Red Hat Enterprise Linux 7

仮想化のチューニングと最適化ガイド

RHEL 上のホストシステムと仮想化ゲストでの KVM パフォーマンス機能の使用

Last Updated: 2023-03-25

Red Hat Enterprise Linux 7 仮想化のチューニングと最適化ガイド

RHEL 上のホストシステムと仮想化ゲストでの KVM パフォーマンス機能の使用

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

法律上の通知

概要

Red Hat Enterprise Linux 仮想化のチューニングと最適化ガイドでは、KVM と仮想化のパフォーマンスについて説明しています。このガイドでは、ホストシステムと仮想化ゲストの KVM パフォーマンス機能とオプションを最大限に活用するためのヒントと提案を見つけることができます。

目次

第1章 はじめに

1.1. 仮想化でパフォーマンスの最適化が重要な理由

KVM 仮想化では、ゲストはホストマシン上のプロセスによって表されます。これは、ホストの処理能力、メモリー、およびその他のリソースを使用して、ゲストの仮想ハードウェアの機能をエミュレートすることを意味します。

ただし、ゲストハードウェアは、ホストよりもリソースの使用効果が低くなる可能性があります。したがって、ゲストが期待される速度でタスクを実行するには、割り当てられるホストリソースの量を調整する必要がある場合があります。さらに、さまざまなタイプの仮想ハードウェアに異なるレベルのオーバーヘッドがある可能性があるため、適切な仮想ハードウェア設定がゲストのパフォーマンスに大きな影響を与える可能性があります。状況によっては、特定の設定を行うことで、仮想マシンによるホストリソースの使用効率が向上します。

1.2. KVM パフォーマンスアーキテクチャーの概要

次のポイントは、システムパフォーマンス、およびプロセスとスレッドの管理に関連する KVM の概要を示しています。

- KVM を使用する場合、ゲストはホスト上で Linux プロセスとして実行されます。
- 仮想 CPU (vCPU) は通常のスレッドとして実装され、Linux スケジューラーで処理されます。
- ゲストは、カーネルから NUMA や Huge Page などの機能を自動的に継承しません。
- ホストのディスクおよびネットワーク I/O の設定は、仮想マシンのパフォーマンスに大きく影響します。
- ネットワークトラフィックは通常、ソフトウェアベースのブリッジを通過します。
- デバイスとそのモデルによっては、その特定のハードウェアのエミュレーションにより、オーバーヘッドが著しくなる可能性があります。

1.3. 仮想化のパフォーマンスの機能と改善

Red Hat Enterprise Linux 7 での仮想化パフォーマンスの向上

次の機能により、Red Hat Enterprise Linux 7 の仮想化パフォーマンスが向上します。

自動 NUMA バランシング

自動 NUMA バランシングにより、Red Hat Enterprise Linux 7 ゲストを手動で調整しなくても、NUMA ハードウェアシステムで実行されているアプリケーションのパフォーマンスが向上します。自動 NUMA バランシングは、スレッドまたはプロセスである可能性のあるタスクを、アクセスしているメモリーの近くに移動します。これにより、ゼロ設定で優れたパフォーマンスが可能になります。ただし、状況によっては、より正確なゲスト設定を提供したり、CPU やメモリーのアフィニティーをホストするようにゲストを設定したりすると、より良い結果が得られる場合があります。

自動 NUMA バランシングの詳細については、「自動 NUMA バランシング」 を参照してください。

VirtIO モデル

virtio モデルを備えた仮想ハードウェアには、すべての特殊性を備えたハードウェアをエミュレートするオーバーヘッドはありません。VirtIO デバイスは、仮想化環境で使用するために特別に設計さ

れているため、オーバーヘッドが低くなります。ただし、すべてのゲストオペレーティングシステムがそのようなモデルをサポートしているわけではありません。

マルチキュー virtio-net

パケットの送受信処理をゲストの使用可能な vCPU の数に合わせてスケーリングできるネットワークアプローチ。

マルチキュー virtio-net の詳細については、「マルチキュー virtio-net」 を参照してください。

ブリッジのゼロコピー送信

ゼロコピー送信モードでは、スループットに影響を与えることなく、ゲストネットワークと外部ネットワーク間で大きなパケットを送信する際のホスト CPU オーバーヘッドが最大 15% 削減されます。ブリッジのゼロコピー送信は、Red Hat Enterprise Linux 7 仮想マシンで完全にサポートされていますが、デフォルトでは無効になっています。

ゼロコピー送信の詳細については、「ブリッジのゼロコピー送信」 を参照してください。

APIC 仮想化 (APICv)

新しい Intel プロセッサーは、Advanced Programmable Interrupt Controller (APICv) のハードウェア仮想化を提供します。APICv は、ゲストが APIC に直接アクセスできるようにすることで、仮想化された AMD64 および Intel 64 ゲストのパフォーマンスを向上させ、APIC によって引き起こされる割り込みレイテンシーと仮想マシンの終了数を大幅に削減します。この機能は、新しい Intel プロセッサーでデフォルトで使用され、I/O パフォーマンスを向上させます。

EOI アクセラレーション

仮想 APIC 機能のない古いチップセットでの高帯域幅 I/O の割り込み終了アクセラレーション。

マルチキュー virtio-scsi

virtio-scsi ドライバーのマルチキューサポートによって提供されるストレージパフォーマンスとスケーラビリティーの向上。これにより、各仮想 CPU に別個のキューを持たせることが可能になります。また仮想 CPU は、その他の vCPU に影響を及ぼすことなく使用するために、割り込みできるようになります。

マルチキュー virtio-scsi の詳細については、「マルチキュー virtio-scsi」 を参照してください。

準仮想化チケットロック

準仮想化チケットロック (pvticketlocks) は、オーバーサブスクライブされた CPU を搭載した Red Hat Enterprise Linux 7 ホストで実行されている Red Hat Enterprise Linux 7 ゲスト仮想マシンのパフォーマンスを向上させます。

準仮想化ページフォールト

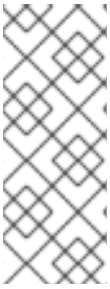
準仮想化ページフォールトは、ホストによってスワップアウトされたページにアクセスしようとすると、ゲストにインジェクトされます。これにより、ホストメモリーがオーバーコミットされ、ゲストメモリーがスワップアウトされた場合の KVM ゲストのパフォーマンスが向上します。

準仮想化時間 vsyscall の最適化

gettimeofday および **clock_gettime** システムコールは、**vsyscall** メカニズムを介してユーザースペースで実行されます。以前は、これらのシステムコールを発行するには、システムをカーネルモードに切り替えてから、ユーザースペースに戻す必要がありました。これにより、一部のアプリケーションのパフォーマンスが大幅に向上します。

Red Hat Enterprise Linux の仮想化パフォーマンス機能

- CPU/カーネル
  - NUMA - Non-Uniform Memory Access。NUMA の詳細については、9章*NUMA* を参照してください。
  - CFS - Completely Fair Scheduler。最新のクラス重視のスケジューラー。
  - RCU - Read Copy Update。共有スレッドデータの処理が改善されます。
  - 最大 160 個の仮想 CPU (vCPU)。
- メモリー
  - メモリーを大量に消費する環境向けのヒュージページやその他の最適化。詳しくは 8章*メモリー* を参照してください。
- ネットワーク
  - vhost-net - 高速なカーネルベースの VirtIO ソリューション。
  - SR-IOV - ネイティブに近いネットワークパフォーマンスレベル用。
- ブロック I/O
  - AIO - 他の I/O 操作とオーバーラップするスレッドのサポート。
  - MSI - PCI バスデバイスの割り込み生成。
  - ディスク I/O スロットリング - ゲストディスク I/O 要求を制御して、ホストリソースの過剰使用を防ぎます。詳しくは 「ディスク I/O スロットリング」 を参照してください。



注記

仮想化のサポート、制限、および機能の詳細については、『Red Hat Enterprise Linux 7 Virtualization Getting Started Guide』 および以下の URL を参照してください。

https://access.redhat.com/certified-hypervisors

https://access.redhat.com/articles/rhel-kvm-limits

第2章 パフォーマンス監視ツール

この章では、ゲスト仮想マシン環境のモニターリングに使用されるツールについて説明します。

2.1. PERF KVM

kvm オプションを指定した **perf** コマンドを使用して、ホストからゲストオペレーティングシステムの統計情報を収集および分析できます。perf パッケージは **perf** コマンドを提供します。次のコマンドを実行してインストールします。

```
# yum install perf
```

In order to use **perf kvm** in the host, you must have access to the **/proc/modules** ファイルと **/proc/kallsyms** ファイルにアクセスできる必要があります。ファイルをホストに転送し、ファイルに関するレポートを実行する場合は、手順2.1「ゲストからホストへの /proc ファイルのコピー」 を参照してください。

手順2.1 ゲストからホストへの /proc ファイルのコピー

重要

必要なファイルを直接コピーする場合 (たとえば、**scp** を使用して)、長さがゼロのファイルのみをコピーします。この手順では、最初にゲスト内のファイルを一時的な場所に保存し (**cat** コマンドを使用)、次にそれらをホストにコピーして **perf kvm** で使用する方法について説明します。

1. **ゲストへのログインとファイルの保存**
   ゲストにログインし、**/proc/modules** と **/proc/kallsyms** を一時的な場所 **/tmp** に保存します。

   ```
   # cat /proc/modules > /tmp/modules
   # cat /proc/kallsyms > /tmp/kallsyms
   ```

2. **ホストへの一時ファイルのコピー**
   ゲストからログオフしたら、次の **scp** コマンドの例を実行して、保存されたファイルをホストにコピーします。ホスト名と TCP ポートが異なる場合は、それらを置き換える必要があります。

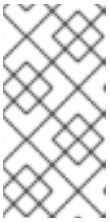
   ```
   # scp root@GuestMachine:/tmp/kallsyms guest-kallsyms
   # scp root@GuestMachine:/tmp/modules guest-modules
   ```

   これで、ホスト上のゲストからの 2 つのファイル (**guest-kallsyms** と **guest-modules**) が作成され、**perf kvm** で使用できるようになりました。

3. **perf kvm を使用したイベントの記録とレポート**
   前の手順で取得したファイルを使用して、ゲスト、ホスト、またはその両方のイベントの記録とレポートが可能になりました。

   次のコマンド例を実行します。

   ```
   # perf kvm --host --guest --guestkallsyms=guest-kallsyms \
   --guestmodules=guest-modules record -a -o perf.data
   ```



注記

コマンドで **--host** と **--guest** の両方が使用されている場合、出力は **perf.data.kvm** に保存されます。**--host** のみが使用されている場合、ファイルの名前は **perf.data.host** になります。同様に、**-guest** のみが使用されている場合、ファイルの名前は **perf.data.guest** になります。

Ctrl-C を押すと、記録が停止します。

4. **イベントのレポート**
次のコマンド例は、記録プロセスによって取得されたファイルを使用し、出力を新しいファイル **analyze** にリダイレクトします。

```
perf kvm --host --guest --guestmodules=guest-modules report -i perf.data.kvm \
--force > analyze
```

analyze ファイルのコンテンツを表示して、記録されたイベントを調べます。

```
# cat analyze


# Events: 7K cycles
#
# Overhead       Command  Shared Object      Symbol
# ........  ............  .................  ........................
#
    95.06%            vi  vi                 [.] 0x48287
     0.61%          init  [kernel.kallsyms]  [k] intel_idle
     0.36%            vi  libc-2.12.so       [.] _wordcopy_fwd_aligned
     0.32%            vi  libc-2.12.so       [.] __strlen_sse42
     0.14%       swapper  [kernel.kallsyms]  [k] intel_idle
     0.13%          init  [kernel.kallsyms]  [k] uhci_irq
     0.11%          perf  [kernel.kallsyms]  [k] generic_exec_single
     0.11%          init  [kernel.kallsyms]  [k] tg_shares_up
     0.10%      qemu-kvm  [kernel.kallsyms]  [k] tg_shares_up

[output truncated...]
```

2.2. 仮想パフォーマンス監視ユニット (VPMU)

仮想パフォーマンスモニターリングユニット (vPMU) は、ゲスト仮想マシンがどのように機能しているかを示す統計を表示します。

仮想パフォーマンスモニターリングユニットを使用すると、ユーザーはゲスト仮想マシンで発生する可能性のあるパフォーマンス問題の原因を特定できます。vPMU は、Intel の PMU (Performance Monitoring Units) に基づいており、Intel マシンでのみ使用できます。

この機能は、Red Hat Enterprise Linux 6 または Red Hat Enterprise Linux 7 を実行しているゲスト仮想マシンでのみサポートされ、デフォルトでは無効になっています。

システムで vPMU がサポートされているかどうかを確認するには、次のコマンドを実行して、ホスト CPU の **arch_perfmon** フラグを確認します。

```
# cat /proc/cpuinfo|grep arch_perfmon
```

vPMU を有効にするには、ゲスト XML で ***cpu mode*** を **host-passthrough** として指定します。

```
# virsh dumpxml guest_name |grep "cpu mode"
<cpu mode='host-passthrough'>
```

vPMU を有効にした後、ゲスト仮想マシンから **perf** コマンドを実行して、仮想マシンのパフォーマンス統計を表示します。

2.3. VIRTUAL MACHINE MANAGER でのパフォーマンスのモニターリング

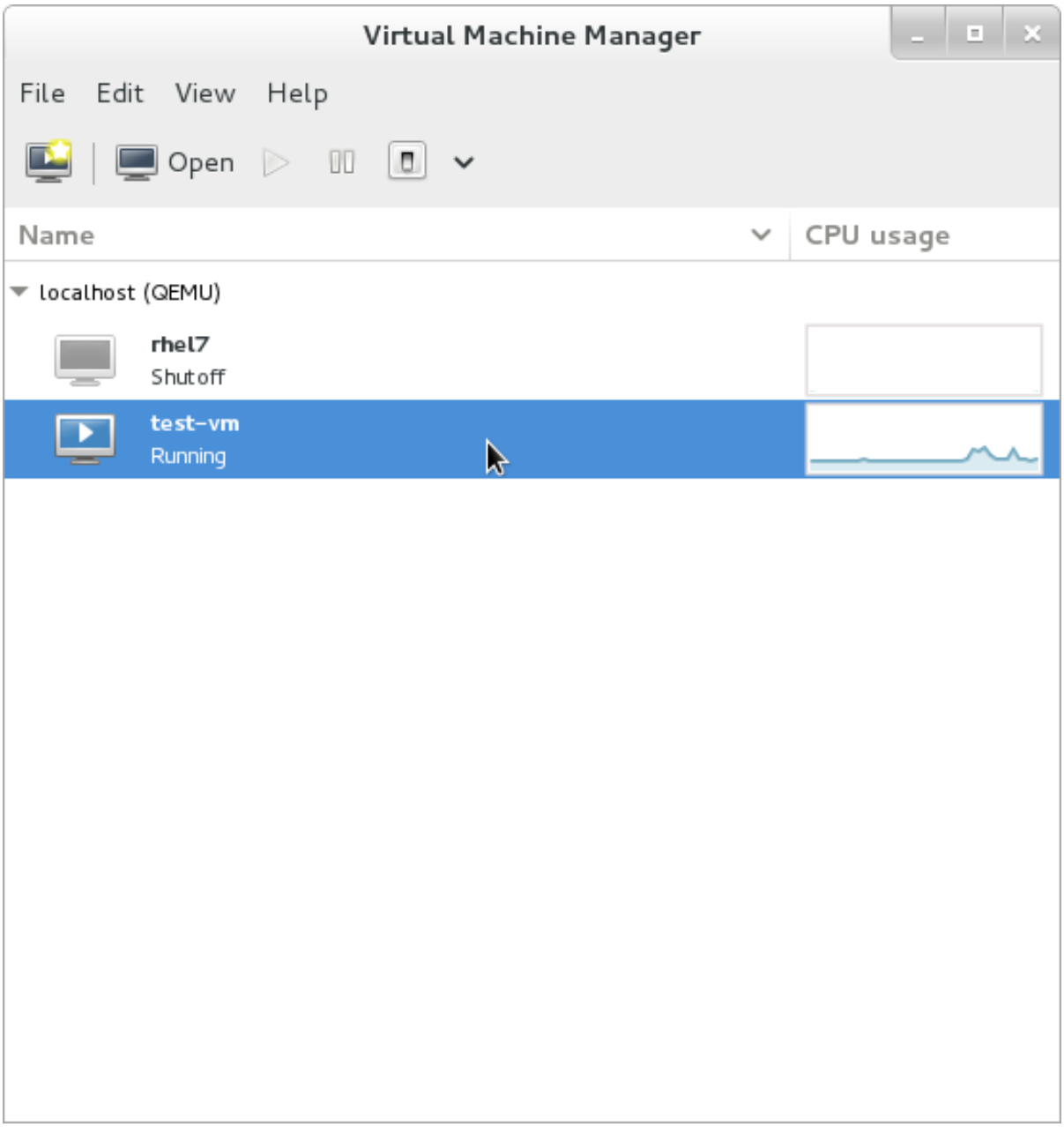
仮想マシンモニターを使用して、システム上の任意の仮想マシンのパフォーマンス情報を表示できます。Virtual Machine Manager に表示されるパフォーマンス情報を設定することもできます。

