Red Hat Enterprise Linux 7.3 Beta Virtualization Tuning and Optimization Guide

---

Optimizing your virtual environment

Jiri Herrmann
Yehuda Zimmerman
Dayle Parker
Scott Radvan
Red Hat Subject Matter Experts

Red Hat Enterprise Linux 7.3 Beta Virtualization Tuning and Optimization Guide

Optimizing your virtual environment

Jiri Herrmann
Red Hat Customer Content Services
jherrman@redhat.com

Yehuda Zimmerman
Red Hat Customer Content Services
yzimmerm@redhat.com

Dayle Parker
Red Hat Customer Content Services

Scott Radvan
Red Hat Customer Content Services

Red Hat Subject Matter Experts

Legal Notice

Abstract

The Red Hat Enterprise Linux Virtualization Tuning and Optimization Guide covers KVM and virtualization performance. Within this guide you can find tips and suggestions for making full use of KVM performance features and options for your host systems and virtualized guests. Note: This document is under development, is subject to substantial change, and is provided only as a preview. The included information and instructions should not be considered complete, and should be used with caution.

Table of Contents

Chapter 1. Introduction

1.1. KVM Performance Architecture Overview

The following points provide a brief overview of KVM as it pertains to system performance and process/thread management:

- When using KVM, guests run as a Linux process on the host.
- Virtual CPUs (vCPUs) are implemented as normal threads, handled by the Linux scheduler.
- Guests inherit features such as NUMA and huge pages from the kernel.
- Disk and network I/O settings in the host have a significant performance impact.
- Network traffic typically travels through a software-based bridge.

1.2. Virtualization Performance Features and Improvements

Virtualization Performance Improvements in Red Hat Enterprise Linux 7

The following features improve virtualization performance in Red Hat Enterprise Linux 7:

Automatic NUMA Balancing

Automatic NUMA balancing improves the performance of applications running on NUMA hardware systems, without any manual tuning required for Red Hat Enterprise Linux 7 guests. Automatic NUMA balancing moves tasks, which can be threads or processes, closer to the memory they are accessing.

For more information on automatic NUMA balancing, see Section 8.2, “Automatic NUMA Balancing”.

Multi-queue virtio-net

A networking approach that enables packet sending/receiving processing to scale with the number of available vCPUs of the guest.

For more information on multi-queue virtio-net, see Section 5.4.2, “Multi-Queue virtio-net”.

Bridge Zero Copy Transmit

Zero copy transmit mode reduces the host CPU overhead in transmitting large packets between a guest network and an external network by up to 15%, without affecting throughput. Bridge zero copy transmit is fully supported on Red Hat Enterprise Linux 7 virtual machines, but disabled by default.

For more information on zero copy transmit, see Section 5.4.1, “Bridge Zero Copy Transmit”.

APIC Virtualization (APICv)

Newer Intel processors offer hardware virtualization of the Advanced Programmable Interrupt Controller (APICv). APICv improves virtualized AMD64 and Intel 64 guest performance by allowing the guest to directly access the APIC, dramatically cutting down interrupt latencies and the number of virtual machine exits caused by the APIC. This feature is used by default in newer Intel processors and improves I/O performance.

EOI Acceleration

End-of-interrupt acceleration for high bandwidth I/O on older chipsets without virtual APIC capabilities.

Multi-queue virtio-scsi

Improved storage performance and scalability provided by multi-queue support in the virtio-scsi driver. This enables each virtual CPU to have a separate queue and interrupt to use without affecting other vCPUs.

For more information on multi-queue virtio-scsi, see Section 6.4.2, “Multi-Queue virtio-scsi”.

Paravirtualized Ticketlocks

Paravirtualized ticketlocks (pvticketlocks) improve the performance of Red Hat Enterprise Linux 7 guest virtual machines running on Red Hat Enterprise Linux 7 hosts with oversubscribed CPUs.

Paravirtualized Page Faults

Paravirtualized page faults are injected into a guest when it attempts to access a page swapped out by the host. This improves KVM guest performance when host memory is overcommitted and guest memory is swapped out.

Paravirtualized Time `vsyscall` Optimization

The **`gettimeofday`** and **`clock_gettime`** system calls execute in the user space through the **`vsyscall`** mechanism. Previously, issuing these system calls required the system to switch into kernel mode, and then back into the user space. This greatly improves performance for some applications.

Virtualization Performance Features in Red Hat Enterprise Linux

- CPU/Kernel
  - NUMA - Non-Uniform Memory Access. See Chapter 8, *NUMA* for details on NUMA.
  - CFS - Completely Fair Scheduler. A modern class-focused scheduler.
  - RCU - Read Copy Update. Better handling of shared thread data.
  - Up to 160 virtual CPUs (vCPUs).
- Memory
  - huge pages and other optimizations for memory-intensive environments. See Chapter 7, *Memory* for details.
- Networking
  - vhost-net - A fast, kernel-based VirtIO solution.
  - SR-IOV - For near-native networking performance levels.
- Block I/O
  - AIO - Support for a thread to overlap other I/O operations.
  - MSI - PCI bus device interrupt generation.

- Disk I/O throttling - Controls on guest disk I/O requests to prevent over-utilizing host resources. See Section 6.4.1, “Disk I/O Throttling” for details.

Note

For more details on virtualization support, limits, and features, refer to the *Red Hat Enterprise Linux 7 Virtualization Getting Started Guide* and the following URLs:

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

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

Chapter 2. Performance Monitoring Tools

This chapter describes tools used to monitor guest virtual machine environments.

2.1. perf kvm

You can use the **perf** command with the **kvm** option to collect and analyze guest operating system statistics from the host. The *perf* package provides the **perf** command. It is installed by running the following command:

```
# yum install perf
```

In order to use **perf kvm** in the host, you must have access to the **/proc/modules** and **/proc/kallsyms** files from the guest. Refer to the following procedure, Procedure 2.1, “Copying /proc files from guest to host” to transfer the files into the host and run reports on the files.

Procedure 2.1. Copying /proc files from guest to host

Important

If you directly copy the required files (for instance, using **scp**) you will only copy files of zero length. This procedure describes how to first save the files in the guest to a temporary location (with the **cat** command), and then copy them to the host for use by **perf kvm**.

1. **Log in to the guest and save files**

   Log in to the guest and save **/proc/modules** and **/proc/kallsyms** to a temporary location, **/tmp**:

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

2. **Copy the temporary files to the host**

   Once you have logged off from the guest, run the following example **scp** commands to copy the saved files to the host. You should substitute your host name and TCP port if they are different:

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

   You now have two files from the guest (**guest-kallsyms** and **guest-modules**) on the host, ready for use by **perf kvm**.

3. **Recording and reporting events with perf kvm**

   Using the files obtained in the previous steps, recording and reporting of events in the guest, the host, or both is now possible.

Run the following example command:

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

Note

If both **--host** and **--guest** are used in the command, output will be stored in **perf.data.kvm**. If only **--host** is used, the file will be named **perf.data.host**. Similarly, if only **--guest** is used, the file will be named **perf.data.guest**.

Pressing Ctrl-C stops recording.

4.

Reporting events

The following example command uses the file obtained by the recording process, and redirects the output into a new file, **analyze**.

