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

Red Hat Enterprise Linux 7 でサブシステムの処理能力を最適化する

Red Hat Subject Matter ExpertsLaura Bailey

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

Red Hat Enterprise Linux 7 でサブシステムの処理能力を最適化する

Red Hat Subject Matter Experts

Laura Bailey

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。

目次

第1章 Red Hat Enterprise Linux 7 パフォーマンス関連の機能

このセクションでは、Red Hat Enterprise Linux 7 で行われているパフォーマンス関連の変更点について簡単に説明しています。

1.1. 7.1 の新機能

- カーネルのアイドルバランスメカニズムでは同じ CPU 上で後半の方でキュー待ちに置かれたタスクが遅延する可能性が少なくなり、アイドルバランス機能を使用した場合の処理待ち時間が低減されます。
- **swapon** コマンドに新しいパラメーター***--discard*** が加わりました。スワップを行う場合に使用する破棄ポリシーを選択できるようになるため柔軟性が飛躍的に高まり SSD (ソリッドステートディスク) でのパフォーマンスが向上します。デフォルトでは **swapon** を実行すると ***--discard*** パラメーターにより swap 領域全体が破棄されます。このため、スワップ中にはまず空き swap ページが破棄されてから再利用が行われます。詳細は **swapon** の man ページ (**man swapon**) をご覧ください。
- tuned プロファイル内で ***cmdline*** パラメーターとカーネルパラメーターの一覧を指定しカーネルのコマンドラインパラメーターを変更するよう **tuned** プロファイルを設定することができるようになります。***cmdline*** で指定したカーネルパラメーターはすべて再起動後に反映されます。

1.2. 7.0 の新機能

- 本ガイドは Red Hat Enterprise Linux 7 に合わせ完全な書き換えおよび再構成を行っています。
- **deadline** が **cfq** に代わり Red Hat Enterprise Linux 7 のデフォルトの I/O スケジューラーになります。この変更により一般的な使用事例のほとんどでパフォーマンスが向上されます。
- XFS ファイルシステムが ext4 に代わりデフォルトのファイルシステムになります。ファイルシステムの最大対応サイズは 500 TB、最大ファイルオフセットは 8 EB になります (スパースファイル)。XFS のチューニング推奨をわかりやすいよう明瞭な記載に更新しました。
- ext4 ファイルシステムは最大 50 TB、そのファイルは最大 16 TB までのサイズに対応できるようになります。チューニング推奨についても適宜更新しました。また、ext2 および ext3 のファイルシステムに対するサポートは ext4 ドライバーで提供されるようになります。
- btrfs ファイルシステムがテクノロジープレビューとして提供されるようになります。
- Red Hat Enterprise Linux 7 では GFS2 関連のマイナーなパフォーマンス改善が行われています。
- **Tuna** が更新され設定ファイルに対応、また **tuned** プロファイルの追加や保存にも対応するようになりました。更新バージョンではイベントベースのサンプリングを使用することでプロセッサーリソースの消費を抑えます。グラフィカルバージョンも更新されリアルタイムのモニタリングが可能になりました。**Tuna** の詳細については「Tuna」、「Tuna を使った CPU、スレッド、割り込みの親和性などの設定」、「Tuna」などで説明しています。
- **tuned** のデフォルトプロファイルが **throughput-performance** になります。**enterprise-storage** プロファイルが削除されその代わりとなります。ネットワーク設定や仮想化に関する新しいプロファイルもいくつか追加されています。また、**tuned** ではシェルスクリプト呼び出しや ***includes*** の機能を提供するようになります。
- **tuned-adm** ツールでは使用しているシステムに適したチューニングプロファイルを勧める **recommend** サブコマンドが提供されるようになります。また、インストール時にシステムにデフォルトのプロファイルを設定するため、そのデフォルトプロファイルに戻る際に使用することもできます。

- Red Hat Enterprise Linux 7 では自動 NUMA バランス機能に対応します。カーネルが自動的にアクティブに使用中のメモリーページのプロセススレッドを検出、そのスレッドとメモリーを複数の NUMA ノードを跨いでグループ化します。スレッドを再スケジュールしてメモリーを移行させ最適な NUMA アライメントとパフォーマンスを得られるようシステムのバランスを取ります。
- ファイルシステムのバリアを有効にした場合でもパフォーマンスの低下が 3% 未満とごくわずかになります。このため **tuned** プロファイルではファイルシステムのバリアを無効にしなくなります。
- OProfile では新しいツール **operf** 使用した Linux Performance Events サブシステムベースのプロファイリングに対応するようになります。このツールは **opcontrol** デーモンの代わりにデータの収集に使用することができます。
- Control グループはシステムでプロセスグループにリソースを割り当てる手段として引き続き使用可能です。Red Hat Enterprise Linux 7 の実装に関する詳細は http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ で 『Red Hat Enterprise Linux 7 Resource Management Guide』 をご覧ください。

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/site/solutions/173863 をご覧ください。

2.1. /proc

proc 「ファイルシステム」は、Linux カーネルの現在の状態を表すファイル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 System
Administrator's Reference Guide
(http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/) 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。

システムモニターの詳細についてはアプリケーションのヘルプまたは Red Hat Enterprise Linux 7 『System Administrator's Guide』 (http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/) を参照してください。

2.3. Performance Co-Pilot (PCP)

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取得や格納、分析を行うツール、サービス、ライブラリーのセット、Performance Co-Pilot (PCP) に対応するようになります。軽量な配信型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。パフォーマンスの測定基準は、Python、Perl、C++ および C インターフェースを使用すると追加できます。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すると利用可能なすべてのパフォーマンスデータを探索することができます。

pcp パッケージではコマンドラインツールおよびベースとなる機能が提供されます。グラフィカルツールの場合は *pcp-gui* パッケージも必要になります。

Red Hat カスタマーポータルには PCP の基本的な使い方に関する記載がいくつか用意されています。索引は https://access.redhat.com/articles/1145953 をご覧ください。

また、*pcp-doc* パッケージでは包括的なドキュメントも用意されています。デフォルトでは **/usr/share/doc/pcp-doc** にインストールされています。PCP では各ツールに対する man ページも提供しています。man ページを表示させるにはコマンドラインで **man *toolname*** と入力します。