2.3.1. Virtual Machine Manager でのパフォーマンス概要の表示

Virtual Machine Manager を使用して仮想マシンのパフォーマンス概要を表示するには、次の手順に従います。

1. Virtual Machine Manager のメインウィンドウで、表示する仮想マシンを強調表示します。

図2.1 表示する仮想マシンの選択



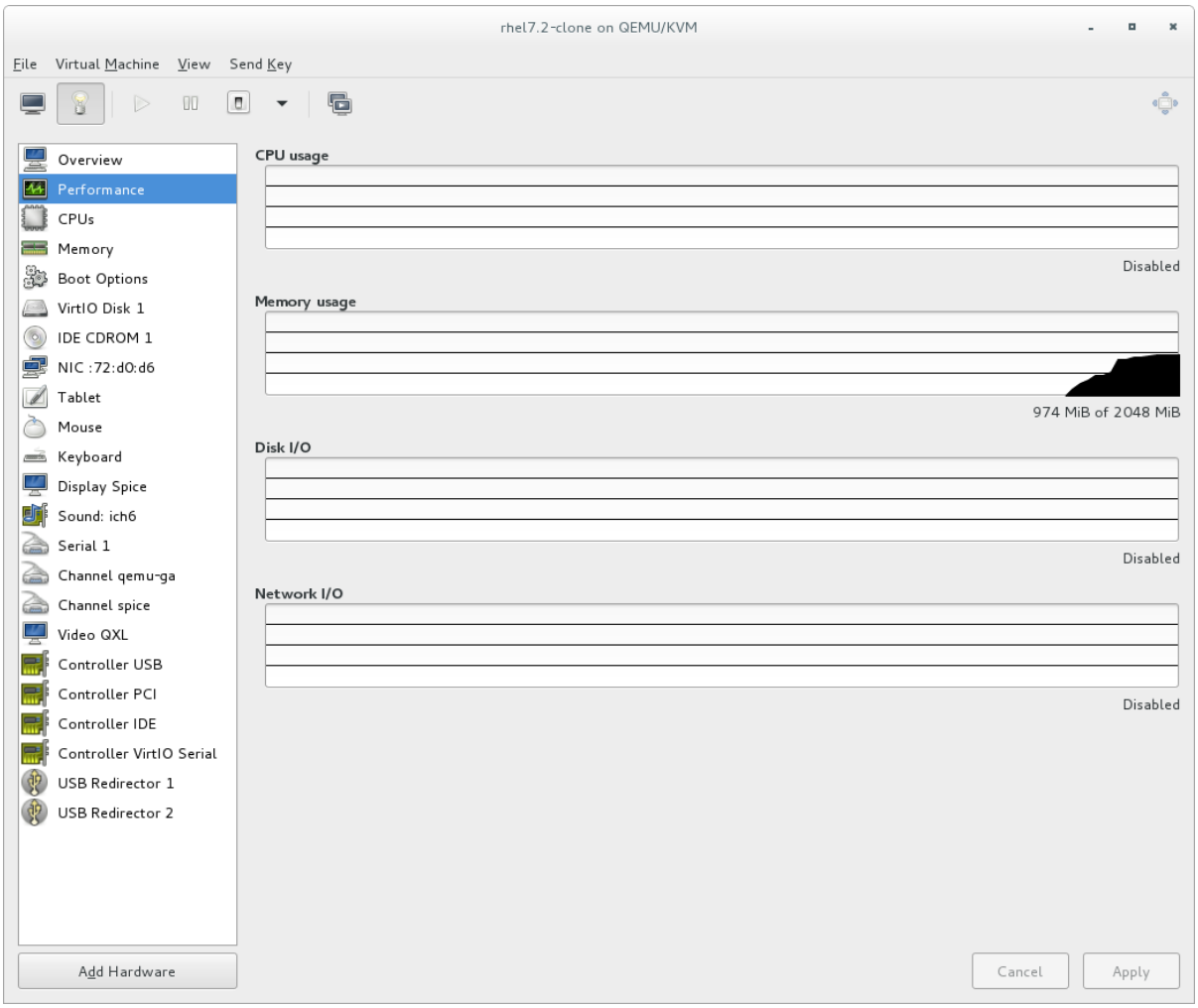
2. 仮想マシンマネージャーの**Edit** メニューから、**Virtual Machine Details** を選択します。

   仮想マシンの詳細ウィンドウが開くと、コンソールが表示される場合があります。これが発生した場合は、**View** をクリックし、**Details** を選択します。デフォルトでは、Overview ウィンドウが最初に開きます。

3. 左側のナビゲーションペインから **Performance** を選択します。

   Performance ビューには、CPU およびメモリーの使用量、ディスクおよびネットワークの入出力など、ゲストのパフォーマンスの概要が表示されます。

図2.2 ゲストのパフォーマンスの詳細の表示



2.3.2. パフォーマンスのモニターリング

パフォーマンスモニターリングの設定は、**virt-manager** の設定ウィンドウで変更できます。

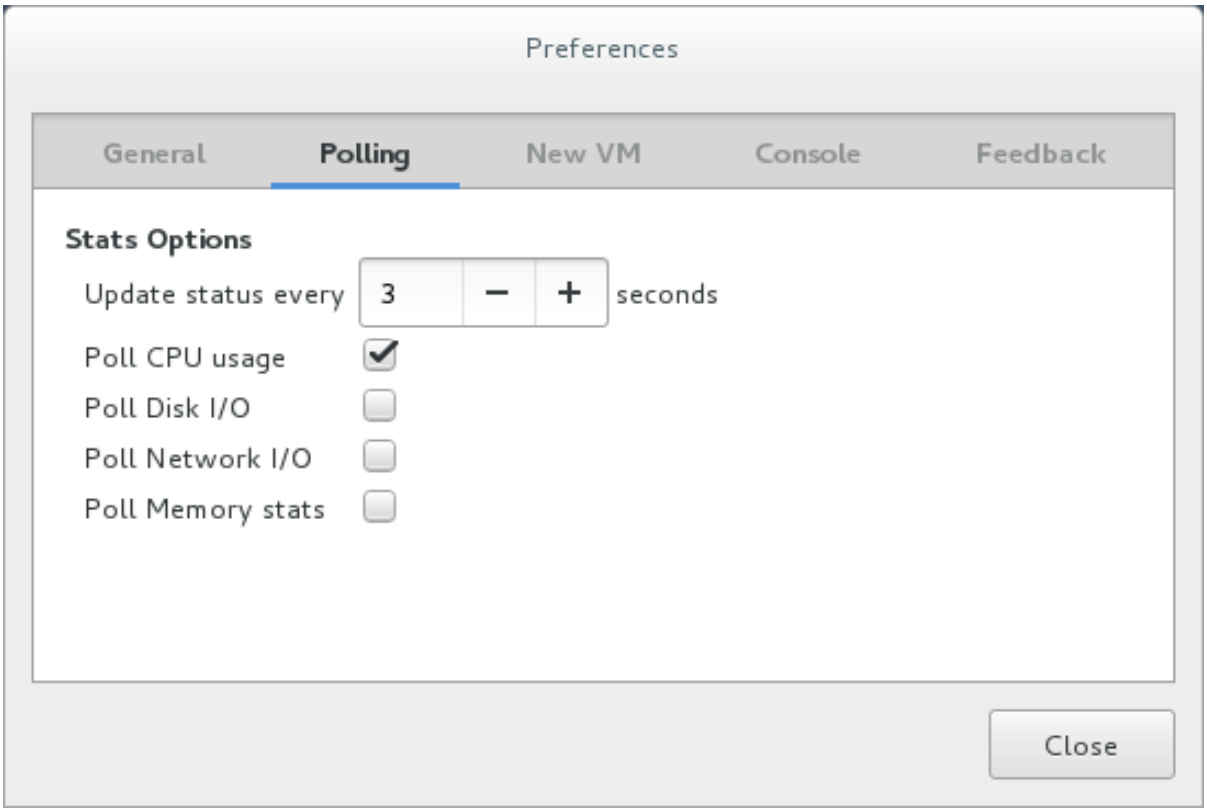
パフォーマンスモニターリングを設定するには、以下を実行します。

1. **Edit** メニューから、**Preferences** を選択します。

   Preferences ウィンドウが表示されます。

2. **Polling** タブから、秒単位の時間または統計ポーリングオプションを指定します。

図2.3 パフォーマンスモニターリングの設定

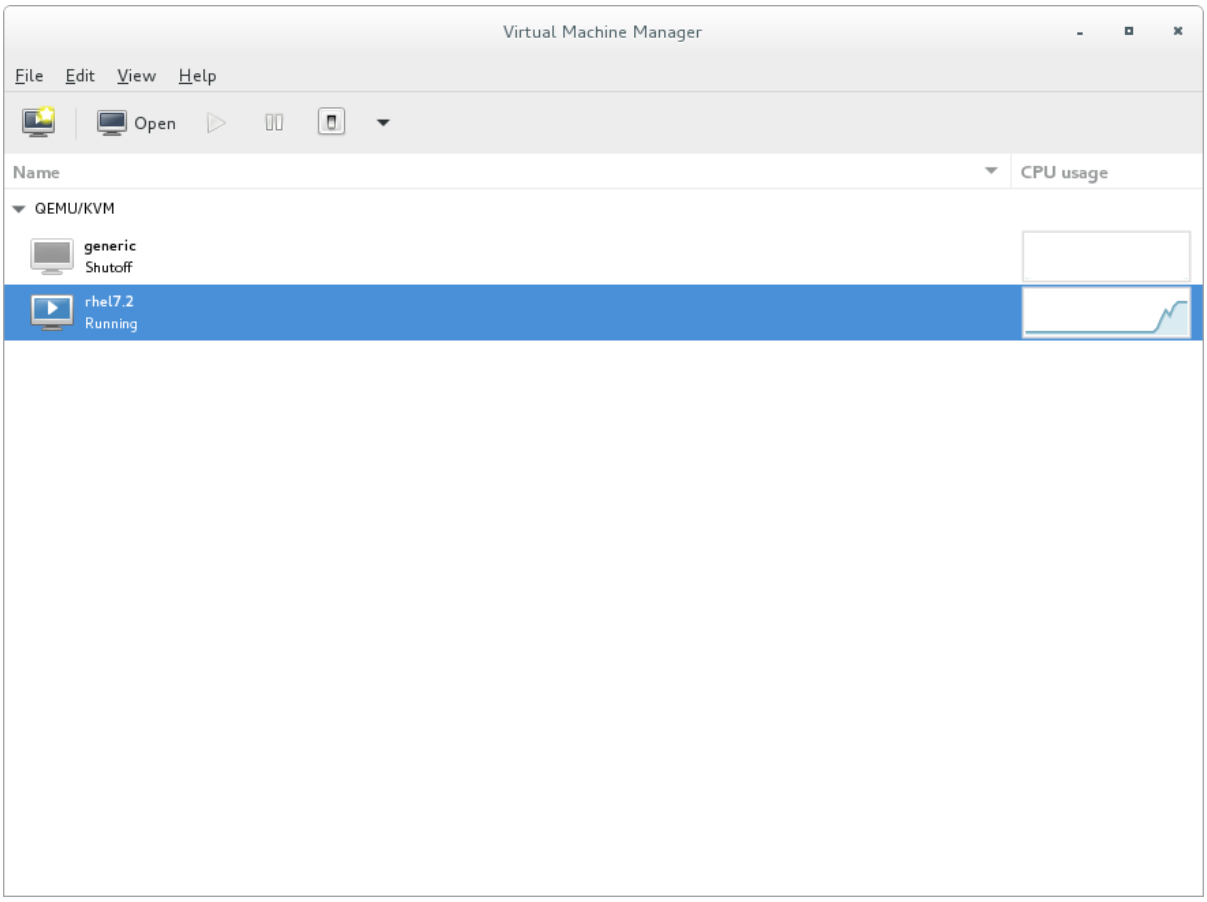


2.3.3. ゲストの CPU 使用率の表示

システム上のすべてのゲストの CPU 使用率を表示するには、以下を実行します。

1. **View** メニューから **Graph** を選択し、**Guest CPU Usage** チェックボックスをオンにします。
2. Virtual Machine Manager は、システム上のすべての仮想マシンの CPU 使用率グラフを表示します。

図2.4 ゲスト CPU 使用率グラフ

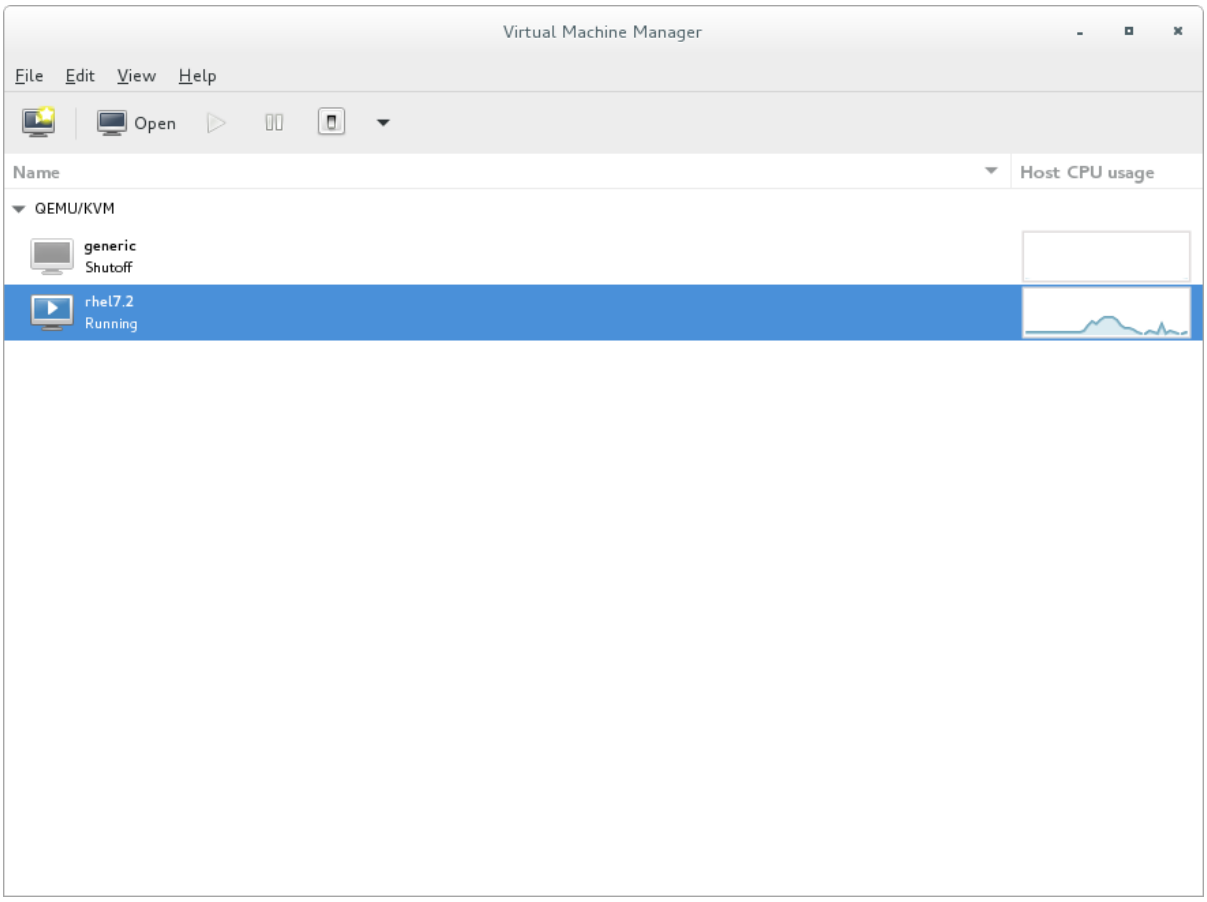


2.3.4. ホストの CPU 使用率の表示

システム上のすべてのホストの CPU 使用率を表示するには、以下を実行します。

1. **View** メニューから **Graph** を選択し、**Host CPU Usage** チェックボックスをオンにします。
2. Virtual Machine Manager は、システム上のホストの CPU 使用率グラフを表示します。

図2.5 ホストの CPU 使用率グラフ



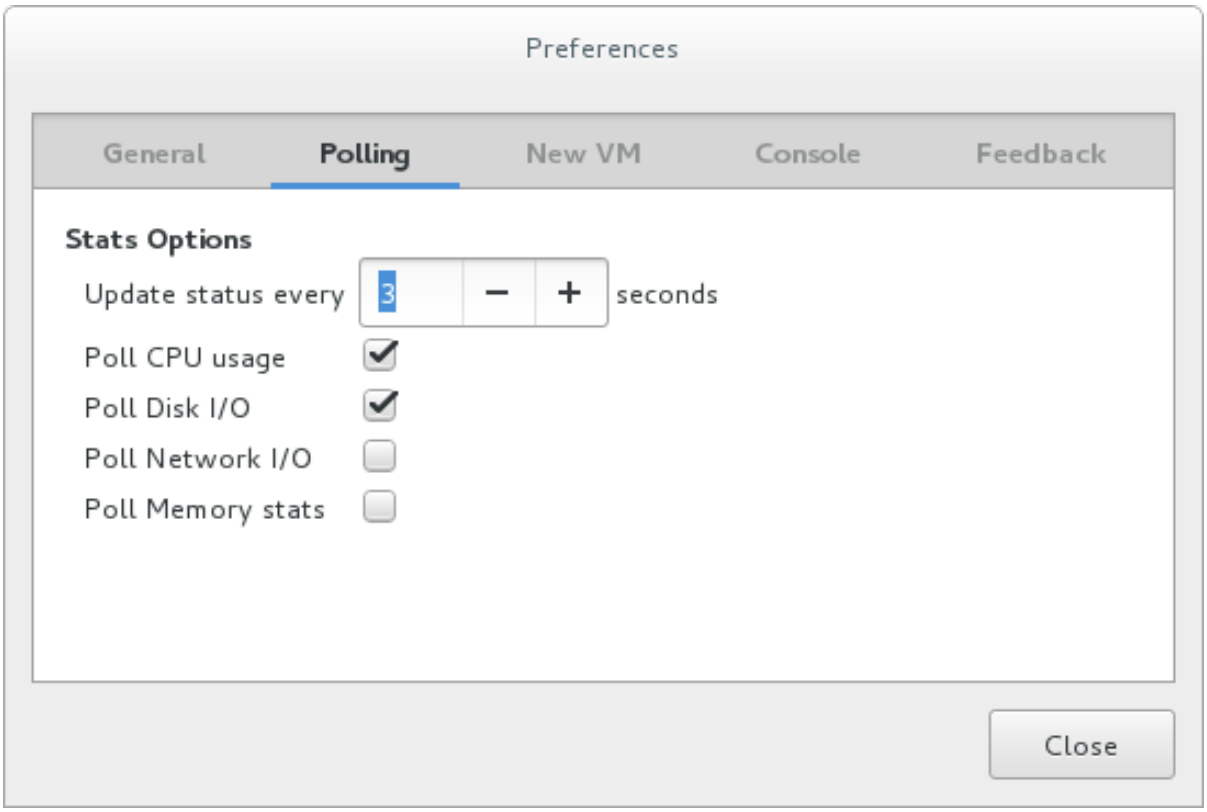
2.3.5. ディスク I/O の表示

システム上のすべての仮想マシンのディスク I/O を表示するには、以下を実行します。

1. ディスク I/O 統計情報の収集が有効になっていることを確認してください。これを行うには、**Edit** メニューから **Preferences** を選択し、**Polling** タブをクリックします。