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

View the contents of the **analyze** file to examine the recorded events:

```
# 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. Virtual Performance Monitoring Unit (vPMU)

The virtual performance monitoring unit (vPMU) displays statistics which indicate how a guest virtual machine is functioning.

The virtual performance monitoring unit allows users to identify sources of possible performance problems in their guest virtual machines. The vPMU is based on Intel's PMU (Performance Monitoring Units) and can only be used on Intel machines.

This feature is only supported with guest virtual machines running Red Hat Enterprise Linux 6 or Red Hat Enterprise Linux 7 and is disabled by default.

To verify if the vPMU is supported on your system, check for the **arch_perfmon** flag on the host CPU by running:

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

To enable the vPMU, specify the ***cpu mode*** in the guest XML as **host-passthrough**:

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

After the vPMU is enabled, display a virtual machine's performance statistics by running the **perf** command from the guest virtual machine.

Chapter 3. Optimizing Virtualization Performance with virt-manager

This chapter covers performance tuning options available in **virt-manager**, a desktop tool for managing guest virtual machines.

3.1. Operating System Details and Devices

3.1.1. Specifying Guest Virtual Machine Details

The **virt-manager** tool provides different profiles depending on what operating system type and version are selected for a new guest virtual machine. When creating a guest, you should provide as many details as possible; this can improve performance by enabling features available for your specific type of guest.

Refer to the following example screen capture of the **virt-manager** tool. When creating a new guest virtual machine, always specify your intended **OS type** and **Version**:

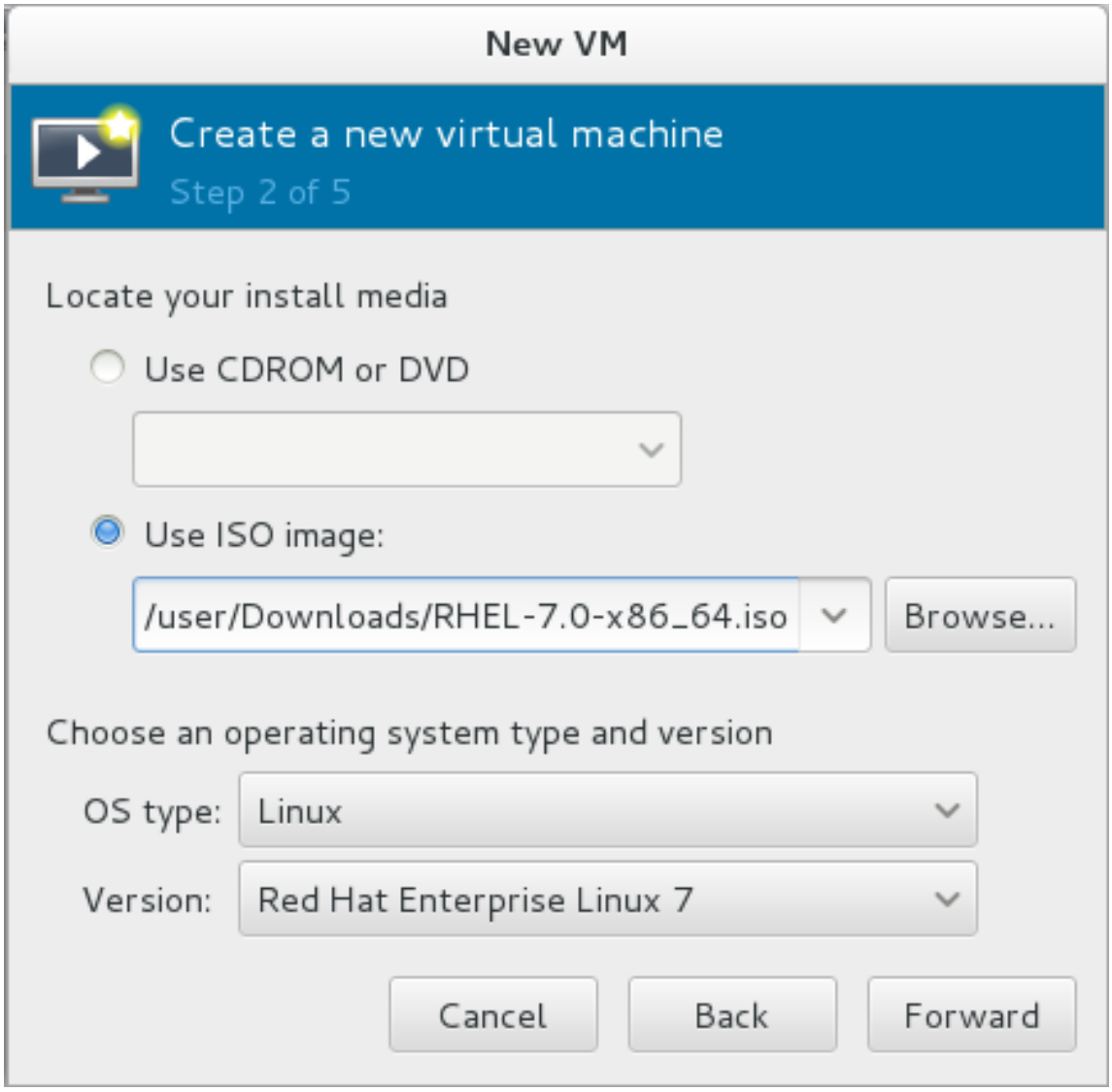


Figure 3.1. Provide the OS type and Version

3.1.2. Remove Unused Devices