2.4. Tuna

Tuna ではスケジューラーのポリシー、スレッドの優先順位、CPU と割り込みの親和性など設定詳細が調整されています。*tuna* ではコマンドラインツールおよび同等の機能性を備えたグラフィカルインターフェースが提供されます。

コマンドラインで Tuna を使用してシステムを設定する方法については「Tuna を使った CPU、スレッド、割り込みの親和性などの設定」で説明しています。Tuna の使い方については「Tuna」または man ページを参照してください。

```
$ man tuna
```

2.5. 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はそのリンクを紹介します。

2.5.1. 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
```

2.5.2. 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
```

2.5.3. 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
```

2.5.4. 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を収集、報告を行います。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
```

2.6. tuned と tuned-adm

tuned はチューニングデーモンです。チューニングプロファイルを設定してオペレーティングシステムが特定の負荷でより効率よく動作するよう適応させます。

一般的な使用事例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-adm 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。事前に用意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 ではテフォルトのプロファイルは **throughput-performance** です。

tuned-adm で与えるプロファイルは節電プロファイルとパフォーマンス強化プロファイルの 2 種類のカテゴリーに分かれます。パフォーマンス強化プロファイルの場合は以下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 を有効にする方法については「tuned」を参照してください。

tuned-adm で提供されるパフォーマンス強化プロファイルについては「tuned-adm」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については Red Hat Enterprise Linux 7 Power Management Guide (http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/) を参照してください。

tuned および tuned-adm の使い方については該当の man ページをご覧ください。

```
$ man tuned
```

```
$ man tuned-adm
```

2.7. 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、記録したり、記録したデータを分析してレポートを作成したりします。

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、Red Hat Enterprise Linux 7 『Developer Guide』 (http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/) でも詳しい説明をお読みいただけます。

2.8. 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 の出力および出力の読み方については「turbostat」を参照してください。

turbostat の詳細については man ページをご覧ください。

```
$ man turbostat
```

2.9. iostat

iostat ツールは *sysstat* パッケージで提供されます。管理側で物理ディスク間の入出力ロードのバランス調整を行う際に便利な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 を最後に開始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量を報告します。iostat の man ページに記載されているパラメーターを使用すると特定のデバイスを集中的に監視することができます。

```
$ man iostat
```

2.10. 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
```

2.11. 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
```

2.12. 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
```

2.13. 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
```

2.14. 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。

SystemTap についての詳細は Red Hat Enterprise Linux 7 『SystemTap Beginner's Guide』 および Red Hat Enterprise Linux 7 『SystemTap TapSet Reference』 を参照してください。いずれのドキュメントも http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ でご覧頂けます。

2.15. 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 System Administrator's Guide』 (http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/) を参照してください。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。

2.16. 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 の詳細については Red Hat Enterprise Linux 7 Developer Guide (http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/) 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。

第3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、「パフォーマンス関連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。

3.1. 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

3.1.1. 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** — によってシステムやアプリケーションのパフォーマンスやシステムのチューニングの際の留意事項に大きく影響を与える可能性があります。

最近のコンピューティングで使用されているとポロジーには主に 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。

3.1.1.1. システムのトポロジーを確認する

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。

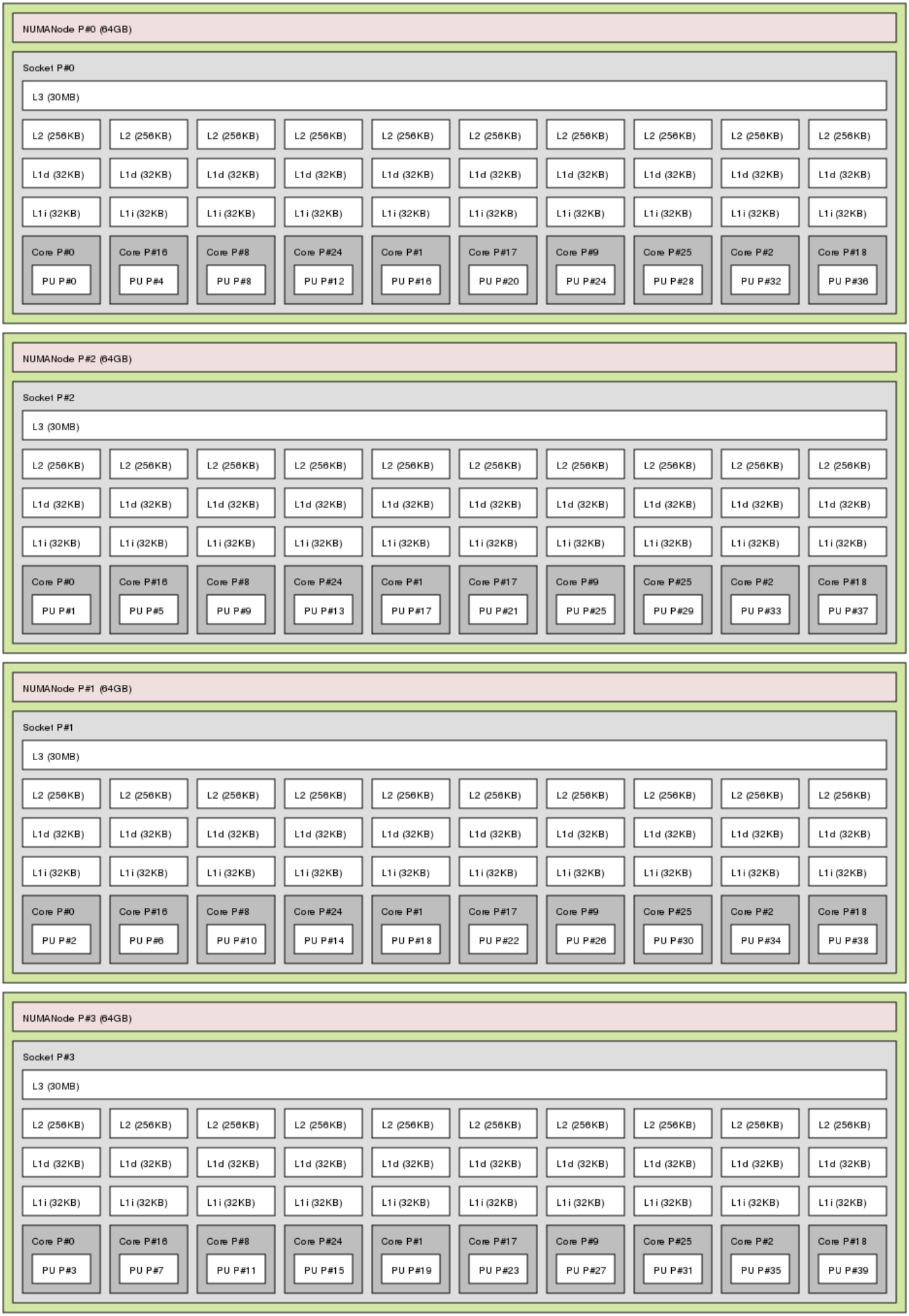
```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
node 1 cpus: 2 6 10 14 18 22 26 30 34 38
node 1 size: 65536 MB
node 1 free: 44321 MB
node 2 cpus: 1 5 9 13 17 21 25 29 33 37
node 2 size: 65536 MB
node 2 free: 44304 MB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

3.1.2. 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 のプロセッサーを無駄なく稼動させるため、スケジューラーはアプリケーションのスレッドの一つをノード B に移動させます。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 で実行しているため、このスレッドに対してノード A のメモリーははローカルメモリーではなくなりますからアクセスに時間がかかるようになります。つまり、このスレッドはノード A のプロセッサーが利用可能になるまで待機させて必要なメモリーがローカルとなる元々のノード A で実行させていた方がノード B に移動して実行を完了するより早かったかもしれません。

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。

3.1.2.1. 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。

3.1.3. 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「割り込みの親和性の設定」または「Tuna を使った CPU、スレッド、割り込みの親和性などの設定」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6章ネットワークを参照してください。

3.2. パフォーマンス関連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。

3.2.1. 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
```