2. **Disk I/O** チェックボックスをオンにします。

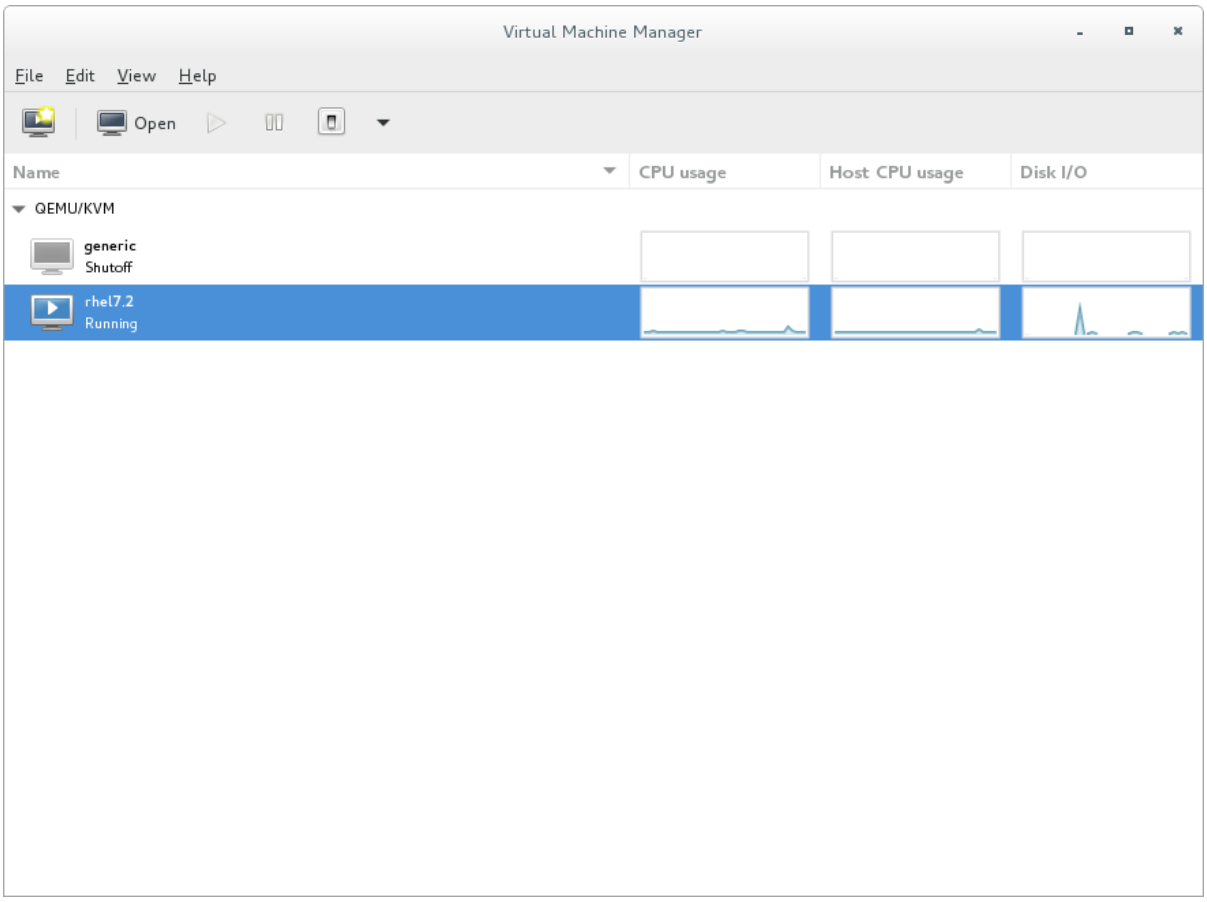
図2.6 ディスク I/O の有効化



3. ディスク I/O 表示を有効にするには、**View** メニューから **Graph** を選択し、**Disk I/O** チェックボックスをオンにします。選択します。

4. Virtual Machine Manager は、システム上のすべての仮想マシンのディスク I/O のグラフを表示します。

図2.7 ディスク I/O の表示



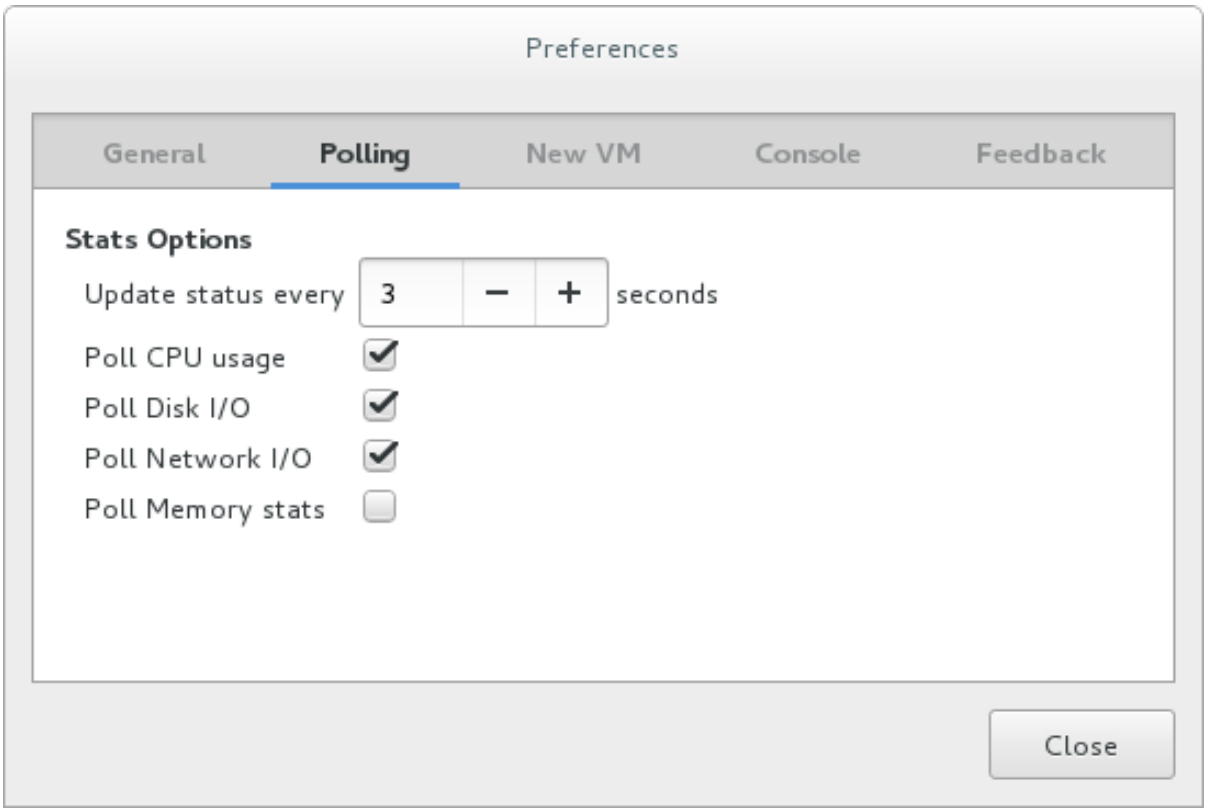
2.3.6. ネットワーク I/O の表示

システム上のすべての仮想マシンのネットワーク I/O を表示するには、以下を実行します。

1. ネットワーク I/O 統計情報の収集が有効になっていることを確認してください。これを行うには、**Edit** メニューから **Preferences** を選択し、**Polling** タブをクリックします。

2. **Network I/O** チェックボックスをオンにします。

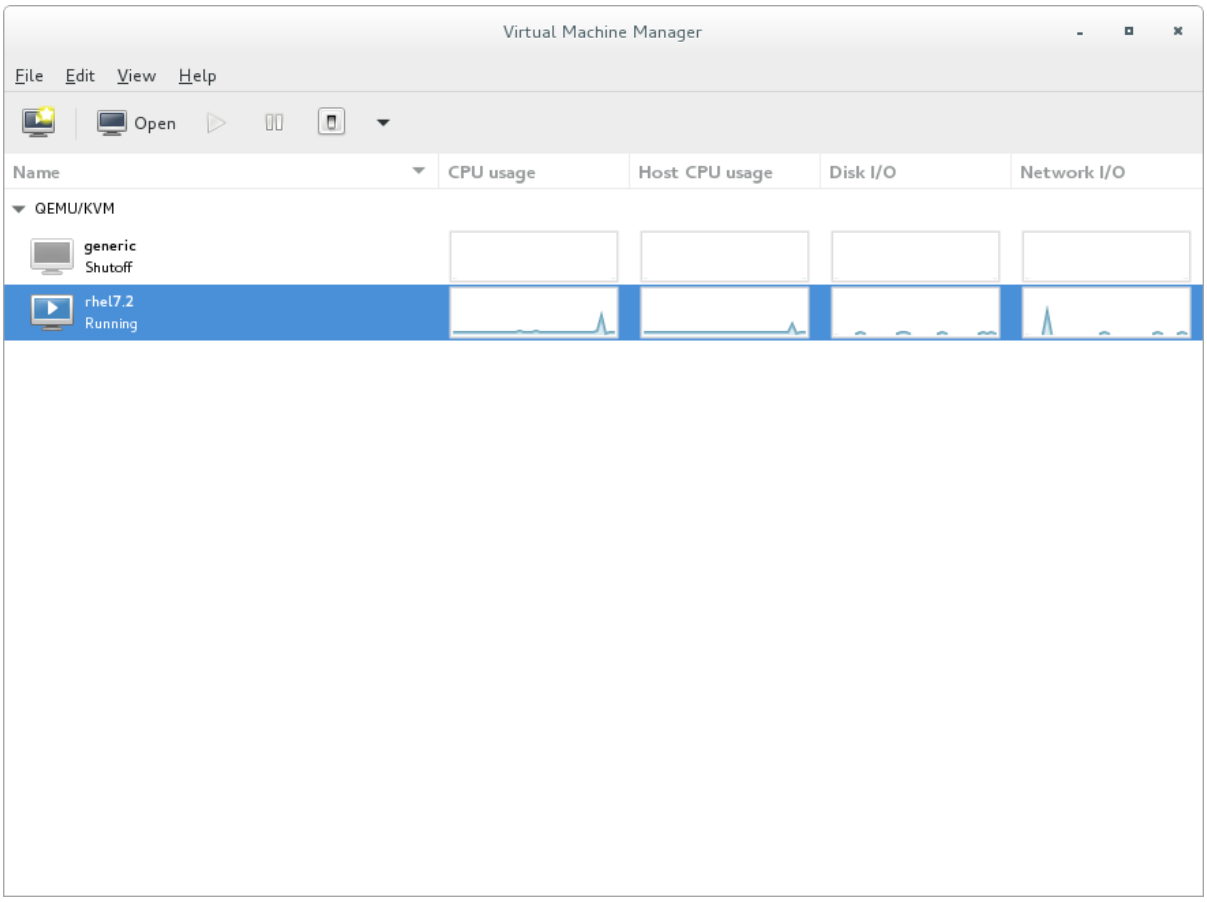
図2.8 ネットワーク I/O の有効化



3. ネットワーク I/O 統計情報を表示するには、**View** メニューから **Graph** を選択し、**Network I/O** チェックボックスをオンにします。

4. Virtual Machine Manager は、システム上のすべての仮想マシンのネットワーク I/O グラフを表示します。

図2.9 ネットワーク I/O の表示



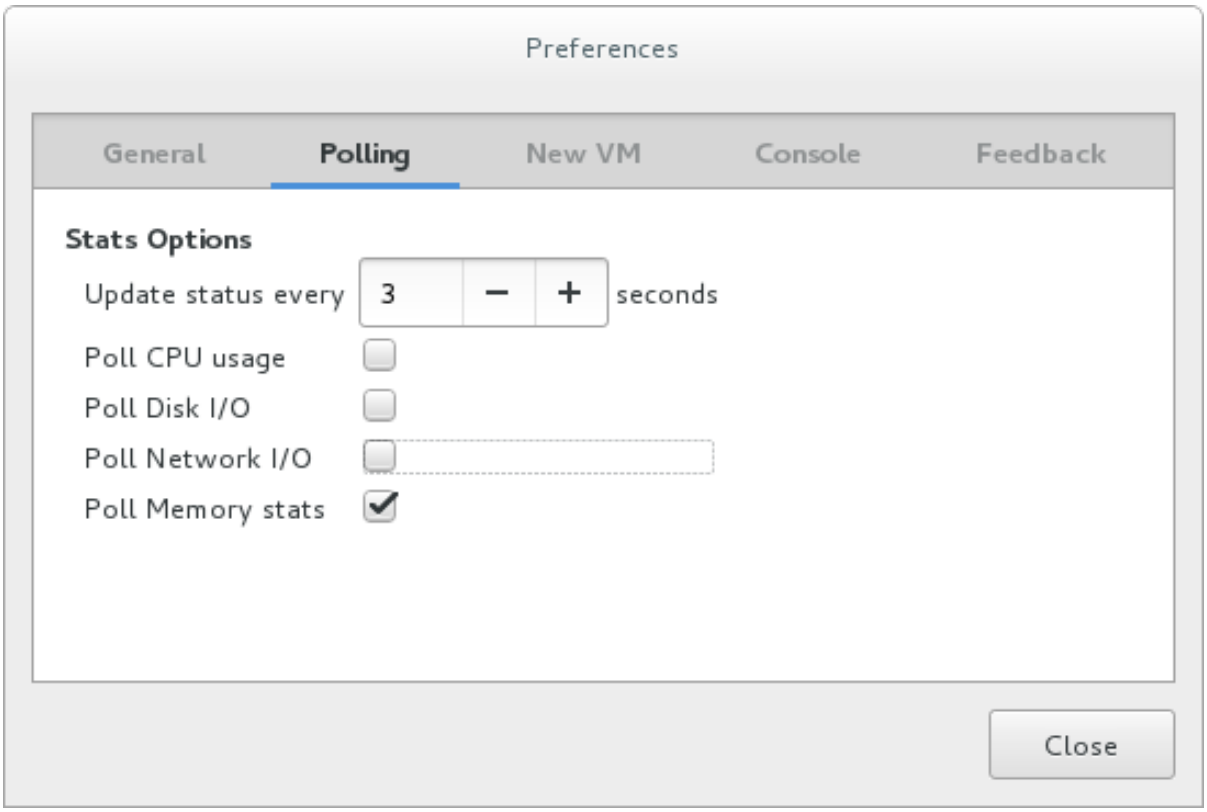
2.3.7. メモリー使用量の表示

システム上のすべての仮想マシンのメモリー使用量を表示するには、以下を実行します。

1. メモリー使用量の統計情報の収集が有効になっていることを確認してください。これを行うには、**Edit** メニューから **Preferences** を選択し、**Polling** タブをクリックします。

2. **Poll Memory stats** チェックボックスをオンにします。

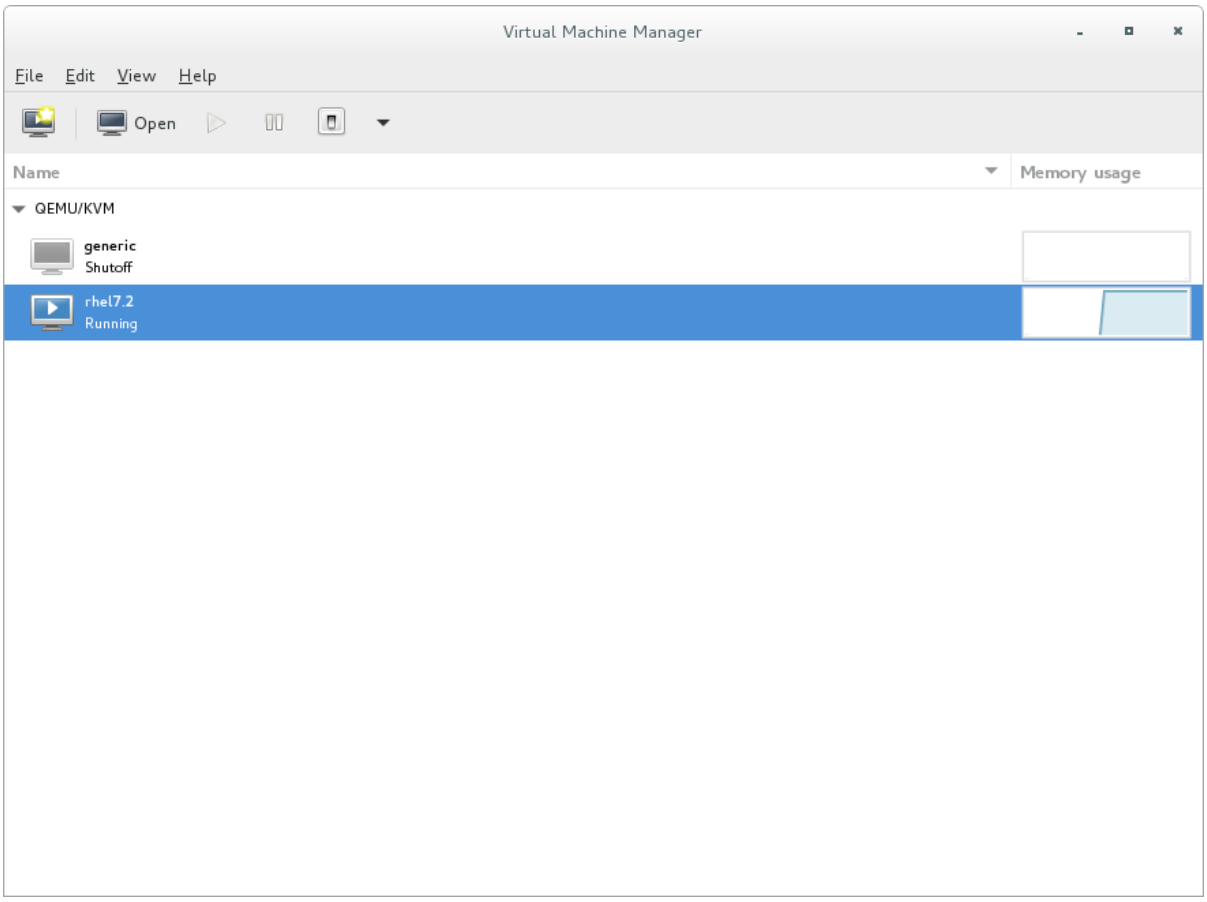
図2.10 メモリー使用量の有効化



3. メモリー使用量を表示するには、**View** メニューから **Graph** を選択し、**Memory Usage** チェックボックスをオンにします。

4. Virtual Machine Manager は、システム上のすべての仮想マシンで使用されているメモリーの割合 (メガバイト単位) を一覧表示します。

図2.11 メモリー使用量の表示



第3章 VIRT-MANAGER による仮想化パフォーマンスの最適化

この章では、ゲスト仮想マシンを管理するためのデスクトップツールである **virt-manager** で使用できる、パフォーマンスチューニングオプションについて説明します。