Removing unused or unnecessary devices can improve performance. For instance, a guest tasked as a web server is unlikely to require audio features or an attached tablet.

Refer to the following example screen capture of the **virt-manager** tool. Click the **Remove** button to remove unnecessary devices:

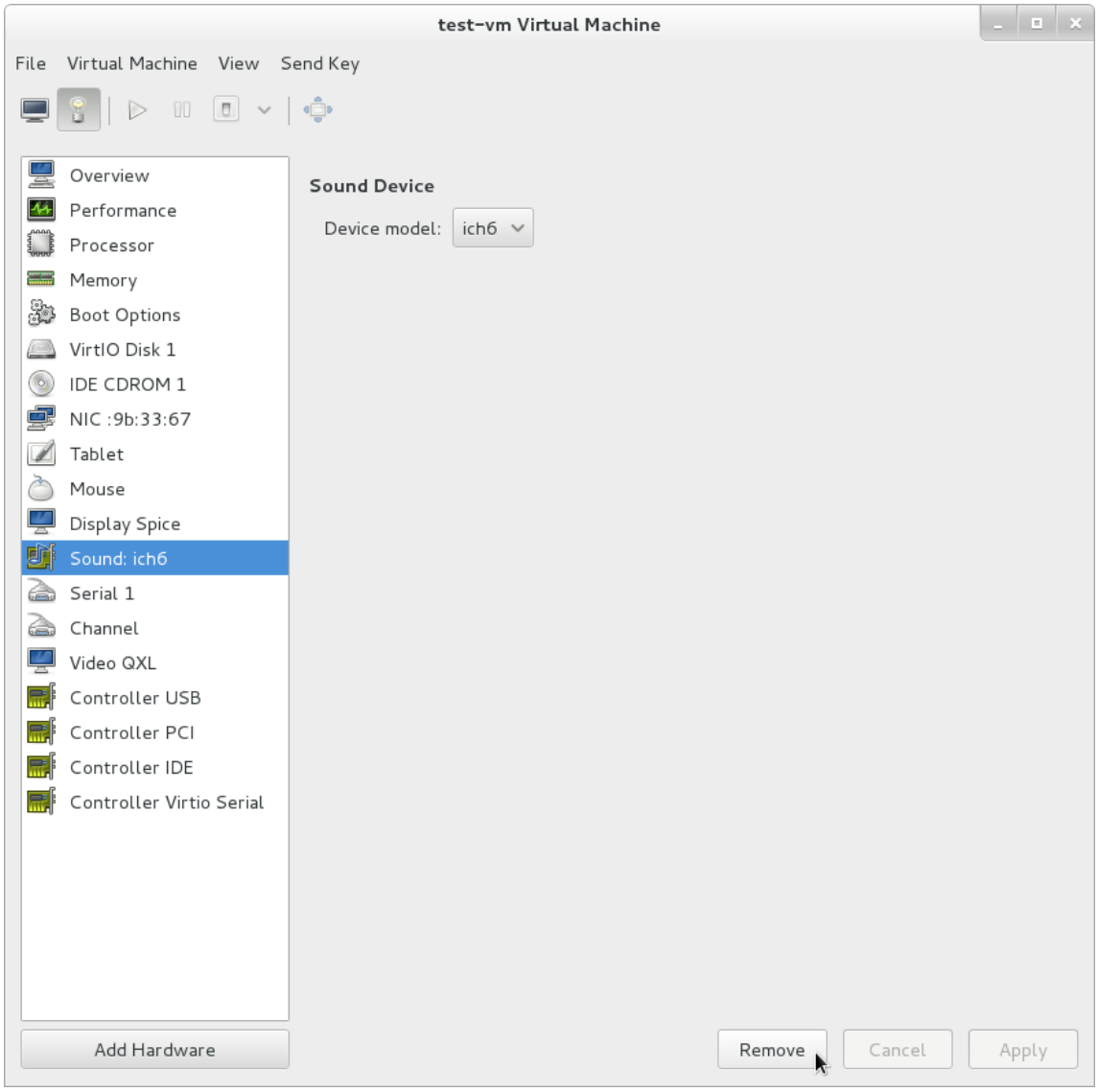


Figure 3.2. Remove unused devices

3.2. CPU Performance Options

Several CPU related options are available to your guest virtual machines. Configured correctly, these options can have a large impact on performance. The following image shows the CPU options available to your guests. The remainder of this section shows and explains the impact of these options.

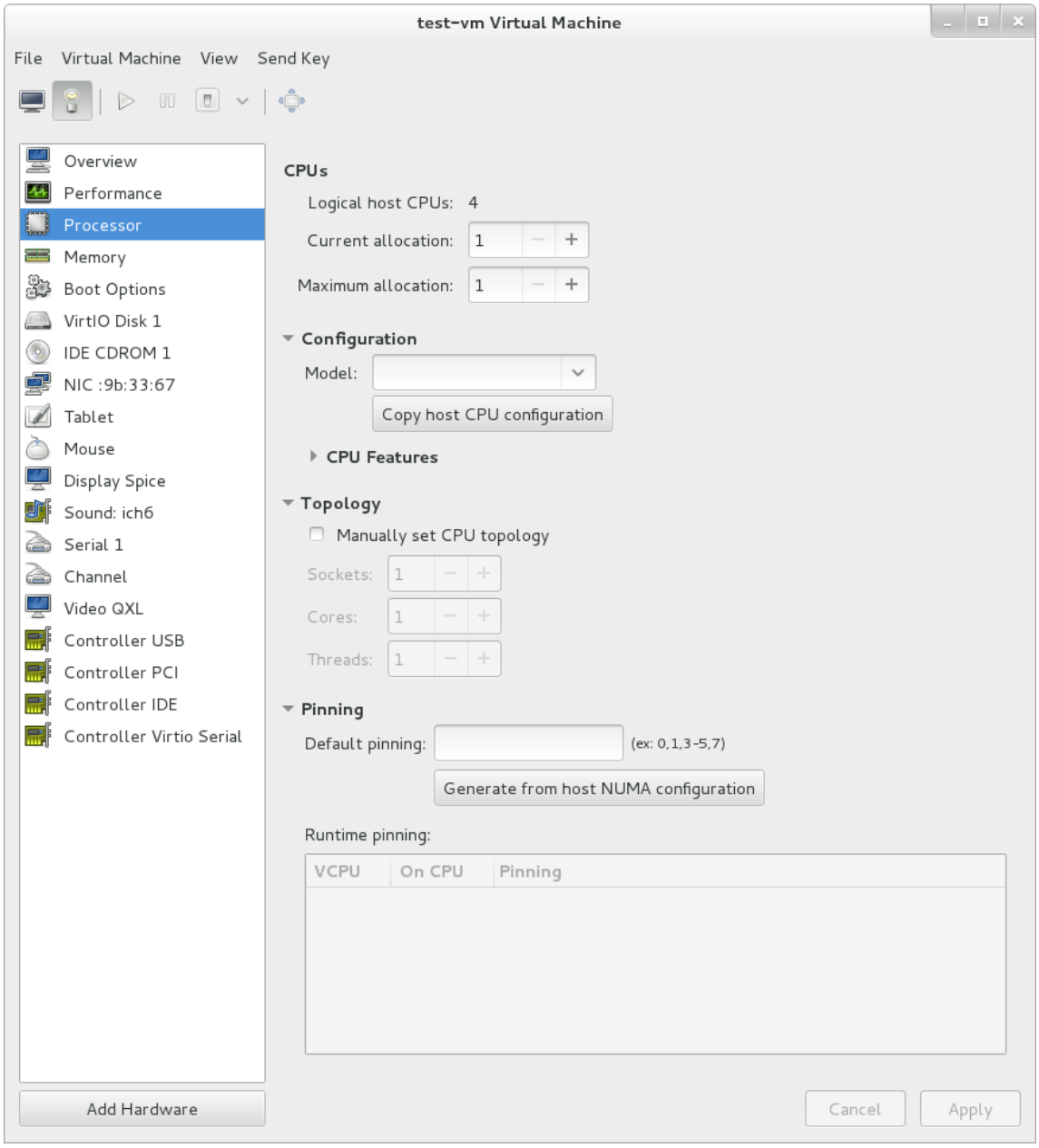


Figure 3.3. CPU Performance Options

3.2.1. Option: Available CPUs

Use this option to adjust the amount of virtual CPUs (vCPUS) available to the guest. If you allocate more than is available on the host (known as *overcommitting*), a warning is displayed, as shown in the following image:

Figure 3.4. CPU overcommit

CPUs are overcommitted when the sum of vCPUs for all guests on the system is greater than the number of host CPUs on the system. You can overcommit CPUs with one or multiple guests if the total number of vCPUs is greater than the number of host CPUs.

Important

As with memory overcommitting, CPU overcommitting can have a negative impact on performance, for example in situations with a heavy or unpredictable guest workload. Refer to the *Red Hat Enterprise Linux Virtualization Deployment and Administration Guide*, *Overcommitting with KVM* for more details on overcommitting.

3.2.2. Option: CPU Configuration

Use this option to select the CPU configuration type, based on the desired CPU model. Expand the list to see available options, or click the *Copy host CPU configuration* button to detect and apply the physical host's CPU model and configuration. Once you select a CPU configuration, its available CPU features/instructions are displayed and can be individually enabled/disabled in the *CPU Features* list. Refer to the following diagram which shows these options:

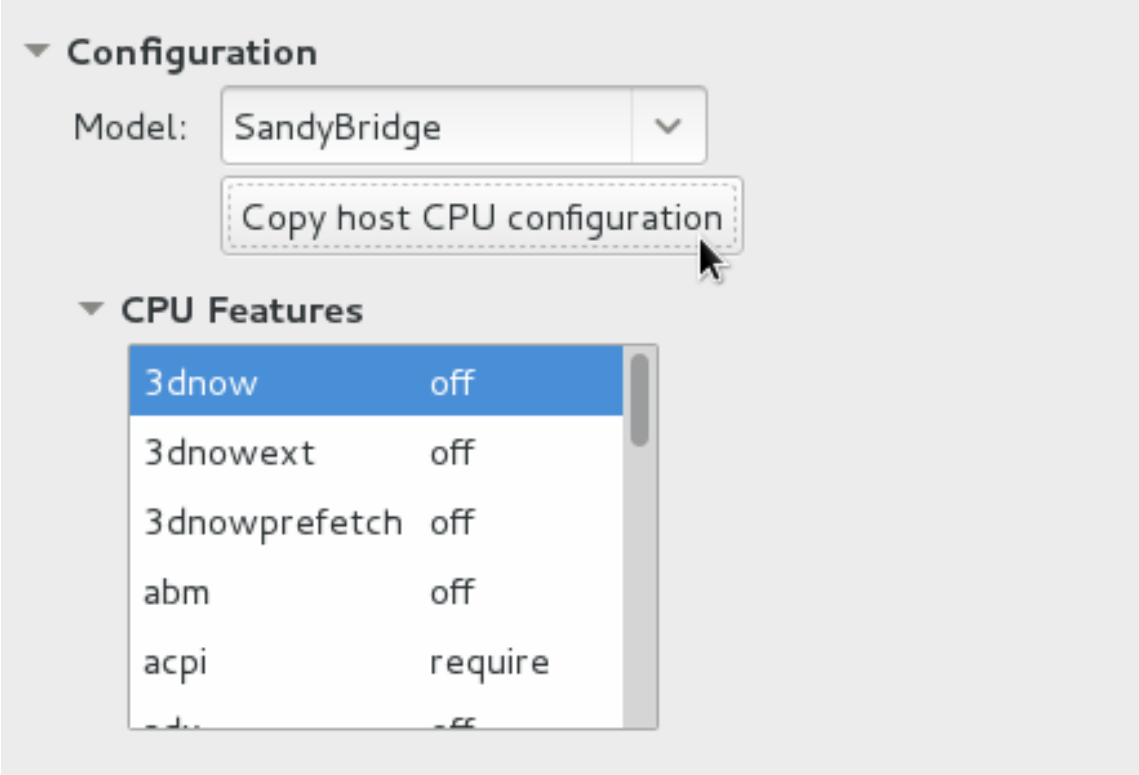


Figure 3.5. CPU Configuration Options

Note

Copying the host CPU configuration is recommended over manual configuration.

Note

Alternately, run the **`virsh capabilities`** command on your host machine to view the virtualization capabilities of your system, including CPU types and NUMA capabilities.

3.2.3. Option: CPU Topology

Use this option to apply a particular CPU topology (Sockets, Cores, Threads) to the virtual CPUs for your guest virtual machine. Refer to the following diagram which shows an example of this option:

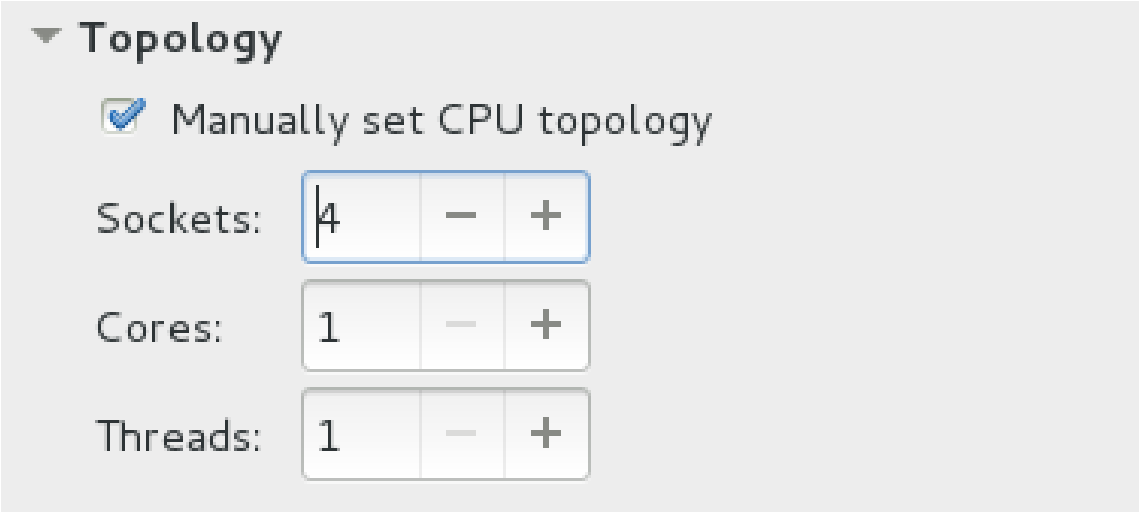


Figure 3.6. CPU Topology Options

Note

Although your environment may dictate other requirements, selecting any desired number of sockets, but with only a single core and a single thread usually gives the best performance results.

3.2.4. Option: CPU Pinning

Large performance improvements can be obtained by adhering to the system's specific NUMA topology. Use this option to automatically generate a pinning configuration that is valid for the host.

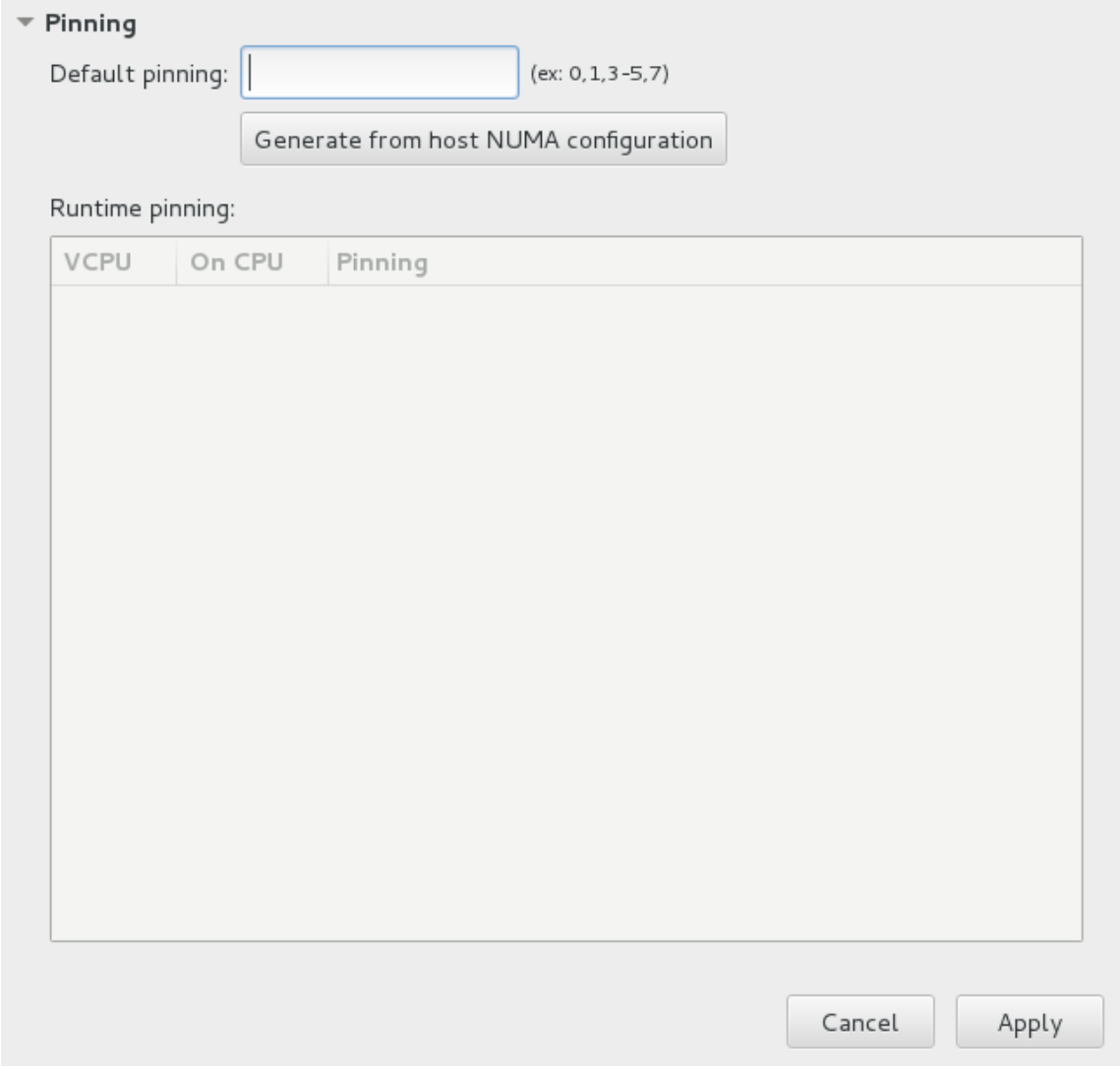


Figure 3.7. CPU Pinning

Warning

Do not use this option if the guest has more vCPUs than a single NUMA node.

Using the Pinning option will constrain the guest's vCPU threads to a single NUMA node; however, threads will be able to move around within that NUMA node. For tighter binding capabilities, use the output from the **`lscpu`** command to establish a 1:1 physical CPU to vCPU binding using **`virsh cpupin`**. Refer to Chapter 8, *NUMA* for more information on NUMA and CPU pinning.

3.3. Virtual Disk Performance Options

Several virtual disk related options are available to your guest virtual machines during installation that can impact performance. The following image shows the virtual disk options available to your guests.

The cache mode, IO mode, and IO tuning can be selected in the **`Virtual Disk`** section in **`virt-manager`**. Set these parameters in the fields under **`Performance options`**, as shown in the following image:

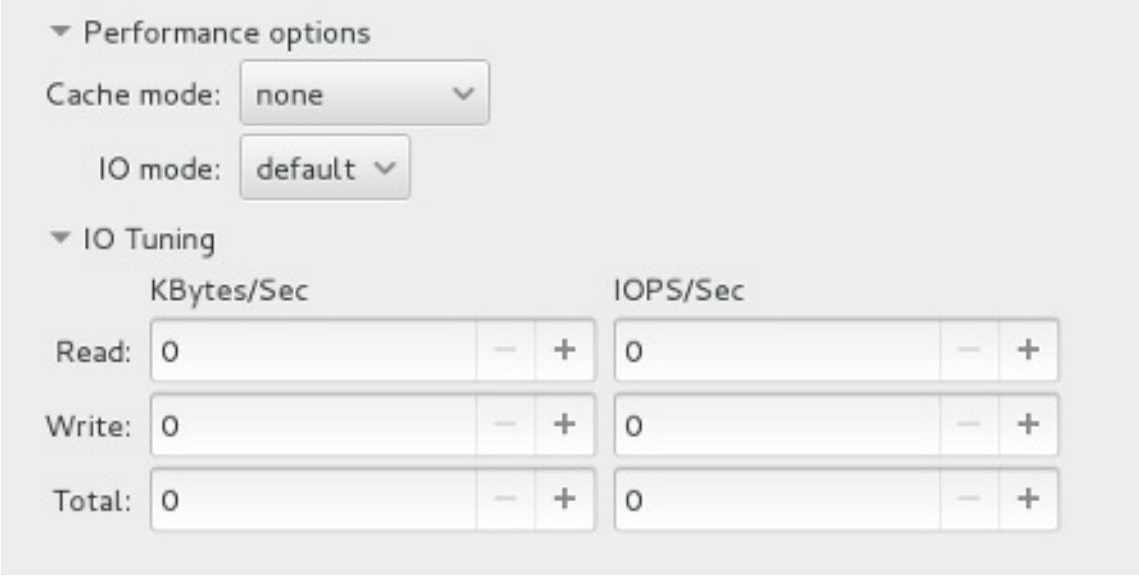


Figure 3.8. Virtual Disk Performance Options

Important

When setting the virtual disk performance options in **virt-manager**, the virtual machine must be restarted for the settings to take effect.

See Section 6.2, “Caching” and Section 6.3, “I/O Mode” for descriptions of these settings and instructions for editing these settings in the guest XML configuration.

Chapter 4. tuned and tuned-adm

This chapter covers using the **tuned** daemon for tuning system settings in virtualized environments.

tuned is a tuning profile delivery mechanism that adapts Red Hat Enterprise Linux for certain workload characteristics, such as requirements for CPU-intensive tasks, or storage/network throughput responsiveness.

The accompanying **ktune** partners with the **tuned-adm** tool to provide a number of tuning profiles that are pre-configured to enhance performance and reduce power consumption in a number of specific use cases. Edit these profiles or create new profiles to create performance solutions tailored to your environment.

The virtualization-related profiles provided as part of **tuned-adm** include:

virtual-guest

Based on the ***throughput-performance*** profile, ***virtual-guest*** also decreases the swappiness of virtual memory.

The ***virtual-guest*** profile is automatically selected when creating a Red Hat Enterprise Linux 7 guest virtual machine. It is the recommended profile for virtual machines.

This profile is available in Red Hat Enterprise Linux 6.3 and later, but must be manually selected when installing a virtual machine.

virtual-host

Based on the ***throughput-performance*** profile, ***virtual-host*** also decreases the swappiness of virtual memory and enables more aggressive writeback of dirty pages. This profile is the recommended profile for virtualization hosts, including both KVM and Red Hat Enterprise Virtualization hosts.

By default in a Red Hat Enterprise Linux 7 installation, the *tuned* package is installed and the **tuned** service is enabled.

To list all available profiles and identify the current active profile, run:

```
# 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
```

It is also possible to create custom **tuned** profiles to encapsulate a set of tuning parameters. For instructions on creating custom **tuned** profiles, refer to the **tuned.conf** man page.

To only display the currently active profile, run:

```
tuned-adm active
```

To switch to one of the available profiles, run:

```
tuned-adm profile profile_name
```

For example, to switch to the **virtual-host** profile, run:

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

Important

After setting a tuned profile in Red Hat Enterprise Linux 7.1 and above, the **tuned** service must be restarted and the system must be rebooted to apply the changes persistently. For more information, see the Red Hat Enterprise Linux 7 Performance Tuning Guide.

In some cases, it is preferable to disable **tuned** to use parameters set manually. To disable all tuning, run:

```
tuned-adm off
```

Note

Refer to the *Red Hat Enterprise Linux 7 Power Management Guide*, available from https://access.redhat.com/documentation/en-US/, for further information about **tuned**, **tuned-adm** and **ktune**.

Chapter 5. Networking

This chapter covers network optimization topics for virtualized environments.

5.1. Networking Tuning Tips

- Use multiple networks to avoid congestion on a single network. For example, have dedicated networks for management, backups, or live migration.
- Usually, matching the default MTU (1500 bytes) in all components is sufficient. If you require larger messages, increasing the MTU value can reduce fragmentation. If you change the MTU, all devices in the path should have a matching MTU value.
- Red Hat recommends not using multiple interfaces in the same network segment. However, if this is unavoidable, you can use ***`arp_filter`*** to prevent ARP Flux, an undesirable condition that can occur in both hosts and guests and is caused by the machine responding to ARP requests from more than one network interface: **`echo 1 > /proc/sys/net/ipv4/conf/all/arp_filter`** or edit **`/etc/sysctl.conf`** to make this setting persistent.

Note

Refer to the following URL for more information on ARP Flux: http://linux-ip.net/html/ether-arp.html#ether-arp-flux

5.2. Virtio and vhost_net

The following diagram demonstrates the involvement of the kernel in the Virtio and vhost_net architectures.

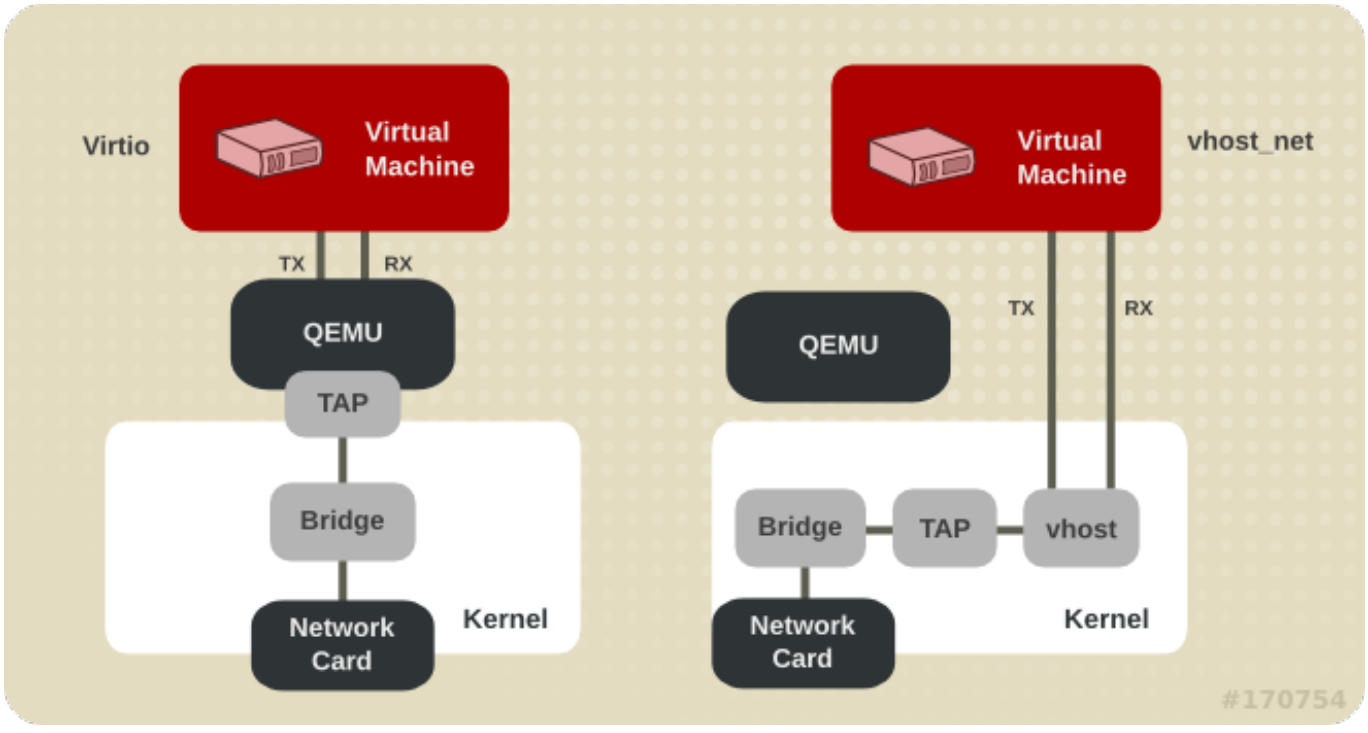


Figure 5.1. Virtio and vhost_net architectures

vhost_net moves part of the Virtio driver from the user space into the kernel. This reduces copy operations, lowers latency and CPU usage.

5.3. Device Assignment and SR-IOV

The following diagram demonstrates the involvement of the kernel in the Device Assignment and SR-IOV architectures.

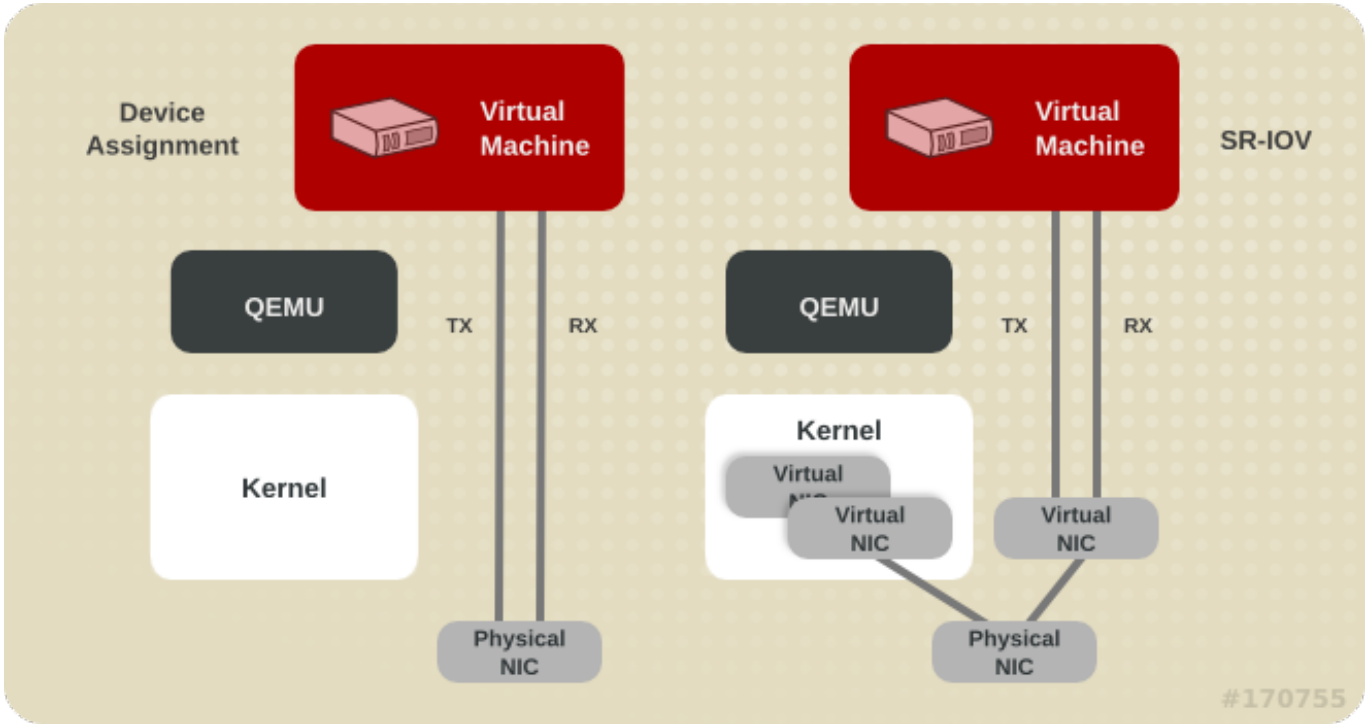


Figure 5.2. Device assignment and SR-IOV

Device assignment presents the entire device to the guest. SR-IOV needs support in drivers and hardware, including the NIC and the system board and allows multiple virtual devices to be created and passed into different guests. A vendor-specific driver is required in the guest, however, SR-IOV offers the lowest latency of any network option.

5.4. Network Tuning Techniques

This section describes techniques for tuning network performance in virtualized environments.

Important

The following features are supported on Red Hat Enterprise Linux 7 hypervisors and virtual machines, but also on virtual machines running Red Hat Enterprise Linux 6.6 and later.

5.4.1. Bridge Zero Copy Transmit

Zero copy transmit mode is effective on large packet sizes. It typically reduces the host CPU overhead by up to 15% when transmitting large packets between a guest network and an external network, without affecting throughput.

It does not affect performance for guest-to-guest, guest-to-host, or small packet workloads.

Bridge zero copy transmit is fully supported on Red Hat Enterprise Linux 7 virtual machines, but disabled by default. To enable zero copy transmit mode, set the ***experimental_zcopytx*** kernel module parameter for the vhost_net module to 1.

Note

An additional data copy is normally created during transmit as a threat mitigation technique against denial of service and information leak attacks. Enabling zero copy transmit disables this threat mitigation technique.

If performance regression is observed, or if host CPU utilization is not a concern, zero copy transmit mode can be disabled by setting ***experimental_zcopytx*** to 0.

5.4.2. Multi-Queue virtio-net

Multi-queue virtio-net provides an approach that scales the network performance as the number of vCPUs increases, by allowing them to transfer packets through more than one virtqueue pair at a time.

Today's high-end servers have more processors, and guests running on them often have an increasing number of vCPUs. In single queue virtio-net, the scale of the protocol stack in a guest is restricted, as the network performance does not scale as the number of vCPUs increases. Guests cannot transmit or retrieve packets in parallel, as virtio-net has only one TX and RX queue.

Multi-queue support removes these bottlenecks by allowing paralleled packet processing.

Multi-queue virtio-net provides the greatest performance benefit when:

- Traffic packets are relatively large.
- The guest is active on many connections at the same time, with traffic running between guests, guest to host, or guest to an external system.
- The number of queues is equal to the number of vCPUs. This is because multi-queue support optimizes RX interrupt affinity and TX queue selection in order to make a specific queue private to a specific vCPU.

Note

Multi-queue virtio-net works well for incoming traffic, but can occasionally hurt performance for outgoing traffic. Enabling multi-queue virtio-net increases the total throughput, and in parallel increases CPU consumption.

5.4.2.1. Configuring Multi-Queue virtio-net

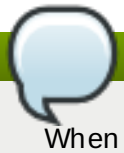
To use multi-queue virtio-net, enable support in the guest by adding the following to the guest XML configuration (where the value of *N* is from 1 to 256, as the kernel supports up to 256 queues for a multi-queue tap device):