3.2.2. 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、プロセスのメモリーがシステム全体に拡散されているのか特定ノードに集中していうのかを表示します。

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
```

3.2.3. /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「割り込みの親和性の設定」を参照してください。

3.3. 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。

3.3.1. 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 の代替として考えられるスクリプトのひとつに**while :; do d=1; done** があります。また、次のリンク https://dl.fedoraproject.org/pub/epel/6/x86_64/repoview/stress.html で入手できるプログラムも代替プログラムとして使用できます。

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
```

3.3.2. 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
```

3.3.3. taskset でプロセスの親和性を設定する

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
```

3.3.4. numactl で NUMA 親和性を管理する

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、ディスク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
```

3.3.5. 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
```

3.3.6. 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。

3.3.6.1. スケジューリングポリシー

3.3.6.1.1. 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。

3.3.6.1.2. 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。

3.3.6.1.3. 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 を変更する方法については http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ にある 『Red Hat Enterprise Linux 7 Deployment Guide』 を参照してください。

3.3.6.2. 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、割り込みの親和性などの設定」を参照してください。

3.3.7. 割り込みの親和性の設定

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。

特定の割り込み要求の割り込み親和性の値は該当の **/proc/irq/*irq_number*/smp_affinity** ファイルに格納されます。***smp_affinity*** はシステム内の全プロセッサーを表す 16 進数のビットマスクで保存されます。デフォルト値は **f** でシステム内のどのプロセッサーでも割り込み要求を処理できるという意味になります。この値を **1** に設定すると割り込み要求を処理できるのはプロセッサー 0 のみになります。

プロセッサーが 32 以上搭載されているシステムでは 32 ビットグループそれぞれに ***smp_affinity*** 値を区切って設定する必要があります。例えば、64 プロセッサーシステムの最初の 32 プロセッサーにのみ割り込み要求の処理を行わせたい場合は以下を実行します。

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

代わりに、BIOS が NUMA トポロジーをエクスポートする場合は **irqbalance** サービスにその情報を使用させサービスを要求しているハードウェアに対してローカルとなるノードで割り込み要求を処理させることもできます。**irqbalance** の詳細については 「irqbalance」を参照してください。

注記

割り込みステアリングに対応するシステムの場合は割り込み要求の ***smp_affinity*** 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については 6章ネットワーク を参照してください。

3.3.8. Tuna を使った CPU、スレッド、割り込みの親和性などの設定

Tuna を使用すると CPU、スレッド、割り込みの親和性などを管理し、管理対象となるエンティティのタイプに合わせた各種動作を行うことができます。**Tuna** の全機能については「Tuna」を参照してください。

指定 CPU (複数可) から全スレッドを切り離すには次のコマンドを実行します。*CPUs* には分離させる CPU 番号を入力します。

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

特定スレッドの実行が可能 CPU 一覧に任意の CPU を含ませるには次のコマンドを実行します。*CPUs* には含ませる CPU 番号を入力します。

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

割り込み要求を指定 CPU に移動させるには次のコマンドを入力します。*CPUs* には CPU 番号、*IRQs* には移動させる割り込み要求をコンマで区切って入力します。

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

代わりに次のコマンドを使ってパターン **sfc1*** の全割り込み要求を検索することもできます。

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

スレッドのポリシーと優先度を変更するには次のコマンドを実行します。*thread* には変更するスレッド、*policy* にはスレッドに適用するポリシー名、*level* には 0 (最も低い優先度) から 99 (最も高い優先度) の整数をそれぞれ入力します。

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

第4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、「パフォーマンスに関する問題の監視と診断」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。「設定ツール」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。

4.1. 留意事項

Red Hat Enterprise Linux 7 はデフォルトでは適度な作業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使用目的が大量のメモリー量を必要とする場合には仮想マシンの管理方法を変更するとパフォーマンスが改善される場合があります。

4.1.1. 用紙サイズ

物理メモリーはページと呼ばれるまとまりで管理されます。各ページの物理的な位置が仮想の位置にマッピングされプロセッサーがメモリーにアクセスできるようになります。このマッピングはページテーブルと呼ばれるデータ構造に格納されています。

デフォルトの場合、ページサイズは約 4 KB です。デフォルトのページサイズはかなり小さいため、大量のメモリーを管理する場合は多量のページ数が必要になります。しかし、ページテーブルで格納できるアドレスマッピング数には限りがあるため、メモリー要件に応じてパフォーマンスレベルを管理するという点では格納可能なアドレスマッピング数を増加させるのはコストがかかり、また容易ではありません。