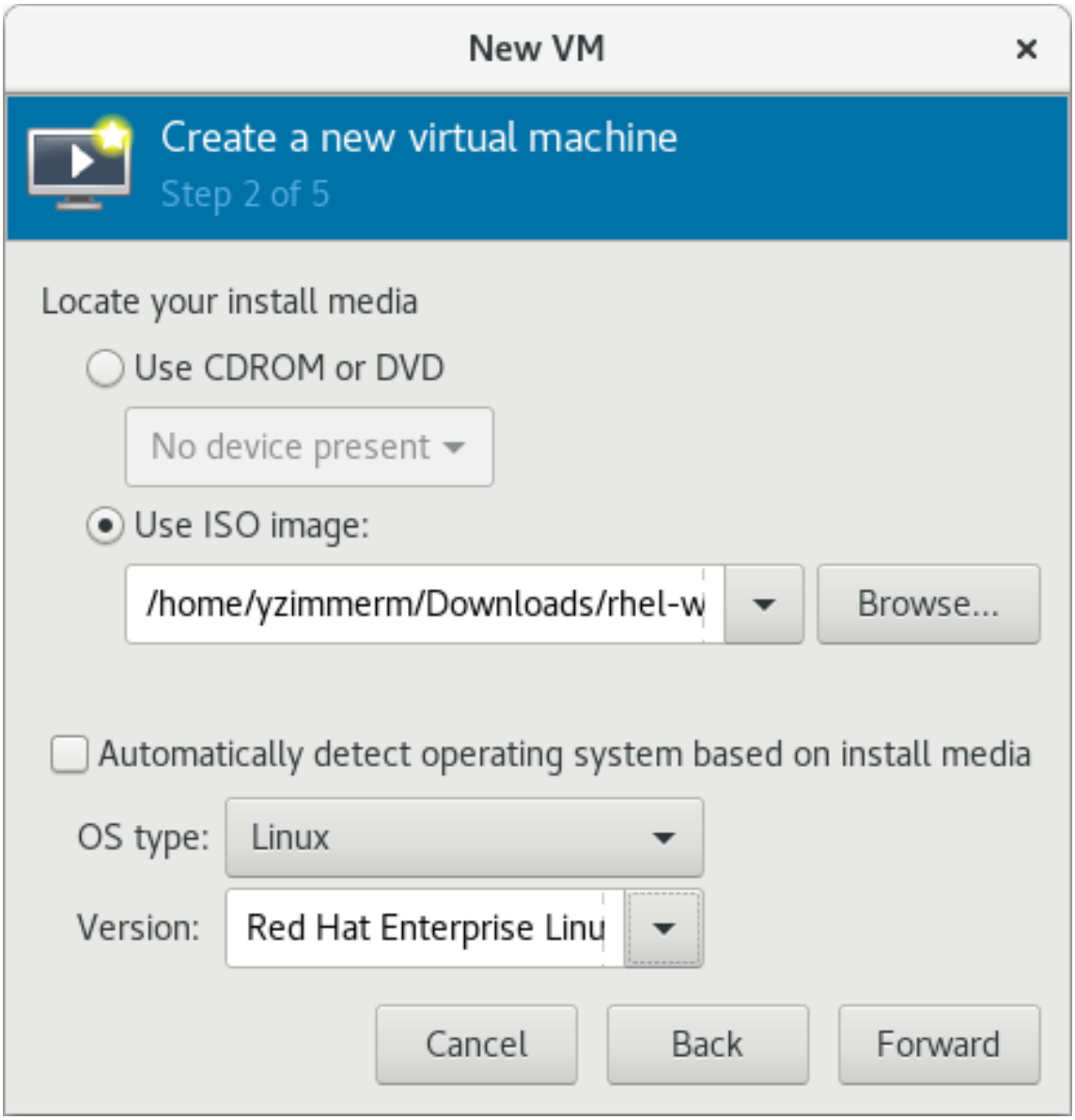
3.1. オペレーティングシステムの詳細とデバイス

3.1.1. ゲスト仮想マシンの詳細の指定

virt-manager ツールは、新しいゲスト仮想マシン用に選択されたオペレーティングシステムのタイプとバージョンに応じて異なるプロファイルを提供します。ゲストを作成するときは、できるだけ多くの詳細を提供する必要があります。これにより、特定のゲストタイプのための機能が有効になり、パフォーマンスを向上させることができます。

以下に示す、**virt-manager** ツールのスクリーンキャプチャーの例を参照してください。新しいゲスト仮想マシンを作成するときは、常に目的の **OS type** と **Version** を指定してください。

図3.1 OS のタイプとバージョンの提供

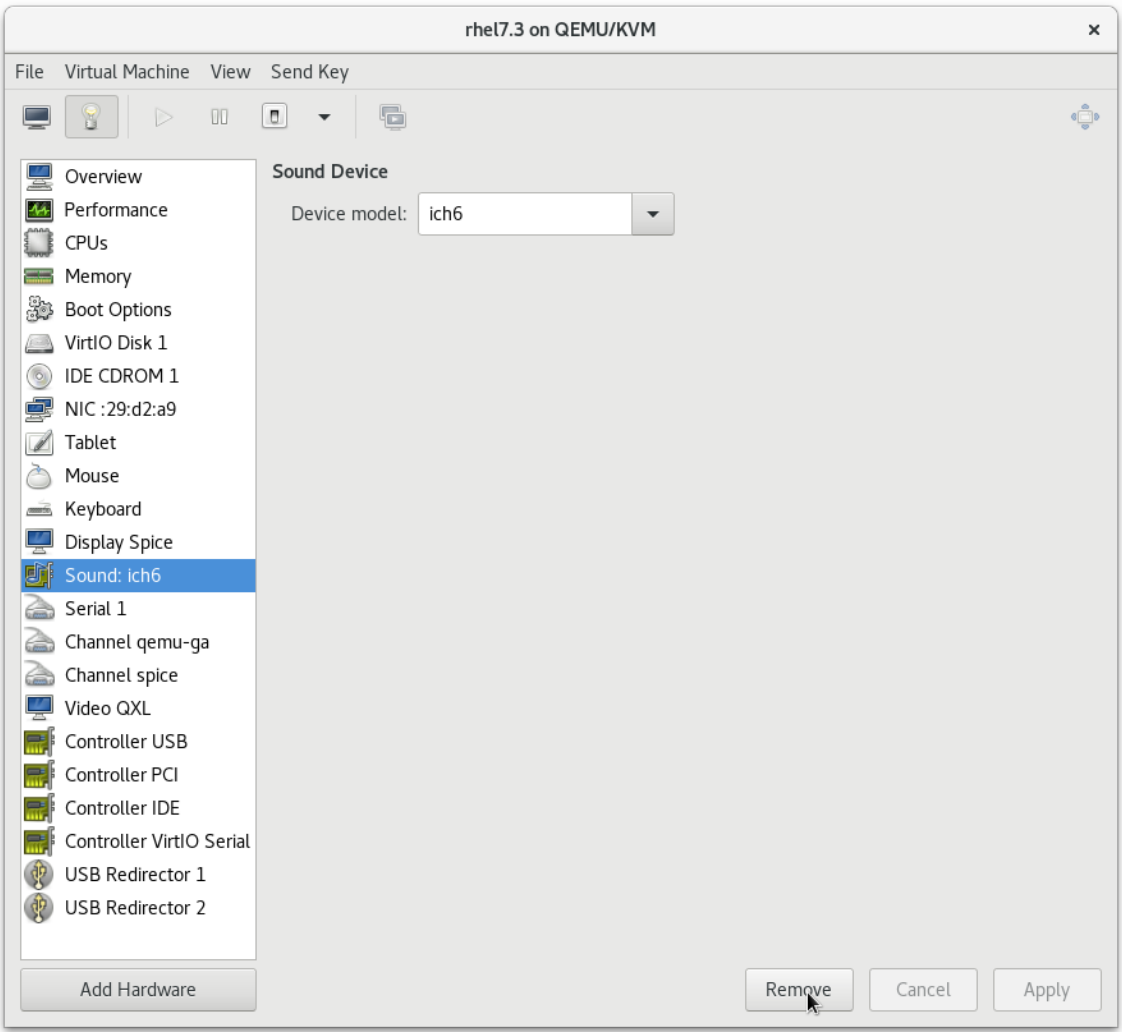


3.1.2. 未使用デバイスの削除

未使用または不要なデバイスを削除すると、パフォーマンスを向上させることができます。たとえば、Web サーバーとしてタスクを行うゲストは、オーディオ機能や接続されたタブレットを必要としない可能性があります。

以下に示す、**virt-manager** ツールのスクリーンキャプチャーの例を参照してください。**Remove** ボタンをクリックして、不要なデバイスを削除します。

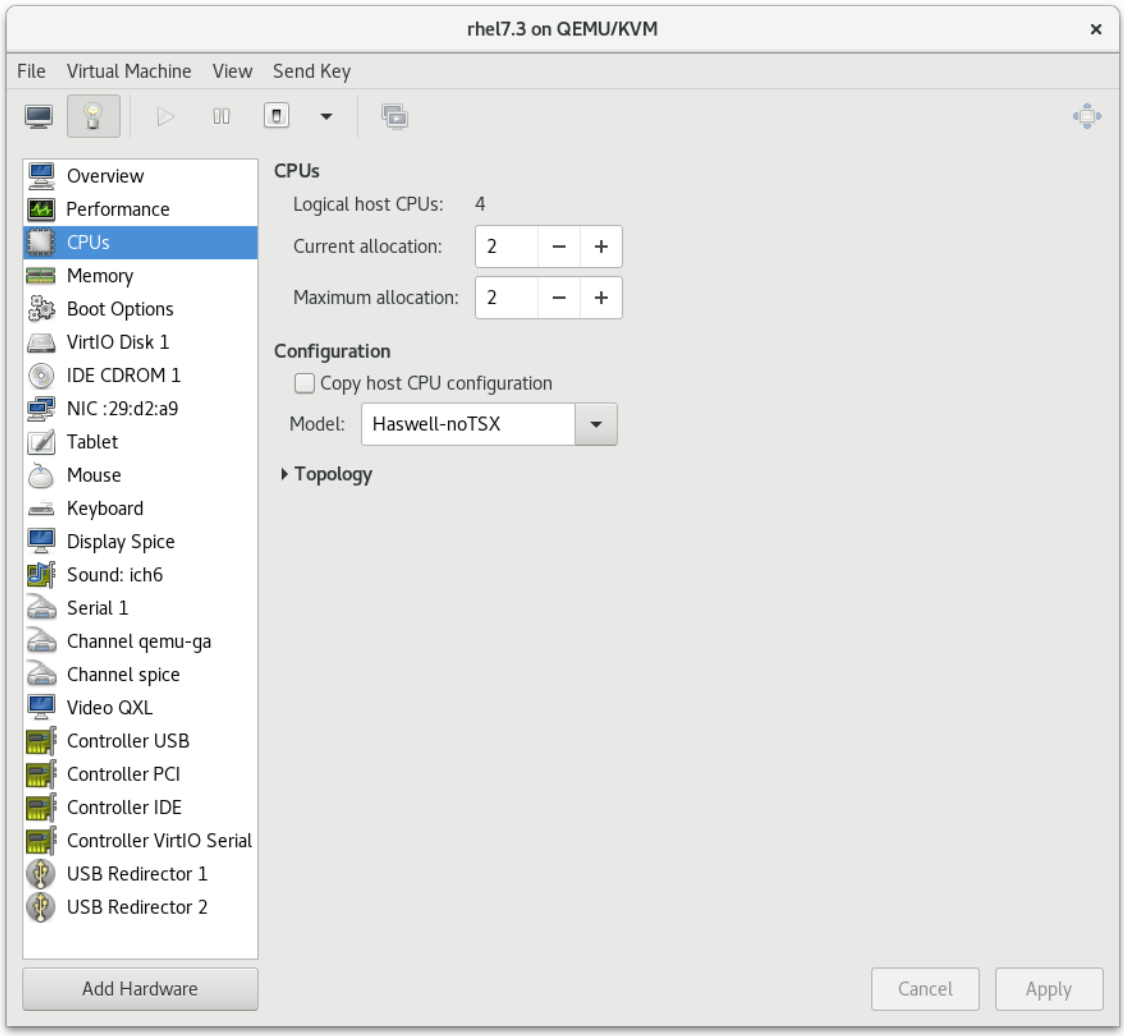
図3.2 未使用デバイスの削除



3.2. CPU パフォーマンスのオプション

ゲスト仮想マシンでは、いくつかの CPU 関連のオプションを使用できます。これらのオプションを正しく設定すると、パフォーマンスに大きな影響を与えることができる可能性があります。次のイメージは、ゲストが使用できる CPU オプションを示しています。このセクションの残りの部分では、これらのオプションの影響を示し、それについて説明します。

図3.3 CPU パフォーマンスのオプション

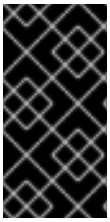


3.2.1. オプション: 使用可能な CPU

このオプションを使用して、ゲストが使用できる仮想 CPU (vCPU) の量を調整します。ホストで使用可能な量より多くを割り当てると (**オーバーコミット**と呼ばれます)、次の図に示すように警告が表示されます。

図3.4 CPU のオーバーコミット

システム上のすべてのゲストの vCPU の合計が、システム上のホスト CPU の数よりも多い場合、CPU はオーバーコミットされます。vCPU の総数がホスト CPU の数よりも多い場合は、1 つまたは複数のゲストで CPU がオーバーコミットになります。



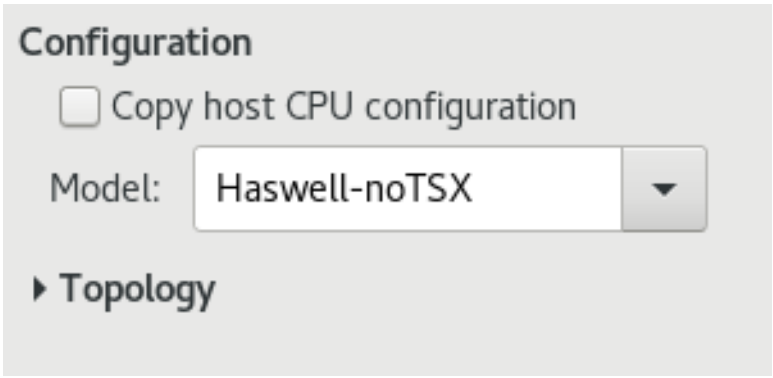
重要

メモリーのオーバーコミットと同様に、CPU のオーバーコミットは、たとえばゲストのワークロードが重くなったり予測できなくなったりするなど、パフォーマンスに悪影響を与える可能性があります。オーバーコミットの詳細については、『Virtualization Deployment and Administration Guide』 を参照してください。

3.2.2. オプション: CPU 設定

このオプションを使用して、目的の CPU モデルに基づいて CPU 設定タイプを選択します。**Copy host CPU configuration** チェックボックスをクリックして、物理ホストの CPU モデルと設定を検出して適用するか、リストを展開して使用可能なオプションを表示します。CPU 設定を選択すると、使用可能な CPU 機能/命令が表示され、**CPU Features** リストで個別に有効/無効にできます。

図3.5 CPU 設定オプション



注記

手動で設定するよりもホスト、ホスト CPU 設定をコピーすることをお勧めします。

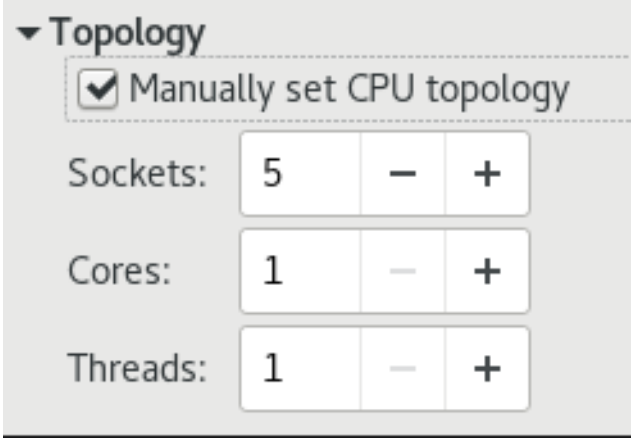
注記

または、ホストマシンで **virsh capabilities** コマンドを実行して、CPU タイプや NUMA 機能などをはじめとするシステムの仮想化機能を表示します。

3.2.3. オプション: CPU トポロジー

このオプションを使用して、特定の CPU トポロジー (ソケット、コア、スレッド) をゲスト仮想マシンの仮想 CPU に適用します。

図3.6 CPU トポロジーのオプション



注記

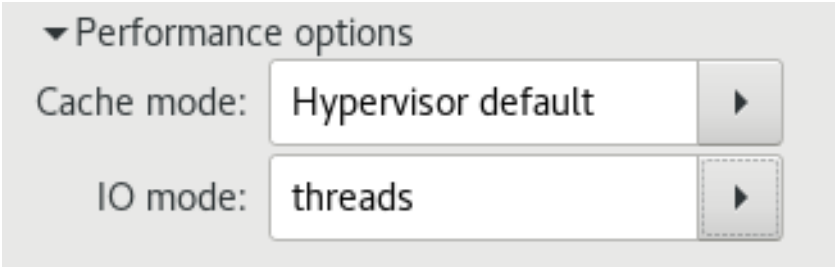
ご使用の環境で他の要件が決まる場合がありますが、意図した数のソケットを選択し、コアとスレッドは1つのみ使用すると、通常は最適なパフォーマンス結果が得られます。

3.3. 仮想ディスクパフォーマンスのオプション

インストール中は、パフォーマンスに影響を与える可能性がある、いくつかの仮想ディスク関連オプションをゲスト仮想マシンで使用できます。次のイメージは、ゲストが使用できる仮想ディスクオプションを示しています。

キャッシュモード、IO モード、および IO チューニングは、**virt-manager** の **Virtual Disk** セクションで選択できます。次の図で示されるとおり、**Performance options** の下のフィールドでこれらのパラメーターを設定します。

図3.7 仮想ディスクパフォーマンスのオプション



重要

virt-manager で仮想ディスクのパフォーマンスオプションを設定する場合、設定を有効にするには仮想マシンを再起動する必要があります。

これらの設定の説明と、ゲスト XML 設定でこれらの設定を編集する手順については、「キャッシュ」および 「I/O モード」 を参照してください。

第4章 TUNED と TUNED-ADM

この章では、**tuned** デーモンを使用して、仮想化環境でシステム設定を調整する方法について説明します。

tuned は、CPU 集中型タスクや、ストレージ/ネットワークスループットの応答などの特定のワークロードの特性に対して Red Hat Enterprise Linux を調整するプロファイル配信メカニズムです。これにより、特定のユースケースで、パフォーマンスを強化し、電力消費を減らすように事前設定されたチューニングプロファイルを多数利用できます。仮想化環境に適したパフォーマンスソリューションを作成するには、これらのプロファイルを編集するか、または新規プロファイルを作成します。

tuned の一部として提供される仮想化関連のプロファイルは次のとおりです。

virtual-guest

virtual-guest は ***throughput-performance*** プロファイルをベースにしており、仮想メモリーのスワップも減少します。

virtual-guest プロファイルは、Red Hat Enterprise Linux 7 ゲスト仮想マシンを作成するときに自動的に選択されます。これは、仮想マシンに推奨されるプロファイルです。

このプロファイルは Red Hat Enterprise Linux 6.3 以降で使用できますが、仮想マシンをインストールするときに手動で選択する必要があります。

virtual-host

throughput-performance プロファイルに基づき、***virtual-host*** はダーティページのより積極的なライトバックも有効にします。このプロファイルは、KVM ホストと Red Hat Virtualization (RHV) ホストの両方を含む仮想化ホストに推奨されるプロファイルです。

デフォルトでは、Red Hat Enterprise Linux 7 のインストールでは、tuned パッケージがインストールされ、**tuned** サービスが有効になります。

