへの割り当てを意図したもので、別のノードに割り当てられたページ数、**numa_foreign** イベントには対応する **numa_miss** イベントが別のノード上にある

interleave_hit
: 当該ノードへの割り当てに成功したインタリーブポリシーページ数

local_node
: 当該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

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適用すると、メモリーの表示単位がメガバイトに変更され (四捨五入で小数点第 2 位まで)、以下のように他の特定の **numastat** 動作に変更を加えます。

-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
: ノードあたりでのシステム全体のメモリー使用量を表示します。**/proc/meminfo** にある情報に類似したものです。

-n
: オリジナルの numastat コマンド (**numa_hit**、**numa_miss**、**numa_foreign**、**interleave_hit**、**local_node**、および **other_node**)と同じ情報が表示されますが、測定単位にメガバイトを使用した更新フォーマットが使われます。

-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 の値が数字の場合は、**numastat** は数値プロセス識別子と仮定します。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、管理者は指定したスケジュールまたはメモリー配置ポリシーでプロセスを実行することができます。**Numactl** は共有メモリーセグメントやファイルに永続的なポリシーを設定したり、プロセスのプロセッサー親和性やメモリー親和性を設定することもできます。

**Numactl** には便利なオプションが多くあります。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、システム上で利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および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および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** は自動で NUMA の親和性を管理するデーモンです。NUMA トポロジーとシステム内のリソース使用を監視して NUMA によるリソース管理と管理を動的に改善します。

**numad** が有効になると、この動作がデフォルトの自動 NUMA バランシングの動作に優先されること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、単に以下のコマンドを実行します。

```
# numad
```

**numad** の実行中は、アクティビティーが **/var/log/numad.log** にログ記録されます。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 親和性の改善のためになされた変更は削除されません。システムの使用方法が大幅に変わる場合は、**numad** を再度実行することで親和性が調整され、新たな条件の下でパフォーマンスが改善されます。

**numad** 管理を特定プロセスに限定するには、以下のオプションで開始します。

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

**-p *pid***
指定 *pid* を明示的な対象一覧に加えます。指定されたプロセスは **numad** プロセスの重要度しきい地に達するまで管理されません。

**-S 0**
プロセススキャンのタイプを **0** に設定し、**numad** 管理を明示的に対象プロセスに限定します。

使用できる **numad** オプションについては **numad** の man ページをご覧ください。

```
$ man numad
```

### A.13.2. numad のサービスとしての使用

**numad** をサービスとして実行する一方、現在のシステムの負荷に応じて動的にシステムのチューニングを行います。アクティビティーは **/var/log/numad.log** にログ記録されます。

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
```

使用できる **numad** オプションについては **numad** の man ページをご覧ください。

```
$ man numad
```

### A.13.3. プレプレースメントアドバイス

**numad** ではプレプレースメントアドバイスサービスを提供しています。各種のジョブ管理システムがこれに問い合わせをして、プロセスのメモリーリソースと CPU の初期組み合わせの際に役立ちます。プレプレースメントアドバイスは **numad** がサービスもしくは実行可能ファイルとして実行しているかに関わらず利用できます。

### A.13.4. KSM をともなう numad の使用

KSM を NUMA システム上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** に変更して NUMA ノードにまたがる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 デーモンが大量のメモリーページをマージすると、**numad** は利用可能なメモリーの正確な分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 は維持負担の少ないシステム全体のパフォーマンス監視ツールで、oprofile パッケージが提供します。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。レガシーの **opcontrol** ツールと新たな **operf** ツールは相互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時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どのデータが収集されるかを設定します。

**operf**

**opcontrol** の代わりとなる予定です。**operf** ツールは Linux Performance Events Subsystem を使用するので、単一プロセスとしてもシステムワイドとしてもより正確なプロファイリングのターゲットが可能になります。また、システム上で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 OProfile がよりうまく共存することが可能になります。**opcontrol** とは異なり、初期設定が不要で、**--system-wide** オプションを使用していなければ、root 権限なしで使用できます。

**opreport**

プロファイル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**、および post-processing ツール) は依然使用可能ですが、プロファイリング方法としては推奨されません。

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。これを使用すると実行中プロセスのプロセッサー親和性を読み出して設定したり、指定プロセッサー親和性でプロセスを起動することができるようになります。

**重要**

**taskset** はローカルの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りパフォーマンスを向上させる必要がある場合は taskset ではなく **numactl** を使用することを推奨しています。

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 -c processors pid
```

*processors* をコンマ区切りのプロセッサー一覧または **1,3,5-7** のようにプロセッサーの範囲で置き換えます。*pid* を再構成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一覧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** についての詳細情報は、man ページを参照してください。

```
$ man taskset
```

## A.16. SYSTEMTAP

**SystemTap** に関する情報は、SystemTap 独自のガイドである Red Hat Enterprise Linux 7 バージョン向けの『SystemTap ビギナーズガイド』および『SystemTap タップセットリファレンス』に記載されています。

# 付録B 改訂履歴

**改訂 10.14-00.1** Wed Sep 23 2020 Ludek Janda
翻訳ファイルを XML ソースバージョン 10.14-00 と同期

**改訂 10.14-00** Fri Apr 6 2018 Marek Suchánek
7.5 GA 公開用ドキュメントの準備

**改訂 10.13-59** Mon May 21 2018 Marek Suchánek
非同期のアップデート。

**改訂 10.13-58** Fri Mar 23 2018 Marek Suchánek
新しいセクション: pqos.

**改訂 10.13-57** Wed Feb 28 2018 Marek Suchánek
非同期のアップデート。

**改訂 10.13-50** Thu Jul 27 2017 Milan Navrátil
7.4 GA 公開用ドキュメントバージョン

**改訂 10.13-44** Tue Dec 13 2016 Milan Navrátil
非同期のアップデート。

**改訂 10.08-38** Wed Nov 11 2015 Jana Heves
7.2 GA リリース向けのバージョン

**改訂 0.3-23** Tue Feb 17 2015 Laura Bailey
RHEL 7.1 GA 向けにビルド。

**改訂 0.3-3** Mon Apr 07 2014 Laura Bailey
RHEL 7.0 GA 用に再構築。