Red Hat Enterprise Linux では静的 huge ページで大容量メモリーを管理する機能も提供しています。静的 huge ページは最大 1 GB まで設定することができます。ただし、手作業での管理は困難なため起動時に割り当てを行う必要があります。Red Hat Enterprise Linux 7.1 からはノードベースでの割り当てが可能になります。

静的 huge ページに代わる大規模な自動化ページが transparent huge ページです。サイズは 2 MB でデフォルトで有効になっています。待ち時間に制約のあるアプリケーションに割り込みを行う場合があるため、これが問題となるアプリケーションを使用する場合は無効にすることが一般的です。

アプリケーションのパフォーマンス改善に huge ページを設定する方法については「huge ページの設定」を参照してください。

4.1.2. 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、Linux オペレーティングシステムでは最近使用したアドレスのキャッシュ TLB が提供されます。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 にない場合は (TLB**ミス**) 物理から仮想へのアドレスマッピングを見つけるためページアドレスからの読み込みを行う必要があります。

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。

Red Hat Enterprise Linux は非常に大きなセグメントでメモリーを管理できる HugeTLB (Huge Translation Lookaside Buffer) を備えています。HugeTLB は一度に大量のアドレスマッピングをキャッシュすることができるため TLB ミスの可能性を低減し大量にメモリーを必要とするアプリケーションでパフォーマンスを向上させることができます。

HugeTLB の設定については「huge ページの設定」を参照してください。

4.2. パフォーマンスに関する問題の監視と診断

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。

4.2.1. 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
```

4.2.2. 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
```

4.2.2.1. 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に **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。

4.2.2.2. 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。

4.2.2.3. 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。

4.3. 設定ツール

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
```

永続的に 1 に設定する場合は次のコマンドを実行します。

```
# sysctl vm.overcommit_memory=1
```

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。

4.3.1. huge ページの設定

huge ページはメモリーに近接した領域に依存するため、メモリーが断片化される前の起動時に定義するのが最適です。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 huge 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 huge 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 huge ページはノード間で平等に分割されます。実行中、huge 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 huge 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 huge 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。

また、次のパラメーターを使用して実行時の huge ページの動作に影響を与えることもできます。

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

指定 NUMA ノードに割り当てる指定サイズの huge 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 huge 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
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 huge 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 huge 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 huge ページ数が取得されます。この余剰 huge 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。

4.3.2. 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。

4.3.2.1. 仮想マシンのパラメーター

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

プロセスの **oom_score** を継承する調整プロセスによってプロセスが生成されます。

スワップ

システムが anonymous メモリーやページキャッシュを優先する度合いを制御する **0** から **100** の値です。値が高いとファイルシステムのパフォーマンスが改善されますが RAM 内であまり活発ではないプロセスを積極的にスワップします。値を低くするとメモリー内のプロセスのスワップを避けるため待ち時間は減少しますが 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。

4.3.2.2. 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

カーネルで割り当てられるファイルの最大処理数を定義します。デフォルト値はカーネルの **files_stat.max_files** の値と一致し、***NR_FILE*** (Red Hat Enterprise Linux の場合 8192) か次の結果のいずれか大きい方に設定されます。

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

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。

4.3.2.3. カーネルのパラメーター

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

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。デフォルト値は **65536** です。

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。デフォルト値は **65536** バイトです。

msgmni

メッセージキュー識別子の最大数を定義します (つまりキューの最大数)。64 ビットのアーキテクチャーのシステム場合、デフォルト値は **1985** になります。

shmall

一度にシステム上で使用可能な共有メモリーの総量をページ数で定義します。Red Hat Enterprise Linux の時点ではデフォルト値は **1<<24** または 33554432 ページになります。

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ページ単位で定義します。Red Hat Enterprise Linux の時点ではデフォルト値は **1<<24** または 33554432 バイトになります。

shmmni

システム全体の共有メモリ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数の最大数を定義します。デフォルト値はカーネルパラメーターの ***max_threads*** 値と同等か以下のいずれかになります。

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

最小値は **20** です。

第5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、「パフォーマンス関連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。

5.1. 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。

5.1.1. ソリッドステートディスク (SSD)

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どのデータへのアクセスに対しても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。ストレージ領域としてはギガバイト単位で見るとコスト高の割にストレージ容量がありませんが、HDD と比べて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せ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 を導入するにあたっての推奨事項については http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ の 『Red Hat Enterprise Linux 7 Storage Administration Guide』 を参照してください。

5.1.2. 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にデフォルトとなる I/O スケジューラーです。**Deadline** は要求が I/O スケジューラーに到達した時点からの待ち時間を保証しています。ほとんどの使用目的に適したスケジューラーで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きる環境に適しています。

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ト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。 non-work-conserving?

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 汎用のブロック層でマージされ、 through a simple last-hit cache? これはシンプルな最後にヒットしたもののキャッシュです。CPU にかかる深がシステムが CPU にバインドされていてストレージが高速な場合、これが最良の I/O スケジューラーとなります。CPU-bound systems?

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。

5.1.3. 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。

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 はエクステントベースの割り当てを採用、事前割り当てや遅延割り当てなど断片化を低減しパフォーマンスを支援する数種類の割り当てスキームを備えています。また、メタデータジャーナリング機能にも対応、クラッシュからのリカバリーを容易にしています。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 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ の『Red Hat Enterprise Linux 7 Storage Administration Guide』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。

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 http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ の『Red Hat Enterprise Linux 7 Storage Administration Guide』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。

ext4 のクラスター化割り当て機能 (bigalloc) は現在、検証が行われていず Red Hat Enterprise Linux 7 では対応していません。

Btrfs (テクノロジープレビュー)

Btrfs は拡張性と耐障害性を備えた管理が容易なコピーオンライト (copy-on-write) のファイルシステムです。ビルトインのスナップショットや RAID サポートが収納され、データおよびメタデータのチェックサムを使用してデータ整合性を提供します。また、データ圧縮を用いてパフォーマンスを高め領域をより効率的に使用します。テクノロジープレビューとしてのサポートになり、ファイルシステムの最大サイズは 50 TB です。

デスクトップおよびクラウドストレージに適しています。デバイスの初期フォーマットを行う際に用途に応じて btrfs を調整するのが最適です。

Red Hat Enterprise Linux 7 では btrfs はテクノロジープレビューとしての提供になります。テクノロジープレビューの機能については https://access.redhat.com/site/support/offerings/techpreview/ をご覧ください。

btrfs の管理については http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ の『Red Hat Enterprise Linux 7 Storage Administration Guide』を参照してください。目的別に btrfs を調整する方法については「btrfs のチューニング」を参照してください。

GFS2

GFS2 は High Availability Add-On の一部になります。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 に対して提供しています。1 クラスター内の全サーバーに整合性のあるファイルシステムイメージを提供し、すべてのサーバーが一つの共有ファイルシステムからの読み取りと書き込みを行うことができるようにします。

ファイルシステムは最大 250 TB の サイズまで対応しています。

GFS2 の管理については http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ の 『Red Hat Enterprise Linux 7 Storage Administration Guide』 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。

5.1.4. 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。

5.1.4.1. 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
```