利用可能な全プロファイルを一覧表示して、現在アクティブなプロファイルを特定するには、以下を実行します。

```
# tuned-adm list
Available profiles:
- balanced
- desktop
- latency-performance
- network-latency
- network-throughput
- powersave
- sap
- throughput-performance
- virtual-guest
- virtual-host
Current active profile: throughput-performance
```

カスタム **tuned** プロファイルを作成して、一連のチューニングパラメーターをカプセル化することもできます。カスタム **tuned** プロファイルを作成する手順については、**tuned.conf** の man ページを参照してください。

現在アクティブなプロファイルのみを表示するには、以下を実行します。

```
tuned-adm active
```

別のプロファイルに切り替える場合は、以下を実行します。

```
tuned-adm profile profile_name
```

たとえば、**virtual-host** プロファイルに切り替えるには、以下を実行します。

```
tuned-adm profile virtual-host
```

重要

Red Hat Enterprise Linux 7.1 以降で tuned プロファイルを設定した後、再起動後に設定済みプロファイルを適用するために **tuned** サービスが有効になっていることを確認してください。

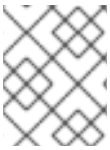
```
# systemctl enable tuned
```

場合によっては、**tuned** を無効にして、手動で設定されたパラメーターを使用することが望ましいこともあります。現在のセッションのすべてのチューニングを無効にするには、以下を実行します。

```
# tuned-adm off
```

tuned を永続的に無効にし、実行したすべての変更を元に戻すには、以下を実行します。

```
# tuned-adm off; systemctl disable tuned
```



注記

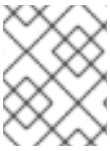
tuned の詳細については、『Red Hat Enterprise Linux 7 Performance Tuning Guide』を参照してください。

第5章 ネットワーク

この章では、仮想化環境のネットワーク最適化に関するトピックについて説明します。

5.1. ネットワークチューニングのヒント

- 複数のネットワークを使用して、単一ネットワークでの輻輳を回避します。たとえば、管理、バックアップ、またはライブマイグレーション専用のネットワークを用意します。
- Red Hat は、同じネットワークセグメントで複数のインターフェイスを使用しないことをお勧めします。ただし、それが避けられない場合は、***arp_filter*** を使用して、ホストとゲストの両方で発生する可能性があり、マシンが複数のネットワークインターフェイスからの ARP 要求に応答することによって引き起こされる望ましくない状態である ARP Flux を防ぐことができます。この設定を永続化するには、**echo 1 > /proc/sys/net/ipv4/conf/all/arp_filter** または **/etc/sysctl.conf** を編集します。



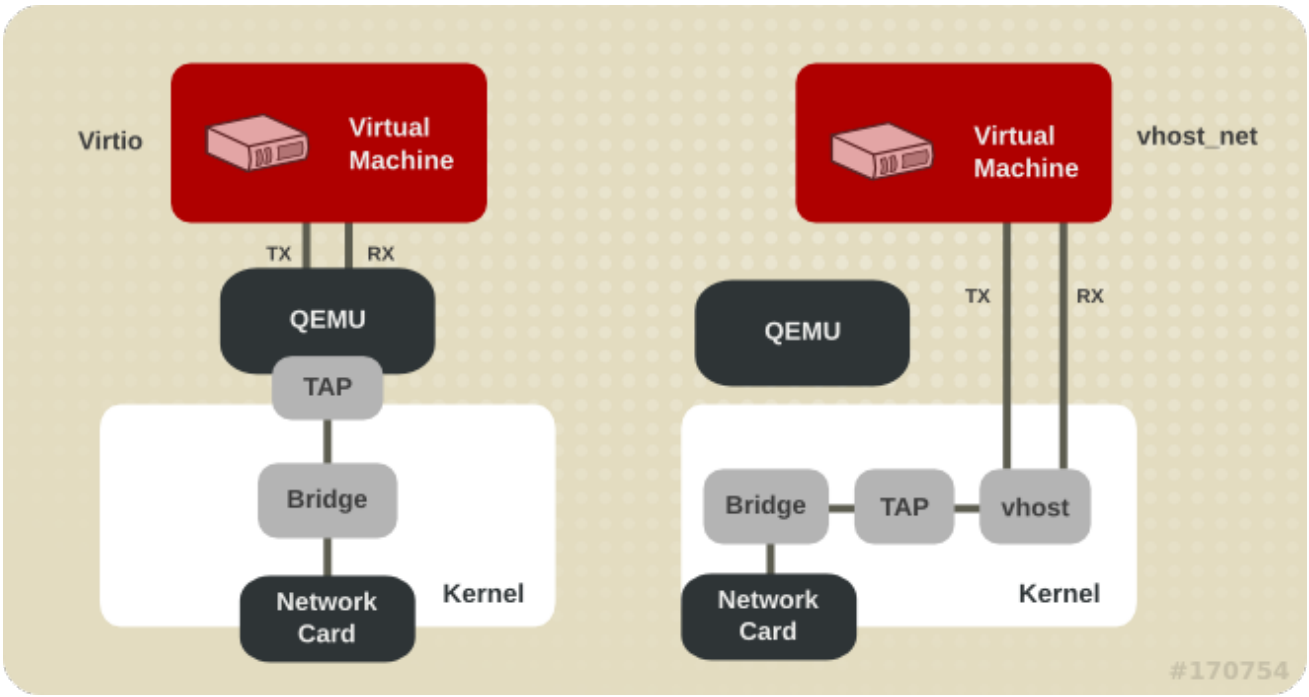
注記

ARP Flux の詳細については、http://linux-ip.net/html/ether-arp.html#ether-arp-flux を参照してください。

5.2. VIRTIO と VHOST_NET

次の図は、Virtio および vhost_net アーキテクチャーへのカーネルの関与を示しています。

図5.1 Virtio および vhost_net アーキテクチャー

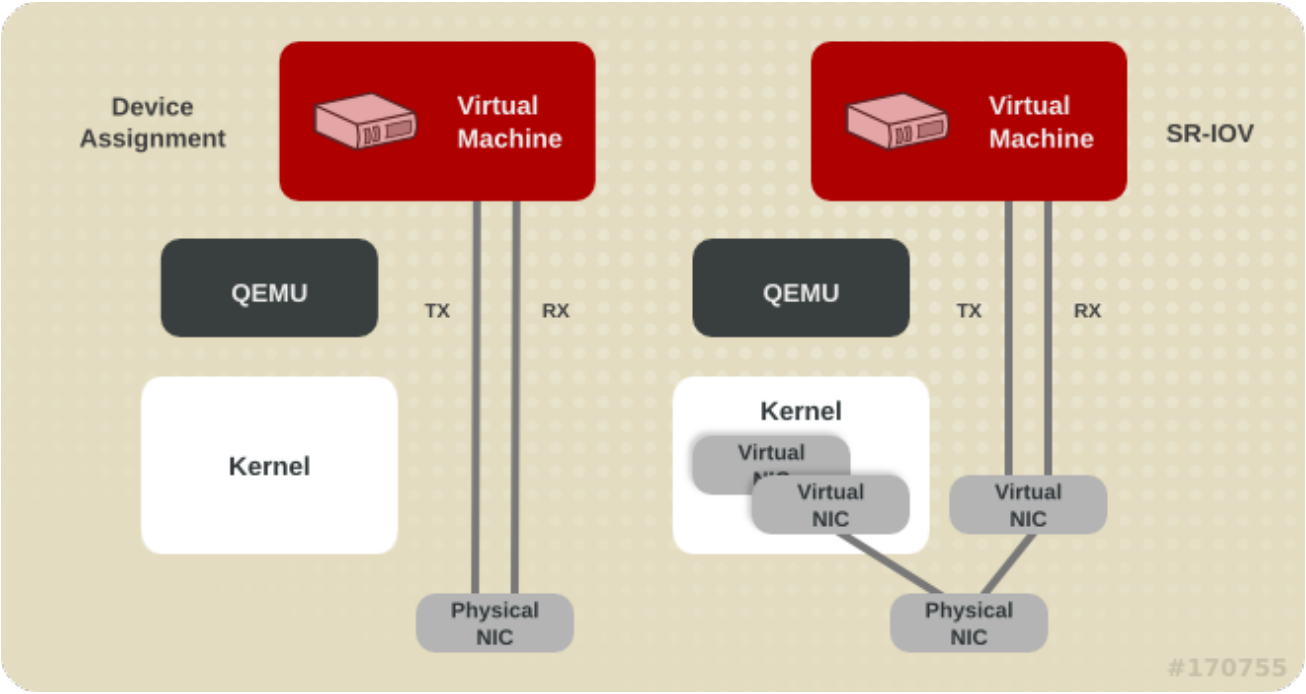


vhost_net は、Virtio ドライバーの一部をユーザースペースからカーネルに移動します。これにより、コピー操作が減少し、レイテンシーと CPU 使用率が低下します。

5.3. デバイスの割り当てと SR-IOV

次の図は、デバイス割り当ておよび SR-IOV アーキテクチャーへのカーネルの関与を示しています。

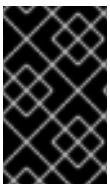
図5.2 デバイスの割り当てと SR-IOV



デバイス割り当ては、デバイス全体をゲストに提示します。SR-IOV は、NIC やシステムボードなどのドライバーおよびハードウェアのサポートを必要とします。また、複数の仮想デバイスを作成してさまざまなゲストに渡すことができます。ゲストにはベンダー固有のドライバーが必要ですが、SR-IOV のレイテンシーはネットワークオプションの中で最も低くなります。

5.4. ネットワークチューニング技術

このセクションでは、仮想化環境でネットワークパフォーマンスをチューニングするための手法について説明します。



重要

以下の機能は、Red Hat Enterprise Linux 7 ハイパーバイザーと仮想マシンでサポートされていますが、Red Hat Enterprise Linux 6.6 以降を実行している仮想マシンでもサポートされています。

5.4.1. ブリッジのゼロコピー送信

ゼロコピー送信モードは、大きなパケットサイズで効果的です。通常は、スループットに影響を与えることなく、ゲストネットワークと外部ネットワーク間で大きなパケットを送信する際のホスト CPU オーバーヘッドが最大 15% 削減されます。

これは、ゲストとゲスト間、ゲストとホスト間、または小さなパケットのワークロードのパフォーマンスには影響を与えません。

ブリッジのゼロコピー送信は、Red Hat Enterprise Linux 7 仮想マシンで完全にサポートされていますが、デフォルトでは無効になっています。ゼロコピー送信モードを有効にするには、vhost_net モジュールの ***experimental_zcopytx*** カーネルモジュールパラメーターを 1 に設定します。詳細な手順については、Virtualization Deployment and Administration Guide を参照してください。

注記

通常、追加のデータコピーは、サービス拒否攻撃や情報漏えい攻撃に対する脅威軽減手法として、送信中に作成されます。ゼロコピー送信を有効にすると、この脅威軽減手法が無効になります。

パフォーマンスの低下が見られる場合、またはホストの CPU 使用率が問題にならない場合は、***experimental_zcopytx*** を 0 に設定することで、ゼロコピー送信モードを無効にできます。

5.4.2. マルチキュー virtio-net

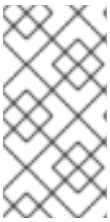
マルチキュー virtio-net は、一度に複数の virtqueue ペアを介してパケットを転送できるようにすることで、vCPU の数が増えるにつれてネットワークパフォーマンスをスケーリングするアプローチを提供します。

今日のハイエンドサーバーにはより多くのプロセッサーが搭載されており、多くの場合、それらで実行されているゲストでは vCPU の数が増加します。シングルキュー virtio-net では、vCPU の数が増えてもネットワークパフォーマンスはスケーリングされないため、ゲストのプロトコルスタックスケールが制限されます。virtio-net には TX キューと RX キューが 1 つしかないため、ゲストはパケットを並行して送信または取得できません。

マルチキューのサポートにより、並列パケット処理が可能になるため、これらのボトルネックが解消されます。

マルチキュー virtio-net は、次の場合に最大のパフォーマンス上の利点を提供します。

- トラフィックパケットが比較的大きい。
- ゲストが多くの接続で同時にアクティブになり、トラフィックがゲストとゲスト、ゲストとホスト、ゲストと外部システムの間に流れる。
- キューの数が vCPU の数と同じ。これは、マルチキューのサポートにより、特定のキューを特定の vCPU に対してプライベートにするために、RX 割り込みアフィニティーと TX キューの選択が最適化されるためです。



注記

現在、マルチキュー virtio-net 接続を設定すると、送信トラフィックのパフォーマンスに悪影響を与える可能性があります。具体的には、TCP (Transmission Control Protocol) ストリームで 1,500 バイト未満のパケットを送信すると発生する可能性があります。詳細は Red Hat ナレッジベース を参照してください。

5.4.2.1. マルチキュー virtio-net の設定

マルチキュー virtio-net を使用するには、ゲスト XML 設定に以下を追加してゲストでのサポートを有効にします (カーネルはマルチキュータップデバイスで最大 256 キューをサポートするため、*N* の値は 1〜256 です)。

```
<interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
    <driver name='vhost' queues='N'/>
</interface>
```

ゲストで *N* の virtio-net キューがある仮想マシンを実行する場合は、次のコマンドでマルチキューサポートを有効にします (*M* の値は 1 から *N* です)。

```
# ethtool -L eth0 combined M
```

5.5. ネットワークパケットのバッチ処理

長い転送パスの設定では、パケットをバッチ処理してからカーネルに送信することで、キャッシュが有効に活用される場合があります。

バッチ処理できるパケットの最大数を設定する場合、*N* がバッチ処理するパケットの最大数です。

```
# ethtool -C $tap rx-frames N
```

type='bridge' または type='network' インターフェイスの **tun**/**tap** rx バッチ処理を提供するには、次のようなスニペットをドメイン XML ファイルに追加します。

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
      <coalesce>
        <rx>
          <frames max='7'/>
        </rx>
      </coalesce>
  </interface>
</devices>
```

第6章 I/O スケジューリング

入出力 (I/O) スケジューラーを使用して、Red Hat Enterprise Linux 7 が仮想化 ホスト である場合と仮想化 ゲスト である場合の両方で、ディスクのパフォーマンスを向上させることができます。

6.1. 仮想化ホストとして RED HAT ENTERPRISE LINUX を使用した I/O スケジューリング

仮想化ゲストのホストとして Red Hat Enterprise Linux 7 を使用する場合、通常はデフォルトの **deadline** スケジューラーが理想的です。このスケジューラーは、ほぼすべてのワークロードで適切に機能します。

ただし、ゲストワークロードでの I/O レイテンシーを最小化するよりも、I/O スループットを最大化することが重要な場合は、代わりに **cfq** スケジューラーを使用することが推奨されます。

6.2. 仮想化ゲストとして RED HAT ENTERPRISE LINUX を使用した I/O スケジューリング

ゲストが実行されているハイパーバイザーに関係なく、Red Hat Enterprise Linux ゲスト仮想マシンで I/O スケジューリングを使用できます。以下は、考慮すべき利点と問題のリストです。

- Red Hat Enterprise Linux のゲストは、多くの場合、**noop** スケジューラーを使用することで大きなメリットが得られます。スケジューラーは、ハイパーバイザーに I/O を送信する前に、ゲストオペレーティングシステムからの小さな要求を大きな要求にマージします。これにより、ハイパーバイザーは I/O 要求をより効率的に処理できるようになり、ゲストの I/O パフォーマンスも大幅に向上します。

- ワークロード I/O とストレージデバイスがどのようにアタッチされているかによっては、**deadline** などのスケジューラーが **noop** よりも有益な場合があります。Red Hat は、パフォーマンステストを実行して、どのスケジューラーがパフォーマンスに最も影響を与えるかを確認することをお勧めします。

- iSCSI、SR-IOV、または物理デバイスパススルーによってアクセスされるストレージを使用するゲストは、**noop** スケジューラーを使用しないでください。これらのメソッドでは、ホストが基盤となる物理デバイスへの I/O 要求を最適化することはできません。

注記

仮想化環境では、ホスト層とゲスト層の両方で I/O をスケジュールすることが有益でない場合があります。複数のゲストがホストオペレーティングシステムによって管理されるファイルシステムまたはブロックデバイスのストレージを使用する場合、ホストはすべてのゲストからの要求を認識しているため、I/O をより効率的にスケジュールできる可能性があります。さらに、ホストは、ゲストの仮想ストレージに直線的にマッピングされないストレージの物理レイアウトを認識します。

合成ベンチマークは通常、仮想環境で共有リソースを使用するシステムのパフォーマンスを正確に比較しないため、すべてのスケジューラーのチューニングは通常の動作条件下でテストする必要があります。

6.2.1. Red Hat Enterprise Linux 7 の I/O スケジューラーの設定

Red Hat Enterprise Linux 7 システムで使用されるデフォルトのスケジューラーは **deadline** です。ただし、Red Hat Enterprise Linux 7 ゲストマシンでは、以下を実行して、スケジューラーを **noop** に変更すると便利な場合があります。

1. **/etc/default/grub** ファイルで、**GRUB_CMDLINE_LINUX** 行の **elevator=deadline** 文字列を **elevator=noop** に変更します。**elevator=** 文字列がない場合は、行の最後に **elevator=noop** を追加します。

   以下は、変更が成功した後の **/etc/default/grub** ファイルを示しています。

   ```
   # cat /etc/default/grub
   [...]
   GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=vg00/lvroot rhgb quiet
   elevator=noop"
   [...]
   ```

2. **/boot/grub2/grub.cfg** ファイルを再構築します。

   - BIOS ベースのマシンの場合:

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

   - UEFI ベースのマシンの場合:

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

第7章 ブロック I/O

この章では、仮想化環境での I/O 設定の最適化について説明します。

7.1. ブロック I/O チューニング

virsh blkiotune コマンドを使用すると、管理者はゲスト XML 設定の **<blkio>** 要素で、ゲスト仮想マシンのブロック I/O パラメーターを手動で設定または表示できます。

仮想マシンの現在の **<blkio>** パラメーターを表示するには、以下を実行します。

```
# virsh blkiotune virtual_machine
```

仮想マシンの **<blkio>** パラメーターを設定するには、**virsh blkiotune** コマンドを使用して、お使いの環境に応じてオプションの値を置き換えます。

```
# virsh blkiotune virtual_machine [--weight number] [--device-weights string] [--config] [--live] [--current]
```

パラメーター にはいかが含まれます。

weight

100 から 1000 までの範囲内の I/O の重み。

デバイスの I/O ウェイトを上げると、I/O 帯域幅の優先度が高まるため、より多くのホストリソースが提供されます。同様に、デバイスのウェイトを下げると、ホストのリソースが少なくなります。

device-weights

/path/to/device,weight,/path/to/device,weight の形式で、1 つ以上のデバイス/重みのペアをリストする単一の文字列。デバイスごとのリストからそのデバイスを削除するには、各重みが 100 - 1000 の範囲内であるか、値が 0 である必要があります。文字列にリストされているデバイスのみが変更されます。他のデバイスのデバイスごとの重みは変更されません。

config

次回の起動時に変更を有効にするには、**--config** オプションを追加します。

live

--live オプションを追加して、実行中の仮想マシンに変更を適用します。

注記

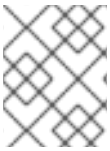
--live オプションでは、ハイパーバイザーがこのアクションをサポートする必要があります。すべてのハイパーバイザーが、最大メモリー制限のライブ変更を許可するわけではありません。

current

--current オプションを追加して、現在の仮想マシンに変更を適用します。

たとえば、以下では、*liftbrul* 仮想マシンの **/dev/sda** デバイスの重みを 500 に変更します。

```
# virsh blkiotune liftbrul --device-weights /dev/sda, 500
```



注記

virsh blkiotune コマンドの使用の詳細については、**virshhelpblkiotune** コマンドを使用してください。

7.2. キャッシュ

キャッシュオプションは、ゲストのインストール中に **virt-manager** を使用して設定するか、既存のゲスト仮想マシンでゲスト XML 設定を編集して設定できます。

表7.1 キャッシングオプション

キャッシングオプション	説明
Cache=none	ゲストからの I/O はホストにキャッシュされませんが、ライトバックディスクキャッシュに保持される場合があります。このオプションは、I/O 要件が大きいゲストに使用します。このオプションは一般的に最善の選択であり、移行をサポートする唯一のオプションです。
Cache=writethrough	ゲストからの I/O はホストにキャッシュされますが、物理メディアに書き込まれます。このモードは遅く、スケーリングの問題が発生しやすくなります。I/O 要件が小さい少数のゲストに最適です。移行が不要な、ライトバックキャッシュをサポートしていないゲスト (Red Hat Enterprise Linux 5.5 以前など) に推奨されます。
Cache=writeback	ゲストからの I/O はホストにキャッシュされます。
Cache=directsync	***writethrough*** に似ていますが、ゲストからの I/O はホストページのキャッシュを回避します。
Cache=unsafe	ホストはすべてのディスク I/O をキャッシュする可能性があり、ゲストからの同期要求は無視されます。
Cache=default	キャッシュモードが指定されていない場合は、システムのデフォルト設定が選択されます。

virt-manager では、キャッシュモードは **Virtual Disk** で指定できます。**virt-manager** を使用してキャッシュモードを変更する方法については、「仮想ディスクパフォーマンスのオプション」 を参照してください。

ゲスト XML でキャッシュモードを設定するには、**driver** タグ内の **cache** 設定を編集してキャッシュオプションを指定します。たとえば、キャッシュを ***writeback*** に設定するには、以下を実行します。

```
<disk type='file' device='disk'>
        <driver name='qemu' type='raw' cache='writeback'/>
```

7.3. I/O モード

I/O モードオプションは、ゲストのインストール中に **virt-manager** を使用して設定するか、既存のゲスト仮想マシンでゲスト XML 設定を編集して設定できます。

表7.2 IO モードオプション

IO モードオプション	説明
IO=native	Red Hat Virtualization (RHV) 環境のデフォルト。このモードは、ダイレクト I/O オプションを備えたカーネル非同期 I/O を指します。
IO=threads	デフォルトは、ホストユーザーモードベースのスレッドです。
IO=default	Red Hat Enterprise Linux 7 のデフォルトはスレッドモードです。

virt-manager では、I/O モードは **Virtual Disk** で指定できます。**virt-manager** を使用して I/O モードを変更する方法については、「仮想ディスクパフォーマンスのオプション」 を参照してください。

ゲスト XML で I/O モードを設定するには、**driver** タグ内の **io** 設定を編集し、***native***、***threads***、または ***default*** を指定します。たとえば、I/O モードを ***threads*** に設定するには、以下を実行します。

```
<disk type='file' device='disk'>
        <driver name='qemu' type='raw' io='threads'/>
```

7.4. ブロック I/O チューニング技術

このセクションでは、仮想化環境でブロック I/O のパフォーマンスをチューニングするためのその他の手法について説明します。

7.4.1. ディスク I/O スロットリング

複数の仮想マシンが同時に実行する場合は、過剰なディスク I/O により、システムパフォーマンスに影響が及ぶ可能性があります。KVM のディスク I/O スロットリングでは、仮想マシンからホストマシンに送られるディスク I/O 要求に制限を設定する機能を利用できます。これにより、仮想マシンが共有リソースを過剰に使用して、他の仮想マシンのパフォーマンスに影響を与えるのを防ぐことができます。

ディスク I/O スロットリングは、異なる顧客に属するゲスト仮想マシンが同じホストで実行されている場合や、異なるゲストに QoS 保証が提供されている場合など、さまざまな状況で役立ちます。ディスク I/O スロットリングは、低速なディスクをシミュレートするために使用することもできます。

I/O スロットリングは、ゲストに割り当てられた各ブロックデバイスに個別に適用でき、スループットおよび I/O 操作の制限に対応します。**virsh blkdeviotune** コマンドを使用して仮想マシンの I/O 制限を設定します。

```
# virsh blkdeviotune virtual_machine device --parameter limit
```

Device は、仮想マシンに接続されているディスクデバイスのいずれかに対して、一意のターゲット名 (**<target dev='name'/>**) またはソースファイル (**<source file='name'/>**) を指定します。ディスクデバイス名のリストについては、**virsh domblklist** コマンドを使用してください。

オプションのパラメーターは次のとおりです。

total-bytes-sec

総スループットの上限 (バイト毎秒単位)。

read-bytes-sec

読み取りスループットの上限 (バイト毎秒単位)。

write-bytes-sec

書き込みスループットの上限 (バイト毎秒単位)。

total-iops-sec

1 秒あたりの I/O 操作の制限合計。

read-iops-sec

1 秒あたりの読み取り I/O 操作数の制限。

write-iops-sec

1 秒あたりの書き込み I/O 操作数の制限。

たとえば、**virtual_machine** の **vda** を 1 秒あたり 1000 の I/O 操作と、1 秒スループットあたり 50 MB にスロットリングするには、次のコマンドを実行します。

```
# virsh blkdeviotune virtual_machine vda --total-iops-sec 1000 --total-bytes-sec 52428800
```

7.4.2. マルチキュー virtio-scsi

マルチキュー virtio-scsi は、virtio-scsi ドライバーのストレージパフォーマンスとスケーラビリティを向上させます。このため、各仮想 CPU に別個のキューを持たせることが可能になります。また仮想 CPU は、その他の vCPU に影響を及ぼすことなく使用するために、割り込みできるようになります。

7.4.2.1. マルチキュー virtio-scsi の設定

マルチキュー virtio-scsi は、Red Hat Enterprise Linux 7 ではデフォルトで無効になっています。

ゲストでマルチキュー virtio-scsi サポートを有効にするには、ゲストの XML 設定に以下を追加します。ここでの *N* は、vCPU キューの合計数です。

```
<controller type='scsi' index='0' model='virtio-scsi'>
<driver queues='N' />
 </controller>
```

第8章 メモリー

この章では、仮想化環境のメモリー最適化オプションについて説明します。

8.1. メモリーチューニングのヒント

仮想化環境でメモリーパフォーマンスを最適化するには、次のことを考慮してください。

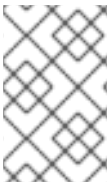
- ゲストが使用するよりも多くのリソースをゲストに割り当てないでください。
- 可能であれば、ゲストを単一の NUMA ノードに割り当てます。ただし、その NUMA ノードでリソースが十分であることを条件とします。NUMA の使用の詳細については、9章*NUMA* を参照してください。

8.2. 仮想マシンでのメモリーチューニング

8.2.1. メモリーモニターリングツール

ベアメタル環境で使用されるツールを使用して、仮想マシンでメモリー使用量をモニターリングできます。メモリー使用量のモニターリングとメモリー関連の問題の診断に役立つツールは次のとおりです。

- **top**
- **vmstat**
- **numastat**
- **/proc/**



注記

これらのパフォーマンスツールの使用の詳細については、『Red Hat Enterprise Linux 7 Performance Tuning Guide』 およびこれらのコマンドの man ページを参照してください。

8.2.2. virsh によるメモリーチューニング

ゲスト XML 設定のオプションの **<memtune>** 要素を使用すると、管理者はゲスト仮想マシンのメモリー設定を手動で設定できます。**<memtune>** を省略した場合、仮想マシンは、仮想マシンの作成時に割り当てられた方法に基づいてメモリーを使用します。

仮想マシンの **<memtune>** 要素でメモリーパラメーターを表示または設定するには、**virsh memtune** コマンドで、実際の環境に応じて値を置き換えてください。

```
# virsh memtune virtual_machine --parameter size
```

オプションのパラメーターは次のとおりです。

hard_limit

仮想マシンが使用できる最大メモリー。単位はキビバイト (1024 バイトのブロック) です。

警告

この制限を低く設定しすぎると、仮想マシンがカーネルによって強制終了される可能性があります。

soft_limit

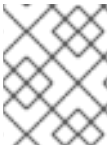
メモリー競合中に強制するメモリー制限。単位はキビバイト (1024 バイトのブロック) です。

swap_hard_limit

仮想マシンが使用できる最大メモリーとスワップ。単位はキビバイト (1024 バイトのブロック) です。***swap_hard_limit*** 値は、***hard_limit*** 値よりも大きくなければなりません。

min_guarantee

仮想マシンに保証される最小メモリー割り当て。単位はキビバイト (1024 バイトのブロック) です。



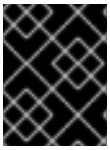
注記

virsh memtune コマンドの使用の詳細については、**# virsh help memtune** を参照してください。

オプションの **<memoryBacking>** 要素には、仮想メモリーページがホストページによってどのようにバックアップされるかに影響を与えるいくつかの要素が含まれる場合があります。

locked を設定すると、ホストはゲストに属するメモリーページをスワップアウトできなくなります。ゲスト XML に以下を追加して、ホストのメモリー内の仮想メモリーページをロックします。

```
<memoryBacking>
      <locked/>
</memoryBacking>
```



重要

locked を設定する場合は、**<memtune>** 要素で***hard_limit***を、ゲストに設定された最大メモリーと プロセス自体で消費されるメモリーに設定する必要があります。

nosharepages を設定すると、ホストはゲスト間で使用されているのと同じメモリーをマージできなくなります。ハイパーバイザーがゲストの共有ページを無効にするように指示するには、ゲストの XML に以下を追加します。

```
<memoryBacking>
      <nosharepages/>
</memoryBacking>
```

8.2.3. Huge Page と Transparent Huge Page

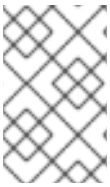
AMD64 および Intel 64 CPU は通常、4kB ページのメモリーを指定しますが、*Huge Page* と呼ばれる 2 MB または 1 GB の大きなページを使用できます。KVM ゲストは、*Transaction Lookaside Buffer* (TLB) に対する CPU キャッシュヒットを増やすことでパフォーマンスを向上させるために、Huge Page メモリーサポートを使用してデプロイできます。

Red Hat Enterprise Linux 7 でデフォルトで有効になっているカーネル機能は、特に大容量メモリーとメモリーを大量に消費するワークロードなど、Huge Page によってパフォーマンスが大幅に向上する可能性があります。Red Hat Enterprise Linux 7 は、Huge Page を使用してページサイズを大きくすることにより、大量のメモリーをより効果的に管理できます。Huge Page 管理の効率と利便性を高めるために、Red Hat Enterprise Linux 7 はデフォルトで *Transparent Huge Pages* (THP) を使用します。Huge Page と THP の詳細については、Performance Tuning Guide を参照してください。

Red Hat Enterprise Linux 7 システムは、2 MB および 1 GB の Huge Page をサポートしており、これは起動時または実行時に割り当てることができます。複数の Huge Page サイズを有効にする手順については 「起動時またはランタイムにおけるゲスト用 1 GB Huge Page の有効化」 を参照してください。

8.2.3.1. Transparent Huge Page の設定

Transparent Huge Page (THP) は、Huge Page の作成、管理、および使用のほとんどの要素を自動化する抽象化レイヤーです。デフォルトでは、パフォーマンスに関するシステム設定を自動的に最適化します。



注記

KSM を使用すると、Transparent Huge Page の発生を減らすことができるため、THP を有効にする前に KSM を無効にすることをお勧めします。詳細は、「KSM の非アクティブ化」 を参照してください。

Transparent Huge Page はデフォルトで有効になります。現在のステータスを確認するには、次のコマンドを実行します。

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

Transparent Huge Page をデフォルトで使用できるようにするには、以下を実行します。

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

これにより、**/sys/kernel/mm/transparent_hugepage/enabled** が ***always*** に設定されます。

Transparent Huge Pages (THP) を無効にするには、以下を実行します。

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

Transparent Huge Page をサポートしていても、静的 Huge Page は使用できます。ただし、静的 Huge Page が使用されていない場合、KVM は通常の 4kB ページサイズの代わりに Transparent Huge Page を使用します。

8.2.3.2. 静的 Huge Page の設定

場合によっては、Huge Page をより細かく制御することが望ましい場合があります。ゲストで静的 Huge Page を使用するには、**virsh edit** を使用してゲスト XML 設定に以下を追加します。

```
<memoryBacking>
      <hugepages/>
</memoryBacking>
```

これは、デフォルトのページサイズを使用する代わりに、Huge Page を使用してゲストにメモリーを割り当てるようにホストに指示します。

次のコマンドを実行して、現在の Huge Page の値を表示します。

```
cat /proc/sys/vm/nr_hugepages
```

手順8.1 Huge Page の設定

次の手順の例は、Huge Page を設定するコマンドを示しています。

1. 現在の Huge Page の値を表示します。

```
# cat /proc/meminfo | grep Huge
AnonHugePages:      2048 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

2. Huge Page は 2 MB 単位で設定されます。Huge Page の数を 25000 に設定するには、次のコマンドを使用します。

```
echo 25000 > /proc/sys/vm/nr_hugepages
```

注記

設定を永続的にするには、ゲストマシンの **/etc/sysctl.conf** ファイルに次の行を追加します。X は意図した Huge Page の数です。

```
# echo 'vm.nr_hugepages = X' >> /etc/sysctl.conf
# sysctl -p
```

その後、ゲストの **/etc/grub2.cfg** ファイルの **/kernel** 行の末尾に **transparent_hugepage=never** を追加します。

3. Huge Page をマウントします。

```
# mount -t hugetlbfs hugetlbfs /dev/hugepages
```

4. 仮想マシンの XML 設定の **memoryBacking** セクションに次の行を追加します。

```
<hugepages>
  <page size='1' unit='GiB'/>
</hugepages>
```

5. **libvirtd** を再起動します。

```
# systemctl restart libvirtd
```

6. - 仮想マシンを起動します。

   ```
   # virsh start virtual_machine
   ```

   - 仮想マシンがすでに実行中の場合は再起動します。

   ```
   # virsh reset virtual_machine
   ```

7. **/proc/meminfo** の変更を確認します。

```
# cat /proc/meminfo | grep Huge
AnonHugePages:         0 kB
HugePages_Total:   25000
HugePages_Free:    23425
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

Huge Page は、ホストだけでなくゲストにもメリットをもたらしますが、Huge Page の合計値は、ホストで利用可能な値よりも小さくなければなりません。

8.2.3.3. 起動時またはランタイムにおけるゲスト用 1 GB Huge Page の有効化

Red Hat Enterprise Linux 7 システムは、2 MB および 1 GB の Huge Page をサポートしており、これは起動時または実行時に割り当てることができます。

手順8.2 起動時の 1 GB Huge Page の割り当て

1. 起動時にさまざまなサイズの Huge Page を割り当てるには、次のコマンドを使用して Huge Page の数を指定します。この例では、4 つの 1 GB の Huge Page と 1024 の 2 MB の Huge Page を割り当てます。

   ```
   'default_hugepagesz=1G hugepagesz=1G hugepages=4 hugepagesz=2M hugepages=1024'
   ```

   このコマンドラインを変更して、起動時に割り当てる Huge Page の別の数を指定します。

   注記

   次の 2 つの手順も、起動時に 1 GB の Huge Page を初めて割り当てるときに完了する必要があります。

2. 2 MB と 1 GB の Huge Page をホストにマウントします。

```
# mkdir /dev/hugepages1G
# mount -t hugetlbfs -o pagesize=1G none /dev/hugepages1G
# mkdir /dev/hugepages2M
# mount -t hugetlbfs -o pagesize=2M none /dev/hugepages2M
```

3. 仮想マシンの XML 設定の **memoryBacking** セクションに次の行を追加します。

```
<hugepages>
  <page size='1' unit='GiB'/>
</hugepages>
```

4. libvirtd を再起動して、ゲストで 1 GB の Huge Page を使用できるようにします。

```
# systemctl restart libvirtd
```

手順8.3 ランタイムの 1 GB Huge Page の割り当て

ランタイムに 1 GB の Huge Page を割り当てることもできます。ランタイムの割り当てにより、システム管理者はそれらのページを割り当てる NUMA ノードを選択できます。ただし、ランタイムのページ割り当ては、メモリーの断片化が原因で、起動時の割り当てよりも割り当てに失敗する可能性が高くなります。

1. ランタイムでさまざまなサイズの Huge Page を割り当てるには、次のコマンドを使用して、Huge Page の数、それらを割り当てる NUMA ノード、および Huge Page サイズの値を置き換えます。

```
# echo 4 > /sys/devices/system/node/node1/hugepages/hugepages-
1048576kB/nr_hugepages
# echo 1024 > /sys/devices/system/node/node3/hugepages/hugepages-
2048kB/nr_hugepages
```

このコマンドの例では、**node1** から 4 つの 1 GB の Huge Page を割り当て、 **node3** から 1024 の 2 MB の Huge Page を割り当てます。

これらの Huge Page 設定は、ホストシステムの空きメモリーの量に応じて、上記のコマンドでいつでも変更できます。

注記

次の 2 つの手順も、ランタイムで 1 GB の Huge Page を初めて割り当てるときに完了する必要があります。

2. 2 MB と 1 GB の Huge Page をホストにマウントします。

```
# mkdir /dev/hugepages1G
# mount -t hugetlbfs -o pagesize=1G none /dev/hugepages1G
# mkdir /dev/hugepages2M
# mount -t hugetlbfs -o pagesize=2M none /dev/hugepages2M
```

3. 仮想マシンの XML 設定の **memoryBacking** セクションに次の行を追加します。

```
<hugepages>
  <page size='1' unit='GiB'/>
</hugepages>
```

4. libvirtd を再起動して、ゲストで 1 GB の Huge Page を使用できるようにします。