5.1.4.2. 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)。

詳細については http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ の『Red Hat Enterprise Linux 7 Storage Administration Guide』 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
```

5.1.4.3. 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をサポートしています (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/sda/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。

5.2. パフォーマンス関連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。

5.2.1. 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 swap 領域への書き込み (KB 単位)

so

スワップアウト、または swap 領域空の読み取り (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
```

5.2.2. 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
```

5.2.2.1. 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 man ページをご覧ください。

```
$ man blktrace
```

```
$ man blkparse
```

5.2.2.2. btt を使った blktrace 出力の分析

Btt 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。

たとえば、**btt** でブロック層に送信されている要求間の時間 (**Q2Q**) がブロック層で要求が費やしている合計時間 (**Q2C**) より長い場合、I/O サブシステムはパフォーマンスの問題とは関係ないかもしれません。デバイスのサービス処理に費やす時間が長い場合 (**D2C**)、デバイスがオーバーロードしているか、デバイスに送られた作業負荷が最適でない可能性があります。ブロック I/O に要求が割り当てられるまでに長時間キュー待ちしている場合は(**Q2G**) 使用中のストレージで I/O の負荷を処理できないことを示している可能性があります。

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 btt
```

5.2.2.3. 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 数に注目します。毎秒の動作数がデバイスの限界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
```

5.2.3. SystemTap を使ったストレージの監視

『Red Hat Enterprise Linux 7 SystemTap Beginner's Guide』 ではストレージのパフォーマンスをプロファイリングして監視する際に役立つサンプルのスクリプトをいくつか挙げて説明しています。

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。

latencytap.stp スクリプトは、タイプの異なる待ち時間が 1 つ以上のプロセスに与える影響を記録します。30 秒ごとに待ち時間のタイプ一覧をプロセスが待機した時間の合計の多い順に分類して出力します。ストレージとネットワーク両方の待ち時間の原因を特定する際に役立ちます。Red Hat ではこのスクリプトで **--all-modules** オプションを使用して待ち時間イベントのマッピングを有効に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。

http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ で 『Red Hat Enterprise Linux 7 SystemTap Beginner's Guide』 をご覧ください。

5.3. 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。

5.3.1. 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 と **tuned-adm** 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したプロファイルを推奨します。また、インストール時にデフォルトのプロファイルを設定するため、デフォルトのプロファイルに戻る場合にも使用できます。

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。

5.3.2. 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のマウントオプションにスケジューラーを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 I/O スケジューラーを設定するには起動時にカーネルコマンドラインにエレベーターパラメーターを追加するか、**/etc/grub2.conf** ファイルを編集して行います。

```
elevator=scheduler_name
```

5.3.3. デバイスの I/O スケジューラーの設定

特定のストレージデバイスにスケジューラーを設定する、またはスケジューラー優先順序を設定する場合は **/sys/block/*devname*/queue/scheduler** ファイルを編集します。*devname* は設定を行うデバイス名です。

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

5.3.4. 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。

5.3.5. 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。

5.3.5.1. 高速ストレージに cfq をチューニングする

校則の外付けストレージアレイや SSD など、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は **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環境の場合に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/ionice/slice_idle** を **0** に設定します
- **/sys/block/*devname*/queue/ionice/quantum** を **64** に設定します
- **/sys/block/*devname*/queue/ionice/group_idle** を **1** に設定します

5.3.6. noop スケジューラーのチューニング

noop I/O スケジューラーは主に高速ストレージを使用し CPU 処理の占める割合が大きいシステムに便利です。要求はブロック層でマージされるため、**noop** の動作を修正する場合は **/sys/block/sdX/queue/** ディレクトリー配下のファイルでブロック層のパラメーターを編集します。

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。

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。

I/O 要求が内部消去ブロックサイズを超えるとパフォーマンスが低下する SSD があります。このような場合には ***max_hw_sectors_kb*** を内部消去ブロックサイズまで減らすことを推奨しています。

nomerges

要求のマージはほとんどの作業負荷に効果的ですが、デバッグが目的の場合はマージを無効にすると役に立つ場合があります。無効にする場合はこのパラメーターを **0** に設定します。デフォルトでは有効になっています (**1**)。

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にさせるまでにそれぞれ 128 の読み取り要求と 128 の書き込み要求をキュー待ちさせることができます。

待ち時間に影響を受けやすいアプリケーションの場合には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するとライトバック I/O が書き込み要求でデバイスのキューを満杯しなくなります。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、ひとつのプロセスが継続してキューを埋めてしまわないようにします。

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

連続的な読み取り動作を行っている際、ページキャッシュですぐに必要になる可能性が高い情報を格納するためオペレーティングシステムに先読みをさせる容量をキロバイト単位で指定します。***read_ahead_kb*** の値を高くすることでデバイスマッパーに効果を現すことがよくあります。各デバイスにマッピングする値として 128 KB から設定し始めると良いでしょう。

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てしまうものがあります。ご使用の SSD デバイスがこれを自動的に **0** に設定しない場合は、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論理を無効にします。

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。

5.3.7. 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。

5.3.7.1. 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。

5.3.7.1.1. 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5.1「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5.1 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	100000–200000	1000000–2000000
16 KB	100000–1000000	1000000–10000000
64 KB	>1000000	>10000000

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まなくなります。

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プ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
```