```
# systemctl restart libvirtd
```

8.3. KERNEL SAME-PAGE MERGING (KSM)

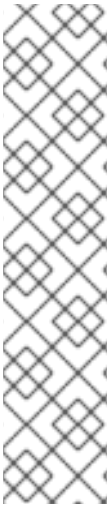
KVM ハイパーバイザーが使用する Kernel same-page Merging (KSM) により、KVM ゲストは同じメモリーページを共有できます。これらの共有ページは通常、一般的なライブラリーまたは他の同一の使用頻度の高いデータです。KSM を使用すると、メモリーの重複を避けることで、同一または同様のゲストオペレーティングシステムのゲスト密度が大きくなります。

共有メモリーの概念は、最新のオペレーティングシステムでは一般的です。たとえば、プログラムが最初に起動されると、そのプログラムはすべてのメモリーを親プログラムと共有します。子プログラムまたは親プログラムがこのメモリーを変更しようとすると、カーネルは新しいメモリー領域を割り当て、元のコンテンツをコピーして、プログラムがこの新しい領域を変更できるようにします。これは、コピーオンライトとして知られています。

KSM は、この概念を逆に使用する Linux 機能です。KSM を使用すると、カーネルはすでに実行中の 2 つ以上のプログラムを検証し、それらのメモリーを比較できます。いずれかのメモリー領域またはページが同一である場合、KSM は複数の同一のメモリーページを 1 つのページに減らします。このページは、コピーオンライトとしてマークされます。ページのコンテンツがゲスト仮想マシンによって変更された場合、そのゲスト用に新しいページが作成されます。

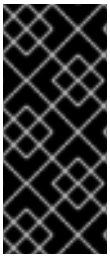
これは、KVM を使用した仮想化に役立ちます。ゲスト仮想マシンが起動すると、ホスト **qemu-kvm** プロセスからのみメモリーを継承します。ゲストが実行されると、ゲストが同じオペレーティングシステムまたはアプリケーションを実行しているときに、ゲストオペレーティングシステムイメージのコンテンツを共有できます。KSM を使用すると、KVM はこれらの同一のゲストメモリー領域を共有するように要求できます。

KSM は、メモリーの速度と使用率を強化します。KSM では、共通のプロセスデータがキャッシュまたはメインメモリーに保存されます。これにより、KVM ゲストのキャッシュミスが減少し、一部のアプリケーションとオペレーティングシステムのパフォーマンスが向上します。次に、メモリーを共有すると、ゲストの全体的なメモリー使用量が削減され、リソースの密度と使用率が向上します。



注記

Red Hat Enterprise Linux 7 では、KSM は NUMA に対応しています。これにより、ページのコアレッシング中に NUMA の局所性を考慮できるため、ページがリモートノードに移動されることに関連するパフォーマンスの低下を防ぐことができます。Red Hat は、KSM が使用されている場合に、ノード間のメモリーマージを回避することを推奨します。KSM を使用している場合は、**/sys/kernel/mm/ksm/merge_across_nodes** を ***0*** に変更して、NUMA ノード間でページがマージされないようにします。これは、**virsh node-memory-tune --shm-merge-across-nodes 0** コマンドを使用して実行できます。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をマージすると、numad が混乱する可能性があります。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。NUMA の詳細については 9章*NUMA* を参照してください。



重要

KSM を考慮しなくても、スワップサイズがコミットされた RAM に対して十分であることを確認してください。KSM は、同一または類似のゲストの RAM 使用量を削減します。十分なスワップスペースがない状態でゲストを KSM でオーバーコミットすることは可能ですが、ゲスト仮想マシンのメモリーを使用するとページが非共有になる可能性があるため、お勧めしません。

Red Hat Enterprise Linux は、KSM を制御するために 2 つの異なる方法を使用します。

- **ksm サービス** は、KSM カーネルスレッドを開始および停止します。
- **ksmtuned サービス** は、**ksm** サービスを制御およびチューニングし、same-page マージを動的に管理します。**ksmtuned** は、**ksm** サービスを開始し、メモリー共有が不要な場合は **ksm** サービスを停止します。新しいゲストが作成または破棄された場合、**ksmtuned** に ***retune*** パラメーターを実行するように指示する必要があります。

これらのサービスは両方とも、標準のサービス管理ツールで制御されます。

注記

Red Hat Enterprise Linux 6.7 では、KSM はデフォルトでオフになっています。

8.3.1. KSM サービス

- **ksm** サービスは qemu-kvm パッケージに含まれています。
- **ksm** サービスが開始されていない場合、Kernel same-page merging (KSM) は 2000 ページのみを共有します。このデフォルト値を使用すると、メモリー節約の利点が限定されます。
- **ksm** サービスが開始されると、KSM はホストシステムのメインメモリーの最大半分を共有します。**ksm** サービスを開始して、KSM がより多くのメモリーを共有できるようにしてください。

```
# systemctl start ksm
Starting ksm:                                  [  OK  ]
```

ksm サービスは、デフォルトの起動シーケンスに追加できます。systemctl コマンドを使用して **ksm** サービスを永続化します。

```
# systemctl enable ksm
```

8.3.2. KSM チューニングサービス

ksmtuned サービスは、**ksm** をループおよびチューニングすることにより、kernel same-page merging (KSM) の設定を微調整します。さらに、ゲスト仮想マシンが作成または破棄されると、**ksmtuned** サービスに libvirt から通知されます。**ksmtuned** サービスにはオプションがありません。

```
# systemctl start ksmtuned
Starting ksmtuned:                             [  OK  ]
```

ksmtuned サービスは、チューニング機能を手動で実行するように **ksmtuned** に指示する ***retune*** パラメーターを使用してチューニングできます。

/etc/ksmtuned.conf ファイルは、**ksmtuned** サービスの設定ファイルです。以下のファイル出力は、デフォルトの **ksmtuned.conf** ファイルです。

```
# Configuration file for ksmtuned.
# How long ksmtuned should sleep between tuning adjustments
# KSM_MONITOR_INTERVAL=60

# Millisecond sleep between ksm scans for 16Gb server.
# Smaller servers sleep more, bigger sleep less.
# KSM_SLEEP_MSEC=10
```

```
# KSM_NPAGES_BOOST - is added to the `npages` value, when `free memory` is less than `thres`.
# KSM_NPAGES_BOOST=300

# KSM_NPAGES_DECAY - is the value given is subtracted to the `npages` value, when `free
memory` is greater than `thres`.
# KSM_NPAGES_DECAY=-50

# KSM_NPAGES_MIN - is the lower limit for the `npages` value.
# KSM_NPAGES_MIN=64

# KSM_NPAGES_MAX - is the upper limit for the `npages` value.
# KSM_NPAGES_MAX=1250

# KSM_THRES_COEF - is the RAM percentage to be calculated in parameter `thres`.
# KSM_THRES_COEF=20

# KSM_THRES_CONST - If this is a low memory system, and the `thres` value is less than
`KSM_THRES_CONST`, then reset `thres` value to `KSM_THRES_CONST` value.
# KSM_THRES_CONST=2048

# uncomment the following to enable ksmtuned debug information
# LOGFILE=/var/log/ksmtuned
# DEBUG=1
```

/etc/ksmtuned.conf ファイル内で、**npages** は、**ksmd** デーモンが非アクティブになる前に **ksm** がスキャンするページ数を設定します。この値は、**/sys/kernel/mm/ksm/pages_to_scan** ファイルでも設定されます。

The**KSM_THRES_CONST** の値は、**ksm** をアクティブ化するためのしきい値として使用される使用可能なメモリーの量を表します。次のいずれかが発生した場合、**ksmd** がアクティブになります。

- 空きメモリーの量が、**KSM_THRES_CONST** で設定されたしきい値を下回る。
- コミットされたメモリーの量としきい値 **KSM_THRES_CONST** が、メモリーの合計量を超えている。

8.3.3. KSM 変数とモニターリング

Kernel same-page merging (KSM) は、モニターリングデータを **/sys/kernel/mm/ksm/** ディレクトリーに保存します。このディレクトリー内のファイルは、カーネルによって更新される、KSM の使用状況と統計の正確な記録です。

以下のリストの変数は、上記のように、**/etc/ksmtuned.conf** ファイルの設定可能な変数でもあります。

/sys/kernel/mm/ksm/ 内のファイル:

full_scans

フルスキャンが実行されます。

merge_across_nodes

異なる NUMA ノードからのページをマージできるかどうか。

pages_shared

共有されたページの総数。

pages_sharing
現在共有されているページ。

pages_to_scan
スキャンされていないページ。

pages_unshared
共有されなくなったページ。

pages_volatile
揮発性ページの数。

run
KSM プロセスが実行されているかどうか。

sleep_millisecs
スリープのミリ秒。

これらの変数は、**virsh node-memory-tune** コマンドを使用して手動でチューニングできます。たとえば、以下は、共有メモリーサービスがスリープ状態になる前にスキャンするページ数を指定します。

```
# virsh node-memory-tune --shm-pages-to-scan number
```

DEBUG=1 行が **/etc/ksmtuned.conf** ファイルに追加された場合、KSM チューニングアクティビティーは **/var/log/ksmtuned** ログファイルに保存されます。ログファイルの場所は、***LOGFILE*** パラメーターを使用して変更できます。ログファイルの場所を変更することはお勧めできません。SELinux 設定の特別な設定が必要になる場合があります。

8.3.4. KSM の非アクティブ化

Kernel same-page merging (KSM) には、特定の環境またはホストシステムで大きすぎる可能性があるパフォーマンスのオーバーヘッドがあります。KSM は、ゲスト間で情報を漏洩するために使用される可能性のあるサイドチャネルを導入する場合もあります。これが懸念される場合は、ゲストごとに KSM を無効にすることができます。

KSM は、**ksmtuned** および **ksm** サービスを停止することで非アクティブ化できます。ただし、このアクションは再起動後に維持されません。KSM を非アクティブ化するには、ターミナルで root として以下を実行します。

```
# systemctl stop ksmtuned
Stopping ksmtuned:                         [  OK  ]
# systemctl stop ksm
Stopping ksm:                              [  OK  ]
```

ksmtuned と **ksm** を停止すると、KSM が非アクティブ化されますが、このアクションは再起動後に維持されません。KSM を永続的に非アクティブ化するには、**systemctl** コマンドを使用します。

```
# systemctl disable ksm
# systemctl disable ksmtuned
```

KSM が無効になっている場合でも、KSM を非アクティブ化する前に共有されていたメモリーページはすべて共有されます。システム内のすべての PageKSM を削除するには、次のコマンドを使用します。

```
# echo 2 >/sys/kernel/mm/ksm/run
```

これが実行されると、**khugepaged** デーモンは KVM ゲストの物理メモリーに Transparent Huge Page を再構築できます。# **echo 0 >/sys/kernel/mm/ksm/run** を使用すると、KSM が停止しますが、以前に作成されたすべての KSM ページの共有は解除されません (これは # **systemctl stop ksmtuned** コマンドと同じです)。

第9章 NUMA

以前から、AMD64 および Intel 64 システムのすべてのメモリーは、すべての CPU から等しくアクセスできました。Uniform Memory Access (UMA) と呼ばれるアクセス時間は、どの CPU が操作を実行しても同じです。

この動作は、最新の AMD64 および Intel 64 プロセッサーには当てはまりません。Non-Uniform Memory Access (NUMA) では、システムメモリーは NUMA**ノード** 間で分割されます。NUMA ノードは、システムメモリーのローカルサブセットと同じアクセスレイテンシーを持つソケットまたは特定の CPU セットに対応します。

この章では、仮想化環境でのメモリー割り当てと NUMA チューニング設定について説明します。

9.1. NUMA メモリーの割り当てポリシー

次のポリシーは、システム内のノードからメモリーがどのように割り当てられるかを定義します。

Strict

Strict ポリシーとは、ターゲットノードにメモリーを割り当てることができない場合に割り当てが失敗することを意味します。

メモリーモード属性を定義せずに NUMA ノードセットリストを指定すると、デフォルトで ***strict*** モードになります。

Interleave

メモリーページは、ノードセットで指定されたノード間で割り当てられますが、ラウンドロビン方式で割り当てられます。

Preferred

メモリーは、単一の優先メモリーノードから割り当てられます。十分なメモリーが利用できない場合は、他のノードからメモリーを割り当てることができます。

目的のポリシーを有効にするには、ドメイン XML ファイルの **<memory mode>** 要素の値として設定します。

```
<numatune>
      <memory mode='preferred' nodeset='0'>
</numatune>
```

重要

strict モードでメモリーがオーバーコミットされ、ゲストに十分なスワップスペースがない場合、カーネルは一部のゲストプロセスを強制終了して、追加のメモリーを取得します。Red Hat は、この状況を防ぐために、***preferred*** 割り当てを使用し、単一のノードセット (たとえば nodeset = '0') を指定することをお勧めします。

9.2. 自動 NUMA バランシング

自動 NUMA バランシングにより、NUMA ハードウェアシステムで実行されているアプリケーションのパフォーマンスが向上します。これは、Red Hat Enterprise Linux 7 システムではデフォルトで有効になっています。

アプリケーションは通常、プロセスのスレッドがスケジュールされているのと同じ NUMA ノード上のメモリーにアクセスしているときに最高のパフォーマンスを発揮します。自動 NUMA バランシングは、スレッドまたはプロセスである可能性のあるタスクを、アクセスしているメモリーの近くに移動します。また、アプリケーションデータを、それを参照するタスクに近いメモリーに移動します。自動 NUMA バランシングがアクティブになっている場合、これはすべてカーネルによって自動的に実行されます。

自動 NUMA バランシングは、多くのアルゴリズムとデータ構造を使用します。これらは、システムで自動 NUMA バランシングがアクティブな場合にのみアクティブになり、割り当てられます。

- プロセスメモリーの定期的な NUMA マッピング解除
- NUMA ヒント障害
- Migrate-on-Fault (MoF) - メモリーを、それを使用するプログラムが実行される場所に移動します
- task_numa_placement - メモリーを実行中のプログラムの近くに移動します

9.2.1. 自動 NUMA バランシングの設定

自動 NUMA バランシングは、Red Hat Enterprise Linux 7 ではデフォルトで有効になっており、NUMA プロパティーを持つハードウェアで起動すると自動的にアクティブになります。

自動 NUMA バランシングは、次の両方の条件が満たされた場合に有効になります。

- **# numactl --hardware** shows multiple nodes
- **# cat /proc/sys/kernel/numa_balancing** shows **1**

アプリケーションの手動 NUMA チューニングは、自動 NUMA バランシングをオーバーライドし、メモリーの定期的なマッピング解除、NUMA 障害、移行、およびそれらのアプリケーションの自動 NUMA 配置を無効にします。

場合によっては、システム全体の手動 NUMA チューニングが推奨されます。

自動 NUMA バランシングを無効にするには、次のコマンドを使用します。

```
# echo 0 > /proc/sys/kernel/numa_balancing
```

自動 NUMA バランシングを有効にするには、次のコマンドを使用します。

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

9.3. LIBVIRT NUMA チューニング

通常、NUMA システムで最高のパフォーマンスを実現するには、ゲストのサイズを単一の NUMA ノード上のリソースの量に制限します。NUMA ノード間でリソースを不必要に分割することは避けてください。

numastat ツールを使用して、プロセスとオペレーティングシステムの NUMA ノードごとのメモリー統計を表示します。

次の例では、**numastat** ツールは、NUMA ノード間で準最適なメモリーアライメントを持つ 4 つの仮想マシンを示しています。

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)     68     16    357   6936      2      3    147    598  8128
51747 (qemu-kvm)    245     11      5     18   5172   2532      1     92  8076
53736 (qemu-kvm)     62    432   1661    506   4851    136     22    445  8116
53773 (qemu-kvm)   1393      3      1      2     12      0      0   6702  8114
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total              1769    463   2024   7462  10037   2672    169   7837 32434
```

numad を実行して、ゲストの CPU とメモリーリソースを自動的に調整します。

次に、**numastat -c qemu-kvm** を再度実行して、**numad** の実行結果を表示します。次の出力は、リソースが調整されたことを示しています。

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
51747 (qemu-kvm)      0      0      7      0   8072      0      1      0  8080
53736 (qemu-kvm)      0      0      7      0      0      0   8113      0  8120
53773 (qemu-kvm)      0      0      7      0      0      0      1   8110  8118
59065 (qemu-kvm)      0      0   8050      0      0      0      0      0  8051
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                 0      0   8072      0   8072      0   8114   8110 32368
```

注記

-c で **numastat** を実行すると、コンパクトな出力が得られます。**-m** オプションを追加すると、ノードごとにシステム全体のメモリー情報が出力に追加されます。詳細は、**numastat** の man ページを参照してください。

9.3.1. ホスト NUMA ノードごとのメモリーのモニターリング

nodestats.py スクリプトを使用して、ホスト上の各 NUMA ノードの合計メモリーと空きメモリーを報告できます。このスクリプトは、実行中のドメインごとに、特定のホストノードに厳密にバインドされているメモリーの量も報告します。以下に例を示します。

```
# /usr/share/doc/libvirt-python-2.0.0/examples/nodestats.py
NUMA stats
NUMA nodes:     0       1       2       3
MemTotal:       3950    3967    3937    3943
MemFree:        66      56      42      41
Domain 'rhel7-0':
        Overall memory: 1536 MiB
Domain 'rhel7-1':
        Overall memory: 2048 MiB
Domain 'rhel6':
        Overall memory: 1024 MiB nodes 0-1
        Node 0: 1024 MiB nodes 0-1
Domain 'rhel7-2':
```

```
Overall memory: 4096 MiB nodes 0-3
Node 0: 1024 MiB nodes 0
Node 1: 1024 MiB nodes 1
Node 2: 1024 MiB nodes 2
Node 3: 1024 MiB nodes 3
```

この例は、4 つのホスト NUMA ノードを示しています。各ノードには合計で約 4 GB の RAM が含まれています (***MemTotal***)。ほぼすべてのメモリーが各ドメインで消費されます (***MemFree***)。4 つのドメイン (仮想マシン) が実行されています。ドメイン *'rhel7-0'* には 1.5 GB のメモリーがあり、特定のホスト NUMA ノードに固定されていません。ただし、ドメイン *'rhel7-2'* には、4 GB のメモリーと 4 つの NUMA ノードがあり、ホストノードに 1:1 で固定されています。

ホスト NUMA ノードの統計情報を出力するには、ご使用の環境の **nodestats.py** スクリプトを作成します。スクリプトの例は、**/usr/share/doc/libvirt-python-*version*/examples/nodestats.py** の libvirt-python パッケージファイルにあります。スクリプトへの特定のパスは、**rpm -ql libvirt-python** コマンドを使用して表示できます。

9.3.2. NUMA vCPU のピニング

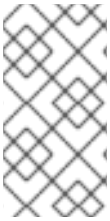
vCPU のピニングは、ベアメタルシステムでのタスクのピニングと同様の利点を提供します。vCPU は、ホストのオペレーティングシステムのユーザー空間タスクとして実行されるので、ピニングすることでキャッシュの効率性が向上します。この一例は、すべての vCPU スレッドが同じ物理ソケットで実行されているため、L3 キャッシュドメインを共有している環境です。

注記

Red Hat Enterprise Linux バージョン 7.0 から 7.2 の場合、アクティブな vCPU のみピニングできます。ただし、Red Hat Enterprise Linux 7.3 では、非アクティブな vCPU もピニングできます。

vCPU のピニングと **numatune** を組み合わせると、NUMA ミスを回避できます。NUMA を使用しない場合のパフォーマンスへの影響は大きく、一般的に、パフォーマンスの 10 % 以上が影響を受けます。vCPU ピニングと **numatune** は、一緒に設定する必要があります。

仮想マシンがストレージまたはネットワーク I/O タスクを実行している場合、すべての vCPU とメモリーを I/O アダプターに物理的に接続されている同じ物理ソケットにピニングすると便利な場合があります。



注記

lstopo ツールを使用して、NUMA トポロジーを視覚化できます。また、vCPU が同じ物理ソケット上のコアにバインドされていることを確認するのにも役立ちます。**lstopo** の詳細については、次のナレッジベースのアーティクルを参照してください: https://access.redhat.com/site/solutions/62879。

重要

物理コアよりも多くの vCPU が存在する場合、ピニングにより複雑さが増します。

次の XML 設定の例では、ドメインプロセスが物理 CPU 0 - 7 にピニングされています。vCPU スレッドは独自の cpuset にピニングされています。たとえば、vCPU0 は物理 CPU 0 にピニングされ、vCPU1 は物理 CPU1 にピニングされます。

```
<vcpu cpuset='0-7'>8</vcpu>
```

```
<cputune>
        <vcpupin vcpu='0' cpuset='0'/>
        <vcpupin vcpu='1' cpuset='1'/>
        <vcpupin vcpu='2' cpuset='2'/>
        <vcpupin vcpu='3' cpuset='3'/>
        <vcpupin vcpu='4' cpuset='4'/>
        <vcpupin vcpu='5' cpuset='5'/>
        <vcpupin vcpu='6' cpuset='6'/>
        <vcpupin vcpu='7' cpuset='7'/>
</cputune>
```

vcpu タグと vcpupin タグの間には直接的な関係があります。vcpupin オプションが指定されていない場合、値は自動的に決定され、親 vcpu タグオプションから継承されます。次の設定は、**vcpu 5** の **<vcpupin>** がないことを示しています。したがって、**vCPU5** は、親タグ **<vcpu>** で指定されているように、物理 CPU 0 - 7 にピニングされます。

```
<vcpu cpuset='0-7'>8</vcpu>
    <cputune>
        <vcpupin vcpu='0' cpuset='0'/>
        <vcpupin vcpu='1' cpuset='1'/>
        <vcpupin vcpu='2' cpuset='2'/>
        <vcpupin vcpu='3' cpuset='3'/>
        <vcpupin vcpu='4' cpuset='4'/>
        <vcpupin vcpu='6' cpuset='6'/>
        <vcpupin vcpu='7' cpuset='7'/>
    </cputune>
```

重要

最適で確定的なパフォーマンスを得るには、**<vcpupin>**、**<numatune>**、および **<emulatorpin>** を一緒に設定する必要があります。**<numatune>** タグの詳細については、「ドメインプロセス」 を参照してください。**<emulatorpin>** タグの詳細については、「emulatorpin の使用」 を参照してください。

9.3.3. ドメインプロセス

Red Hat Enterprise Linux で提供されているように、libvirt は libnuma を使用してドメインプロセスのメモリーバインディングポリシーを設定します。これらのポリシーのノードセットは、**static** (ドメイン XML で指定) または **auto** (numad クエリーによって設定) のいずれかに設定できます。これらを **<numatune>** タグ内で設定する方法の例は、以下の XML 設定を参照してください。

```
<numatune>
    <memory mode='strict' placement='auto'/>
</numatune>
```

```
<numatune>
    <memory mode='strict' nodeset='0,2-3'/>
</numatune>
```

libvirt は **sched_setaffinity(2)** を使用して、ドメインプロセスの CPU バインディングポリシーを設定します。cpuset オプションは、**static** (ドメイン XML で指定) または **auto** (numad クエリーによって設定) のいずれかです。これらを **<vcpu>** タグ内で設定する方法の例は、以下の XML 設定を参照してください。

```
<vcpu placement='auto'>8</vcpu>
```

```
<vcpu placement='static' cpuset='0-10,^5'>8</vcpu>
```

<vcpu> と **<numatune>** に使用する配置モードには、暗黙的な継承ルールがあります。

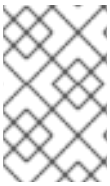
- **<numatune>** の配置モードは、デフォルトでは **<vcpu>** と同じ配置モードに設定され、**<nodeset>** が指定されている場合は *static* です。
- 同様に、**<vcpu>** の配置モードは、デフォルトでは **<numatune>** と同じ配置モードに設定され、**<cpuset>** が指定されている場合は *static* です。

つまり、ドメインプロセスの CPU チューニングとメモリーチューニングは別々に指定および定義できますが、他の配置モードに依存するように設定することもできます。

起動時にすべての vCPU をピニングせずに、選択した数の vCPU を起動するように numad を使用してシステムを設定することもできます。

たとえば、32 個の vCPU を備えたシステムで起動時に 8 個の vCPU のみを有効にするには、次のように XML を設定します。

```
<vcpu placement='auto' current='8'>32</vcpu>
```



注記

vcpu と numatune の詳細については、次の URL を参照してください:
http://libvirt.org/formatdomain.html#elementsCPUAllocation および
http://libvirt.org/formatdomain.html#elementsNUMATuning

9.3.4. ドメイン vCPU スレッド

ドメインプロセスのチューニングに加えて、libvirt は、XML 設定内の各 vcpu スレッドのピニングポリシーの設定も許可します。**<cputune>** タグ内の各 vcpu スレッドのピニングポリシーを設定します。

```
<cputune>
      <vcpupin vcpu="0" cpuset="1-4,^2"/>
      <vcpupin vcpu="1" cpuset="0,1"/>
      <vcpupin vcpu="2" cpuset="2,3"/>
      <vcpupin vcpu="3" cpuset="0,4"/>
</cputune>
```

このタグでは、libvirt は cgroup または **sched_setaffinity(2)** のいずれかを使用して、vcpu スレッドを指定された cpuset にピニングします。

注記

<cputune> の詳細については、次の URL を参照してください:
http://libvirt.org/formatdomain.html#elementsCPUTuning

さらに、単一の NUMA ノードよりも多くの vCPU を使用して仮想マシンをセットアップする必要がある場合は、ゲストがホスト上で NUMA トポロジーを検出するようにホストを設定します。これにより、CPU、メモリー、および NUMA ノードの 1:1 マッピングが可能になります。たとえば、これは 4 つ

の vCPU と 6 GB のメモリーを備えたゲスト、および次の NUMA 設定を備えたホストに適用できます。

```
4 available nodes (0-3)
Node 0: CPUs 0 4, size 4000 MiB
Node 1: CPUs 1 5, size 3999 MiB
Node 2: CPUs 2 6, size 4001 MiB
Node 3: CPUs 0 4, size 4005 MiB
```

このシナリオでは、次のドメイン XML 設定を使用します。

```
<cputune>
 <vcpupin vcpu="0" cpuset="1"/>
 <vcpupin vcpu="1" cpuset="5"/>
 <vcpupin vcpu="2" cpuset="2"/>
 <vcpupin vcpu="3" cpuset="6"/>
</cputune>
<numatune>
  <memory mode="strict" nodeset="1-2"/>
</numatune>
<cpu>
 <numa>
  <cell id="0" cpus="0-1" memory="3" unit="GiB"/>
  <cell id="1" cpus="2-3" memory="3" unit="GiB"/>
 </numa>
</cpu>
```

9.3.5. Cache Allocation Technology によるパフォーマンスの向上

特定の CPU モデルでカーネルが提供する Cache Allocation Technology (CAT) を利用できます。これにより、ホスト CPU のキャッシュの一部を vCPU スレッドに割り当てることができ、リアルタイムのパフォーマンスが向上します。

vCPU キャッシュの割り当てを **cachetune** タグ内で設定する方法の例は、以下の XML 設定を参照してください。

```
<domain>
  <cputune>
    <cachetune vcpus='0-1'>
      <cache id='0' level='3' type='code' size='3' unit='MiB'/>
      <cache id='0' level='3' type='data' size='3' unit='MiB'/>
    </cachetune>
  </cputune>
</domain>
```

上記の XML ファイルは、vCPU 0 および 1 のスレッドを、1 回は L3CODE に、もう 1 回は L3DATA に、最初の L3 キャッシュ (level='3' id='0') から 3 MiB が割り当てられるように設定します。

注記

1 つの仮想マシンに複数の **<cachetune>** 要素を設定できます。

詳細は、upstream **libvirt** ドキュメント の **cachetune** を参照してください。

9.3.6. emulatorpin の使用

ドメインプロセスのピニングポリシーを調整するもう 1 つの方法は、**<cputune>** 内の **<emulatorpin>** タグを使用することです。

<emulatorpin> タグは、**エミュレーター** (vCPU を含まないドメインのサブセット) がピニングされるホスト物理 CPU を指定します。**<emulatorpin>** タグは、エミュレータースレッドプロセスに正確なアフィニティーを設定する方法を提供します。その結果、vhost スレッドは物理 CPU とメモリーの同じサブセットで実行されるため、キャッシュ局所性を活用できます。たとえば、以下のようになります。

```
<cputune>
      <emulatorpin cpuset="1-3"/>
</cputune>
```

注記

Red Hat Enterprise Linux 7 では、自動 NUMA バランシングがデフォルトで有効になっています。自動 NUMA バランシングにより、vhost-net エミュレータースレッドは vCPU タスクをより確実に追跡するため、**<emulatorpin>** の手動チューニングの必要性は低下します。自動 NUMA バランシングの詳細については、「自動 NUMA バランシング」 を参照してください。

9.3.7. virsh を使用した vCPU ピニングのチューニング

重要

これらはサンプルコマンドです。環境に応じて値を置き換える必要があります。

次の **virsh** コマンドの例は、ID が 1 の vcpu スレッド *rhel7* を物理 CPU2 にピニングします。

```
% virsh vcpupin rhel7 1 2
```

virsh コマンドを使用して、現在の vcpu ピニング設定を取得することもできます。以下に例を示します。

```
% virsh vcpupin rhel7
```

9.3.8. virsh を使用したドメインプロセス CPU ピニングのチューニング

重要

これらはサンプルコマンドです。環境に応じて値を置き換える必要があります。

emulatorpin オプションは、各ドメインプロセスに関連付けられているスレッドに CPU アフィニティー設定を適用します。完全にピニングするには、各ゲストに **virsh vcpupin** (前に示したとおり) および **virsh emulatorpin** の両方を使用する必要があります。以下に例を示します。

```
% virsh emulatorpin rhel7 3-4
```

9.3.9. virsh を使用したドメインプロセスメモリーポリシーのチューニング

ドメインプロセスメモリーは動的にチューニングできます。次のコマンド例を参照してください。

```
% virsh numatune rhel7 --nodeset 0-10
```

これらのコマンドのその他の例は、**virsh** の man ページにあります。

9.3.10. ゲスト NUMA トポロジー

ゲスト NUMA トポロジーは、ゲスト仮想マシンの XML の **<cpu>** タグ内の **<numa>** タグを使用して指定できます。次の例を参照してください。値は適宜、置き換えます。

```
<cpu>
      ...
  <numa>
    <cell cpus='0-3' memory='512000'/>
    <cell cpus='4-7' memory='512000'/>
  </numa>
  ...
</cpu>
```

各 **<cell>** 要素は、NUMA セルまたは NUMA ノードを指定します。 **cpus** はノードの一部である CPU または CPU 範囲を指定し、**memory** はノードメモリーをキビバイト (1024 バイトのブロック) で指定します。各セルまたはノードには、0 から始まる昇順で **cellid** または **nodeid** が割り当てられます。

重要

CPU ソケット、コア、およびスレッドのトポロジーが設定されているゲスト仮想マシンの NUMA トポロジーを変更する場合は、単一のソケットに属するコアとスレッドが同じ NUMA ノードに割り当てられていることを確認してください。同じソケットのスレッドまたはコアが異なる NUMA ノードに割り当てられている場合、ゲストは起動に失敗する可能性があります。

警告

Huge Page と同時にゲスト NUMA トポロジーを使用することは、Red Hat Enterprise Linux 7 ではサポートされていません。Red Hat Virtualization や Red Hat OpenStack Platform などのレイヤード製品でのみ使用できます。

9.3.11. PCI デバイスの NUMA ノードの局所性

新しい仮想マシンを起動するときは、ホスト NUMA トポロジーと NUMA ノードへの PCI デバイスの所属の両方を知っておくことが重要です。これにより、PCI パススルーが要求されたときに、ゲストが正しい NUMA ノードにピニングされて最適なメモリーパフォーマンスが得られます。

たとえば、ゲストが NUMA ノード 0-1 にピニングされているが、その PCI デバイスの 1 つがノード 2 に所属する場合、ノード間のデータ転送には時間がかかります。

Red Hat Enterprise Linux 7.1 以降では、libvirt はゲスト XML 内の PCI デバイスの NUMA ノード局所性を報告し、管理アプリケーションがパフォーマンスに関するより良い決定を行えるようにします。

この情報は、**/sys/devices/pci*/*/numa_node** の **sysfs** ファイルに表示されます。これらの設定を確認する 1 つの方法は、**lstopo** ツールを使用して **sysfs** データを報告することです。

```
# lstopo-no-graphics
Machine (126GB)
  NUMANode L#0 (P#0 63GB)
    Socket L#0 + L3 L#0 (20MB)
      L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0 + PU L#0 (P#0)
      L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1 + PU L#1 (P#2)
      L2 L#2 (256KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2 + PU L#2 (P#4)
      L2 L#3 (256KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3 + PU L#3 (P#6)
      L2 L#4 (256KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4 + PU L#4 (P#8)
      L2 L#5 (256KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5 + PU L#5 (P#10)
      L2 L#6 (256KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6 + PU L#6 (P#12)
      L2 L#7 (256KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7 + PU L#7 (P#14)
    HostBridge L#0
      PCIBridge
        PCI 8086:1521
          Net L#0 "em1"
        PCI 8086:1521
          Net L#1 "em2"
        PCI 8086:1521
          Net L#2 "em3"
        PCI 8086:1521
          Net L#3 "em4"
      PCIBridge
        PCI 1000:005b
          Block L#4 "sda"
          Block L#5 "sdb"
          Block L#6 "sdc"
          Block L#7 "sdd"
      PCIBridge
        PCI 8086:154d
          Net L#8 "p3p1"
        PCI 8086:154d
          Net L#9 "p3p2"
      PCIBridge
        PCIBridge
          PCIBridge
            PCIBridge
              PCI 102b:0534
                GPU L#10 "card0"
                GPU L#11 "controlD64"
      PCI 8086:1d02
  NUMANode L#1 (P#1 63GB)
    Socket L#1 + L3 L#1 (20MB)
      L2 L#8 (256KB) + L1d L#8 (32KB) + L1i L#8 (32KB) + Core L#8 + PU L#8 (P#1)
      L2 L#9 (256KB) + L1d L#9 (32KB) + L1i L#9 (32KB) + Core L#9 + PU L#9 (P#3)
      L2 L#10 (256KB) + L1d L#10 (32KB) + L1i L#10 (32KB) + Core L#10 + PU L#10 (P#5)
      L2 L#11 (256KB) + L1d L#11 (32KB) + L1i L#11 (32KB) + Core L#11 + PU L#11 (P#7)
      L2 L#12 (256KB) + L1d L#12 (32KB) + L1i L#12 (32KB) + Core L#12 + PU L#12 (P#9)
      L2 L#13 (256KB) + L1d L#13 (32KB) + L1i L#13 (32KB) + Core L#13 + PU L#13 (P#11)
      L2 L#14 (256KB) + L1d L#14 (32KB) + L1i L#14 (32KB) + Core L#14 + PU L#14 (P#13)
      L2 L#15 (256KB) + L1d L#15 (32KB) + L1i L#15 (32KB) + Core L#15 + PU L#15 (P#15)
    HostBridge L#8
      PCIBridge
```

```
    PCI 1924:0903
      Net L#12 "p1p1"
    PCI 1924:0903
      Net L#13 "p1p2"
  PCIBridge
    PCI 15b3:1003
      Net L#14 "ib0"
      Net L#15 "ib1"
      OpenFabrics L#16 "mlx4_0"
```

この出力は次のことを示しています。

- NIC **em*** とディスク **sd*** は、NUMA ノード 0 とコア 0、2、4、6、8、10、12、14 に接続されている。
- NIC **p1*** と **ib*** は、NUMA ノード 1 およびコア 1、3、5、7、9、11、13、15 に接続されている。

9.4. NUMA-AWARE KERNEL SAMEPAGE MERGING (KSM)

Kernel SamePage Merging (KSM) を使用すると、仮想マシンで同一のメモリーページを共有できます。KSM は、システムが NUMA メモリーを使用していることを検出し、異なる NUMA ノード間でページのマージを制御できます。

sysfs /sys/kernel/mm/ksm/merge_across_nodes パラメーターを使用して、異なる NUMA ノード間でのページのマージを制御します。デフォルトでは、すべてのノードのページをマージできます。このパラメーターがゼロに設定されている場合、同じノードのページのみがマージされます。

通常、システムメモリーを過剰にサブスクライブしない限り、KSM 共有を無効にすることで、ランタイムのパフォーマンスが向上します。

重要

KSM が複数のゲスト仮想マシンを備えた NUMA ホスト上のノード間でマージする場合、より離れたノードからのゲストと CPU では、マージされた KSM ページへのアクセスレイテンシーが大幅に増加する可能性があります。

ハイパーバイザーがゲストの共有ページを無効にするように指示するには、ゲストの XML に以下を追加します。

```
<memoryBacking>
      <nosharepages/>
</memoryBacking>
```

<memoryBacking> 要素を使用したメモリー設定のチューニングの詳細については、「virsh によるメモリーチューニング」 を参照してください。

付録A 更新履歴

改訂 1.0-35 Thus May 23 2019 Jiri Herrmann
7.7 ベータ版公開用バージョン

改訂 1.0-34 Tue Oct 25 2018 Jiri Herrmann
7.6 GA リリースのバージョン

改訂 1.0-32 Tue Aug 14 2018 Jiri Herrmann
7.6 ベータ版公開用バージョン

改訂 1.0-31 Wed Apr 4 2018 Jiri Herrmann
7.5 GA 公開用バージョン

改訂 1.0-27 Mon Jul 27 2017 Jiri Herrmann
7.4 GA 公開用バージョン

改訂 1.0-24 Mon Oct 17 2016 Jiri Herrmann
7.3 GA 公開用バージョン

改訂 1.0-22 Mon Dec 21 2015 Laura Novich
いくつかのバグ修正によるガイドの再公開

改訂 1.0-19 Thu Oct 08 2015 Jiri Herrmann
改訂履歴の整理