5.3.7.1.2. 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。

5.3.7.2. 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。

5.3.7.2.1. 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 を初期化する場合、規模の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
```

5.3.7.2.2. 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
```

5.3.7.3. btrfs のチューニング

Red Hat Enterprise Linux 7.0 からは btrfs がテクノロジープレビューとして提供されます。本セクションは今後のリリースで btrfs が完全対応になると更新されます。

5.3.7.4. 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6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。

6.1. 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。

6.1.1. 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。

6.1.2. 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 「割り込みの親和性の設定」 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。

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。

6.2.1. 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** より **ss** の使用を推奨しています。

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
```

6.2.2. 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
```

6.2.3. 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
```

6.2.4. 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
```

6.2.5. /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。

6.2.6. SystemTap を使ったネットワークの監視

『Red Hat Enterprise Linux 7 SystemTap Beginner's Guide』 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。

詳細については http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ の 『Red Hat Enterprise Linux 7 SystemTap Beginner's Guide』 を参照してください。

6.3. 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。

6.3.1. ネットワークパフォーマンス向けの tuned-adm プロファイル

tuned-adm 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。

6.3.2. ハードウェアバッファの設定

多数のパケットがハードウェアバッファでドロップされてい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

着信トラフィックにフィルターをかける、参加するマルチキャストグループ数を減らす、キュー待ちの割合を減らすためブロードキャストのトラフィック量を減らすなどの選択肢があります。着信トラフィックにフィルターをかける方法については 『Red Hat Enterprise Linux 7 Security Guide』 を参照してください。マルチキャストグループについては 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については 『Red Hat Enterprise Linux 7 System Administrator's Reference Guide』 または設定するデバイスに関連するドキュメントを参照してください。Red Hat Enterprise Linux 7 に関連するドキュメントはすべて http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ でご覧いただくことができます。

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。

6.3.3. 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。

6.3.3.1. 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。

6.3.4. 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。

着信トラフィックの速度を低下させる

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。

6.3.4.1. 着信トラフィックの速度を低下させる

着信トラフィックにフィルターをかける、または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低くして着信トラフィックのの速度を遅くします。着信トラフィックにフィルターをかける方法については http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ の 『Red Hat Enterprise Linux 7 Security Guide』 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。

6.3.4.2. 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

このパラメーターでソケットで使用される受信バッファのデフォルトサイズを管理します。指定する値は **/proc/sys/net/core/rmem_max** の値より小さくしてください。

setsockopt を使って SO_RCVBUF に大きな値を設定する

このパラメーターでソケットの受信バッファの最大サイズを管理します (バイト単位)。**getsockopt** システムコールを使ってバッファの現在の値を確定します。このパラメーターの詳細については man ページをご覧ください。

```
$ man 7 socket
```

6.3.5. 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** インターフェイスに 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 ドライバーの中では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。

6.3.6. 受信パケットテアリング (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、CPU のビットマップ上の位置の値を 1 に設定します。たとえば、CPU 0、1、2、および 3 で割り込みを処理する場合は **rps_cpus** の値を **00001111** (1+2+4+8)、もしくは **f** (16進数で 15 の値) に設定します。

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。

6.3.7. 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。

6.3.8. 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. Tuna

Tuna を使ってプロセッサーやスケジューリング親和性を管理することができます。本セクションではコマンドラインインターフェースについて説明していますが、同様の機能を備えたグラフィカルインターフェースもあります。グラフィカルユーティリティーはコマンドラインで **tuna** を実行すると起動します。

Tuna は順番に処理される数種類のコマンドラインパラメーターを受け取ります。次のコマンドでは負荷を 4 つのソケットシステムに分散しています。

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

--gui

グラフィカルユーザーインターフェースを起動します。

--cpus

Tuna で管理する CPU をコンマで区切った一覧をとります。新しい一覧を指定するまでこの一覧が有効になります。

--config_file_apply

システムに適用するプロファイル名をとります。

--config_file_list

プレロードされるプロファイルを表示します。

--cgroup

--show_threads と併用します。管理グループを有効にした場合に***--show_threads*** で表示されるプロセスが属する管理グループタイプを表示します。

--affect_children

指定すると **Tuna** は子スレッド、親スレッド両方に影響します。

--filter

影響を受けるエントリーのみを表示するよう表示にフィルターをかけます。

--isolate

CPU のコンマで区切った一覧をとります。**Tuna** により指定 CPU からの全スレッドの移行が行われます。

--include

CPU のコンマで区切った一覧をとります。Tuna により指定 CPU での全スレッドの実行が許可されます。

--no_kthreads

このパラメーターを指定すると Tuna はカーネルスレッドには作用しなくなります。

--move

選択したエンティティーを指定 CPU に移動します。

--priority

スレッドの優先度とスケジューラーのポリシーを指定します。使用できるスケジューラーポリシーは **OTHER**、**FIFO**、**RR**、**BATCH**、**IDLE** になります。

ポリシーが **FIFO** または **RR** の場合、有効な優先度の値は 1 (最も低い) から 99 (最も高い) の間の整数になります。デフォルト値は **1** です。たとえば、**tuna --threads 7861 --priority=RR:40** の場合、ポリシーには **RR** (ラウンドロビン)、優先度には **40** がスレッド **7861** に設定されます。

ポリシーが **OTHER**、**BATCH**、**IDLE** のいずれかになる場合は有効な優先度の値は **0** です。

--show_threads

スレッド一覧を表示します。

--show_irqs

IRQ 一覧を表示します。

--irqs

Tuna を作用させる IRQ のコンマで区切った一覧をとります。新しい一覧を指定するまでこの一覧が有効になります。一覧に IRQ を追加する場合は **+**、一覧から削除する場合は **-** を使用します。

--save

指定ファイルにカーネルスレッドのスケジュールを保存します。

--sockets

Tuna に管理させる CPU ソケットのコンマで区切った一覧をとります。このオプションはひとつのプロセッサーキャッシュを共有するコアや同じ物理チップにあるコアなどシステムのテクノロジーを考慮に入れています。

--threads

Tuna に管理させるスレッドのコンマで区切った一覧をとります。新しい一覧を指定するまでこの一覧が有効になります。一覧にスレッドを追加する場合は **+**、一覧から削除する場合は **-** を使用します。

--no_uthreads

動作がユーザーのスレッドに作用しないようにします。

--what_is

選択したエンティティーの詳細なヘルプを表示します。

--spread

--threads で指定したスレッドを **--cpus** で指定した CPU 間で均等に分散します。

A.3. 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.4. 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.5. 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。また***update_interval*** パラメーターで tuned のチェック使用とチューニング詳細の更新の間隔を秒単位で設定することもできます。

tuned の詳細については man ページをご覧ください。

```
$ man tuned
```

A.6. tuned-adm

tuned-adm はいくつか特定の使用事例でパフォーマンスを改善できる数種類のプロファイルを提供しているコマンドラインツールです。また、システムの評価を行い推奨チューニングプロファイルを出力するサブコマンド (**tuned-adm recommend**) も提供しています。このサブコマンドはインストール時にシステムのデフォルトプロファイルを設定するためデフォルトのプロファイルに戻りたい場合に使用することが

できます。

Red Hat Enterprise Linux 7 からはチューニングプロファイルの有効化または無効化の一部として **tuned-adm** にコマンドを実行できる機能が含まれるようになります。これにより適用するチューニングプロファイルを選択する前にシステムがマスターのデータベースノードかどうかをチェックするなど **tuned-adm** では利用できない環境固有のチェックを追加できるようになります。

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既存のプロファイルに独自の **tuned-adm** プロファイルをベースできるようになります。

tuned-adm では以下のようなチューニングプロファイルが提供され、また Red Hat Enterprise Linux 7 で対応しています。

throughput-performance

処理能力の改善に焦点をあてたサーバープロファイルになります。デフォルトのプロファイルでほとんどのシステムに推奨となります。

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 huge ページを有効にし、**cpupower** を使って **performance** cpufreq ガバナーを設定、入出力スケジューラー **deadline** に設定します。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 チューニングや透過的な huge ページの 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 huge 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。

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 huge ページと NUMA 自動負荷分散を無効にします。また、**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。さらに ***busy_read*** と ***busy_poll*** の時間を **50** µs に ***tcp_fastopen*** を **3** に設定します。

network-throughput

ネットワーク処理能力の改善に焦点をあてたサーバープロファイルです。

intel_pstate と **max_perf_pct=100** を設定しカーネルのネットワークバッファサイズを大きくして節電よりパフォーマンスを重視します。透過的な huge 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 huge 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 huge 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。

tuned-adm で提供される節電プロファイルについては 『Red Hat Enterprise Linux 7 Power Management Guide』 (http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/) を参照してください。

tuned-adm の使い方については man ページをご覧ください。

```
$ man tuned-adm
```

A.7. perf

perf ツールは便利なコマンドを数多く備えています。その一部を本セクションで説明しています。**perf** の詳細については 『Red Hat Enterprise Linux 7 Developer Guide』 (http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/) または man ページをご覧ください。

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.8. 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。ツールに関する詳細は man ページをご覧ください。コマンドラインで **man *toolname*** (*toolname* にはツール名を入力) と入力します。

pcp-doc パッケージは詳細なドキュメントをデフォルトでは **/usr/share/doc/pcp-doc** ディレクトリーにインストールします。

Red Hat カスタマーポータルには PCP の基本的な使い方に関する記載がいくつか用意されています。索引は https://access.redhat.com/articles/1145953 をご覧ください。

A.9. 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** (1000000 バイト)、**M** (1048576 バイト) の値が使用できます。

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.10. 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.11. 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

CPU が c0 状態だったあいだのクロック平均速度

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.12. 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.13. 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.14. 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.14.1. コマンドラインから numad を使用する

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、アクティビティが **/var/log/numad.log** にログ記録されます。アクティビティは、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.14.2. numad をサービスとして使用する

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.14.3. 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.14.4. 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.15. 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下さい。

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.16. 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.17. SystemTap

SystemTap は専用ガイドで詳細に説明されています。Red Hat Enterprise Linux 7 バージョンの『SystemTap Beginner's Guide』 および 『SystemTap TapSet Reference』 は、いずれも http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ で見ることができます。

付録B 改訂履歴

改訂 0.3-24.2 **Wed Nov 18 2015** **Noriko Mizumoto**
翻訳完成

改訂 0.3-24.1 **Wed Nov 18 2015** **Kenzo Moriguchi**
翻訳ファイルを XML ソースバージョン 0.3-24 と同期

改訂 0.3-24 **Mon Feb 23 2015** **Laura Bailey**
ビジーポーリングサポートの確認についての詳細を訂正 (BZ1080703)。

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

改訂 0.3-22 **Tue Feb 17 2015** **Laura Bailey**
ビジーポーリングサポートのチェックを追加 (BZ1080703)。

改訂 0.3-21 **Thu Feb 05 2015** **Laura Bailey**
新たな調整プロファイルパラメーター cmdline に言及 (BZ1130818)。
swapon コマンドの新規 discard パラメーターに関する「7.1 の新機能」セクションに注意点を追加 (BZ1130826)。

改訂 0.3-20 **Fri Jan 09 2015** **Charles Boyle**
pgrep コマンドのエラーを訂正 (BZ1155253)。
vm.swappiness の説明を修正 (BZ1148419)。

改訂 0.3-19 **Fri Dec 05 2014** **Laura Bailey**
スプラッシュページの sort_order を更新。

改訂 0.3-18 **Wed Nov 26 2014** **Laura Bailey**
7.1 Beta の PCP 記事索引へのリンクを追加 (BZ1083387)。

改訂 0.3-15 **Mon Nov 24 2014** **Laura Bailey**
アイドルバランシング変更の注意点を追加 (BZ1131851)。

改訂 0.3-14 **Wed Nov 19 2014** **Laura Bailey**
huge ページ割り当ての詳細を更新 (BZ1131367)。
ランタイム時に huge ページを割り当てる方法についての例を追加 (BZ1131367)。

改訂 0.3-12 **Thu Nov 13 2014** **Laura Bailey**
RHEL 7.1 での SHMALL および SHMMAX の新規デフォルト値を追加 (BZ1131848)

改訂 0.3-11 **Mon Nov 10 2014** **Laura Bailey**
ext4 のクラスター化割り当て機能 (bigalloc) のサポート状況についての注意点を追加 (BZ794607)。

改訂 0.3-10 **Fri Oct 31 2014** **Laura Bailey**
ノードごとの静的 huge ページについて説明 (BZ1131832)。

改訂 0.3-9 **Tue Jul 22 2014** **Laura Bailey**
latencytap.stp スクリプトについての説明を追加 (BZ988155)。

改訂 0.3-7 **Thu Jun 26 2014** **Laura Bailey**
CPU の章の誤字を修正。
I/O スケジューラーの調整代替への言及を削除。

改訂 0.3-5 **Wed Jun 11 2014** **Laura Bailey**
リダイレクトしない access.redhat.com リンクに末尾のスラッシュを追加。

改訂 0.3-4 **Tue Jun 10 2014** **Laura Bailey**
irqbalance 付録に割り込みおよび CPU 禁止の詳細を追加 (BZ852981)。

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

改訂 0.3-2 **Mon Apr 07 2014** **Laura Bailey**
RT@294949 用に本書の構成を更新。

改訂 0.2-38 **Mon Apr 07 2014** **Laura Bailey**
OProfile の更新データを追加 (BZ955882)
古くなったコメントを削除。

改訂 0.2-34 **Fri Apr 04 2014** **Laura Bailey**
lstopo 出力の画像スタイルを修正 (BZ1042800)。
irqbalance デーモンの詳細を追加 (BZ955890)。
制御グループについての詳細を追加、修正 (BZ794624)。
PCP についての詳細を追加 (BZ955883)。
XFS チューニングの詳細を更新 (BZ794616)。
OProfile の更新データを追加 (BZ955882)

改訂 0.2-27 **Fri Mar 28 2014** **Laura Bailey**
Jeremy Eder からのフィードバックに基づいて busy_poll セクションを修正 (RT276607)。
nohz_full セクションを修正し、Jeremy Eder からのフィードバックに基づいて詳細を追加 (RT284423)
SystemTap セクションに詳細を追加 (BZ955884)。
SSD セクションに詳細を追加 (BZ955900)。
tuned-adm recommend コマンドに関する詳細を追加 (BZ794623)。
機能セクションの自動 NUMA バランシングについての注意点を修正 (BZ794612)。
用語問題および 新規画像を含む NUMA に関する出力例の問題を修正 (BZ1042800)。
Jeremy Eder からのフィードバックに基づいて RSS と連結した irqbalance についての詳細を修正。

改訂 0.2-19 **Fri Mar 21 2014** **Laura Bailey**
メモリーの章に透過的 huge ページについての詳細を追加 (BZ794621)。
NUMA ノードに関する用語の使用を修正 (BZ1042800)。
kernel の制限を更新 (BZ955894)。
tickless カーネルセクションを下書き (RT284423)。
ビジーポーリングセクションを下書き (RT276607)。
ファイルシステムバリについての情報を更新。
ノードごとの huge ページ割り当てで不明瞭な情報を削除。将来に有用な情報を追加するために BZ1079079 を作成。
ソリッドステートディスクについての詳細を追加 (BZ955900)。
見直しマーカーを削除。

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

Jeremy Eder および Joe Mario からのフィードバックを適用。
Tuna GUI への更新に言及 (BZ955872)。
ネットワークの章およびツールについての付録に SystemTap の詳細を追加 (BZ955884)。

改訂 0.2-12 **Fri Mar 07 2014** **Laura Bailey**
自動 NUMA 移行のサポートについて言及 (BZ794612)。
Jeremy Eder からのフィードバックを適用。

改訂 0.2-11 **Fri Mar 07 2014** **Laura Bailey**
Jeremy Eder からのフィードバックを適用。

改訂 0.2-10 **Mon Feb 24 2014** **Laura Bailey**
Lukáš Czerner からのフィードバックに基づいて Ext4 の情報を修正 (BZ#794607)。

改訂 0.2-9 **Mon Feb 17 2014** **Laura Bailey**
Bill Gray からのフィードバックに基づき CPU の章を修正。
Bill Gray からのフィードバックに基づきメモリーの章とツールの付録を修正、追加。

改訂 0.2-8 **Mon Feb 10 2014** **Laura Bailey**
isolcpus 起動パラメーターを CPU の章に追加 (RT276607)。
SME フィードバックに基づきパラメーターの説明を修正し、新規パラメーターを追加 (BZ#970844)。
推奨される tuned-adm プロファイルをネットワークの章に追加。
フラグセクションに見直しのための備考を追加。

改訂 0.2-4 **Mon Feb 03 2014** **Laura Bailey**
numactl --membind パラメーターが文書化されていることを確認 (BZ#922070)。
Tuna についての詳細をツールの導入、CPU の章、ツールの付録に追加 (BZ#970844)。
ストレージとファイルシステムの章の構成エラーを修正。
相互参照を追加。

改訂 0.2-2 **Fri Jan 31 2014** **Laura Bailey**
書き直してから再構築を完了。
本ガイド内のツールがすべて、それを提供するパッケージと記述されていることを確認。

改訂 0.1-11 **Thu Dec 05 2013** **Laura Bailey**
RHEL 7.0 ベータ用にガイドを再構成。

改訂 0.1-10 **Wed Nov 27 2013** **Laura Bailey**
ベータ前のカスタムビルド。

改訂 0.1-9 **Tue Oct 15 2013** **Laura Bailey**
カスタマーフィードバックに基づくマイナーな修正 (BZ#1011676)

改訂 0.1-7 **Mon Sep 09 2013** **Laura Bailey**
RHEL 6.5 からの新コンテンツをマージ。
編集者のフィードバックを適用。

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

ext4 ファイルシステムの制限を更新 (BZ#794607)。
64-bit ファイルシステムの理論上の最大値を修正。
パフォーマンス関連の変更を追跡するために新機能のセクションを追加。
デフォルトの I/O スケジューラーを **cfq** から **deadline** に変更 (BZ#794602)。
BTRFS チューニングのドラフトコンテンツを追加 (BZ#794604)。
XFS セクションを更新し、ディレクトリーのブロックサイズについての推奨事項を明確化、さらに XFS 対応制限を更新 (BZ#794616)。

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

RHEL 7 ドラフトとして更新および公開。

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

RHEL 6.4 バージョンから分岐。