```
<interface type='network'>
```

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

When running a virtual machine with *N* virtio-net queues in the guest, enable the multi-queue support with the following command (where the value of *M* is from 1 to *N*):

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



Note

When using multi-queue, it is recommended to change the **max_files** variable in the **/etc/libvirt/qemu.conf** file to 2048. The default limit of 1024 can be insufficient for multi-queue and cause guests to be unable to start when multi-queue is configured.

Chapter 6. Block I/O

This chapter covers optimizing I/O settings in virtualized environments.

6.1. Block I/O Tuning

The **virsh blkiotune** command allows administrators to set or display a guest virtual machine's block I/O parameters manually in the **<blkio>** element in the guest XML configuration.

To display current **<blkio>** parameters for a virtual machine:

```
# virsh blkiotune virtual_machine
```

To set a virtual machine's **<blkio>** parameters, refer to the following command and replace values according to your environment:

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

Parameters include:

weight

The I/O weight, within the range 100 to 1000.

device-weights

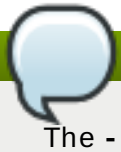
A single string listing one or more device/weight pairs, in the format of **/path/to/device,weight,/path/to/device,weight**. Each weight must be within the range 100-1000, or the value 0 to remove that device from per-device listings. Only the devices listed in the string are modified; any existing per-device weights for other devices remain unchanged.

config

Add the **--config** option for changes to take effect at next boot.

live

Add the **--live** option to apply the changes to the running virtual machine.



Note

The **--live** option requires the hypervisor to support this action. Not all hypervisors allow live changes of the maximum memory limit.

current

Add the **--current** option to apply the changes to the current virtual machine.

Note

See # **virsh help blkiotune** for more information on using the **virsh blkiotune** command.

6.2. Caching

Caching options can be configured with **virt-manager** during guest installation, or on an existing guest virtual machine by editing the guest XML configuration.

Table 6.1. Caching options

Caching Option	Description
Cache=none	I/O from the guest is not cached on the host, but may be kept in a writeback disk cache. Use this option for guests with large I/O requirements. This option is generally the best choice, and is the only option to support migration.
Cache=writethrough	I/O from the guest is cached on the host but written through to the physical medium. This mode is slower and prone to scaling problems. Best used for small number of guests with lower I/O requirements. Suggested for guests that do not support a writeback cache (such as Red Hat Enterprise Linux 5.5 and earlier), where migration is not needed.
Cache=writeback	I/O from the guest is cached on the host.
Cache=directsync	Similar to ***writethrough***, but I/O from the guest bypasses the host page cache.
Cache=unsafe	The host may cache all disk I/O, and sync requests from guest are ignored.
Cache=default	If no cache mode is specified, the system's default settings are chosen.

In **virt-manager**, the caching mode can be specified under **Virtual Disk**. For information on using **virt-manager** to change the cache mode, see Section 3.3, “Virtual Disk Performance Options”

To configure the cache mode in the guest XML, edit the **cache** setting inside the **driver** tag to specify a caching option. For example, to set the cache as ***writeback***:

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

6.3. I/O Mode

I/O mode options can be configured with **virt-manager** during guest installation, or on an existing guest virtual machine by editing the guest XML configuration.

Table 6.2. IO mode options

IO Mode Option	Description
IO=native	The default for Red Hat Enterprise Virtualization environments. This mode refers to kernel asynchronous I/O with direct I/O options.
IO=threads	The default are host user-mode based threads.
IO=default	The default in Red Hat Enterprise Linux 7 is threads mode.

In **virt-manager**, the I/O mode can be specified under **Virtual Disk**. For information on using **virt-manager** to change the I/O mode, see Section 3.3, “Virtual Disk Performance Options”

To configure the I/O mode in the guest XML, edit the **io** setting inside the **driver** tag, specifying ***native***, ***threads***, or ***default***. For example, to set the I/O mode to ***threads***:

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

6.4. Block I/O Tuning Techniques

This section describes more techniques for tuning block I/O performance in virtualized environments.

6.4.1. Disk I/O Throttling

When several virtual machines are running simultaneously, they can interfere with system performance by using excessive disk I/O. Disk I/O throttling in KVM provides the ability to set a limit on disk I/O requests sent from virtual machines to the host machine. This can prevent a virtual machine from over-utilizing shared resources and impacting the performance of other virtual machines.

Disk I/O throttling can be useful in various situations, for example when guest virtual machines belonging to different customers are running on the same host, or when quality of service guarantees are given for different guests. Disk I/O throttling can also be used to simulate slower disks.

I/O throttling can be applied independently to each block device attached to a guest and supports limits on throughput and I/O operations. Use the **virsh blkdeviotune** command to set I/O limits for a virtual machine. Refer to the following example:

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

Device specifies a unique target name (**<target dev='name'/>**) or source file (**<source file='name'/>**) for one of the disk devices attached to the virtual machine. Use the **virsh domblklist** command for a list of disk device names.

Optional parameters include:

total-bytes-sec

The total throughput limit in bytes per second.

read-bytes-sec

The read throughput limit in bytes per second.

write-bytes-sec

The write throughput limit in bytes per second.

total-iops-sec

The total I/O operations limit per second.

read-iops-sec

The read I/O operations limit per second.

write-iops-sec

The write I/O operations limit per second.

For example, to throttle **vda** on **virtual_machine** to 1000 I/O operations per second and 50 MB per second throughput, run this command:

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

6.4.2. Multi-Queue virtio-scsi

Multi-queue virtio-scsi provides improved storage performance and scalability in the virtio-scsi driver. It enables each virtual CPU to have a separate queue and interrupt to use without affecting other vCPUs.

6.4.2.1. Configuring Multi-Queue virtio-scsi

Multi-queue virtio-scsi is disabled by default on Red Hat Enterprise Linux 7.

To enable multi-queue virtio-scsi support in the guest, add the following to the guest XML configuration, where *N* is the total number of vCPU queues:

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

Chapter 7. Memory

This chapter covers memory optimization options for virtualized environments.

7.1. Memory Tuning Tips

To optimize memory performance in a virtualized environment, consider the following:

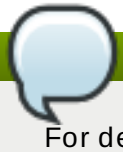
- Do not allocate more resources to guest than it will use.
- If possible, assign a guest to a single NUMA node, providing that resources are sufficient on that NUMA node. For more information on using NUMA, see Chapter 8, *NUMA*.

7.2. Memory Tuning on Virtual Machines

7.2.1. Memory Monitoring Tools

Memory usage can be monitored in virtual machines using tools used in bare metal environments. Tools useful for monitoring memory usage and diagnosing memory-related problems include:

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



Note

For details on using these performance tools, refer to the *Red Hat Enterprise Linux 7 Performance Tuning Guide* and the man pages for these commands.

7.2.2. Memory Tuning with virsh

The optional **`<memtune>`** element in the guest XML configuration allows administrators to configure guest virtual machine memory settings manually. If **`<memtune>`** is omitted, default memory settings apply.

Display or set memory parameters in the **`<memtune>`** element in a virtual machine with the **`virsh memtune`** command, replacing values according to your environment:

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

Optional parameters include:

`hard_limit`

The maximum memory the virtual machine can use, in kibibytes (blocks of 1024 bytes).

Warning

Setting this limit too low can result in the virtual machine being killed by the kernel.

soft_limit

The memory limit to enforce during memory contention, in kibibytes (blocks of 1024 bytes).

swap_hard_limit

The maximum memory plus swap the virtual machine can use, in kibibytes (blocks of 1024 bytes). The ***swap_hard_limit*** value must be more than the ***hard_limit*** value.

min_guarantee

The guaranteed minimum memory allocation for the virtual machine, in kibibytes (blocks of 1024 bytes).

Note

See `# virsh help memtune` for more information on using the `virsh memtune` command.

The optional **<memoryBacking>** element may contain several elements that influence how virtual memory pages are backed by host pages.

Setting ***locked*** prevents the host from swapping out memory pages belonging to the guest. Add the following to the guest XML to lock the virtual memory pages in the host's memory:

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

Important

When setting ***locked***, a ***hard_limit*** must be set in the **<memtune>** element to the maximum memory configured for the guest, plus any memory consumed by the process itself.

Setting ***nosharepages*** prevents the host from merging the same memory used among guests. To instruct the hypervisor to disable share pages for a guest, add the following to the guest's XML:

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

7.2.3. Huge Pages and Transparent Huge Pages (THP)

x86 CPUs usually address memory in 4kB pages, but they are capable of using larger 2 MB or 1 GB pages known as *huge pages*. KVM guests can be deployed with huge page memory support in order to improve performance by increasing CPU cache hits against the Transaction Lookaside Buffer (TLB).

A kernel feature enabled by default in Red Hat Enterprise Linux 7, huge pages can significantly increase performance, particularly for large memory and memory-intensive workloads. Red Hat Enterprise Linux 7 is able to more effectively manage large amounts of memory by increasing the page size through the use of huge pages.

Red Hat Enterprise Linux 7 systems support 2 MB and 1 GB huge pages, which can be allocated at boot or at runtime. See Section 7.2.3.3, “Enabling 1 GB huge pages for guests at boot or runtime” for instructions on enabling multiple huge page sizes.

7.2.3.1. Configuring Transparent Huge Pages

Transparent huge pages (THP) automatically optimize system settings for performance. By allowing all free memory to be used as cache, performance is increased. As KSM can reduce the occurance of transparent huge pages, you may want to disable it before enabling THP. If you want to disable KSM, refer to Section 7.3.4, “Deactivating KSM”.

Transparent huge pages are enabled by default. To check the current status, run:

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

To enable transparent huge pages to be used by default, run:

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

This will set **/sys/kernel/mm/transparent_hugepage/enabled** to ***always***.

To disable transparent huge pages:

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

Transparent Huge Page support does not prevent the use of static huge pages. However, when static huge pages are not used, KVM will use transparent huge pages instead of the regular 4kB page size.

7.2.3.2. Configuring Static Huge Pages

In some cases, greater control of huge pages is preferable. To use static huge pages on guests, add the following to the guest XML configuration using **virsh edit**:

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

This instructs the host to allocate memory to the guest using huge pages, instead of using the default page size.

In Red Hat Enterprise Linux 7.1 and above, huge pages from the host can be allocated to guest NUMA nodes. Specify the huge page size, units, and the guest NUMA nodeset in the ***<memoryBacking>*** element in the guest XML. For more information and an example of configuration, see Section 8.3.10, “Assigning Host Huge Pages to Multiple Guest NUMA Nodes”.

View the current huge pages value by running the following command:

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

Procedure 7.1. Setting huge pages

The following example procedure shows the commands to set huge pages.

1. View the current huge pages value:

```
# 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 pages are set in increments of 2MB. To set the number of huge pages to 25000, use the following command:

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

Note

To make the setting persistent, add the following lines to the **/etc/sysctl.conf** file on the guest machine, with X being the intended number of huge pages:

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

Afterwards, add **transparent_hugepage=never** to the kernel boot parameters by appending it to the end of the **/kernel** line in the **/etc/grub2.cfg** file on the guest.

3. Mount the huge pages:

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

4. Restart **libvirtd**, then restart the virtual machine with the following commands:

```
# systemctl start libvirtd
```

```
# virsh start virtual_machine
```

5. Verify the changes in **/proc/meminfo**:

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

Huge pages can benefit not only the host but also guests, however, their total huge pages value must be less than what is available in the host.

7.2.3.3. Enabling 1 GB huge pages for guests at boot or runtime

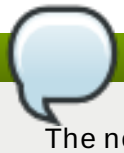
Red Hat Enterprise Linux 7 systems support 2 MB and 1 GB huge pages, which can be allocated at boot or at runtime.

Procedure 7.2. Allocating 1 GB huge pages at boot time

1. To allocate different sizes of huge pages at boot time, use the following command, specifying the number of huge pages. This example allocates 4 1 GB huge pages and 1024 2 MB huge pages:

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

   Change this command line to specify a different number of huge pages to be allocated at boot.

   Note

   The next two steps must also be completed the first time you allocate 1 GB huge pages at boot time.

2. Mount the 2 MB and 1 GB huge pages on the host:

   ```
   # 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. Restart libvirtd to enable the use of 1 GB huge pages on guests:

   ```
   # systemctl restart libvirtd
   ```

Procedure 7.3. Allocating 1 GB huge pages at runtime

1 GB huge pages can also be allocated at runtime. Runtime allocation allows the system administrator to choose which NUMA node to allocate those pages from. However, runtime page allocation can be more prone to allocation failure than boot time allocation due to memory fragmentation.

1. To allocate different sizes of huge pages at runtime, use the following command, replacing values for the number of huge pages, the NUMA node to allocate them from, and the huge page size:

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

This example command allocates 4 1 GB huge pages from **node1** and 1024 2MB huge pages from **node3**.

These huge page settings can be changed at any time with the above command, depending on the amount of free memory on the host system.

Note

The next two steps must also be completed the first time you allocate 1 GB huge pages at runtime.

2. Mount the 2 MB and 1 GB huge pages on the host:

```
# 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. Restart libvirtd to enable the use of 1 GB huge pages on guests:

```
# systemctl restart libvirtd
```

Note

See Section 8.3.10, “Assigning Host Huge Pages to Multiple Guest NUMA Nodes” to configure NUMA-node specific huge pages.

7.3. Kernel Same-page Merging (KSM)

Kernel Same-page Merging (KSM), used by the KVM hypervisor, allows KVM guests to share identical memory pages. These shared pages are usually common libraries or other identical, high-use data. KSM allows for greater guest density of identical or similar guest operating systems by avoiding memory duplication.

The concept of shared memory is common in modern operating systems. For example, when a program is first started, it shares all of its memory with the parent program. When either the child or parent program tries to modify this memory, the kernel allocates a new memory region, copies the original contents and allows the program to modify this new region. This is known as copy on write.

KSM is a Linux feature which uses this concept in reverse. KSM enables the kernel to examine two or more already running programs and compare their memory. If any memory regions or pages are identical, KSM reduces multiple identical memory pages to a single page. This page is then marked copy on write. If the contents of the page is modified by a guest virtual machine, a new page is created for that guest.

This is useful for virtualization with KVM. When a guest virtual machine is started, it only inherits the memory from the host **qemu-kvm** process. Once the guest is running, the contents of the guest operating system image can be shared when guests are running the same operating system or applications. KSM allows KVM to request that these identical guest memory regions be shared.

KSM provides enhanced memory speed and utilization. With KSM, common process data is stored in

cache or in main memory. This reduces cache misses for the KVM guests, which can improve performance for some applications and operating systems. Secondly, sharing memory reduces the overall memory usage of guests, which allows for higher densities and greater utilization of resources.

Note

In Red Hat Enterprise Linux 7, KSM is NUMA aware. This allows it to take NUMA locality into account while coalescing pages, thus preventing performance drops related to pages being moved to a remote node. Red Hat recommends avoiding cross-node memory merging when KSM is in use. If KSM is in use, change the **/sys/kernel/mm/ksm/merge_across_nodes** tunable to ***0*** to avoid merging pages across NUMA nodes. This can be done with the **virsh node-memory-tune --shm-merge-across-nodes 0** command. Kernel memory accounting statistics can eventually contradict each other after large amounts of cross-node merging. As such, numad can become confused after the KSM daemon merges large amounts of memory. If your system has a large amount of free memory, you may achieve higher performance by turning off and disabling the KSM daemon. Refer to Chapter 8, *NUMA*" for more information on NUMA.

Important

Ensure the swap size is sufficient for the committed RAM even with KSM. KSM reduces the RAM usage of identical or similar guests. Overcommitting guests with KSM without sufficient swap space may be possible, but is not recommended because guest virtual machine memory use can result in pages becoming unshared.

Red Hat Enterprise Linux uses two separate methods for controlling KSM:

- The **ksm** service starts and stops the KSM kernel thread.
- The **ksmtuned** service controls and tunes the **ksm** service, dynamically managing same-page merging. The **ksmtuned** service starts **ksm** and stops the **ksm** service if memory sharing is not necessary. The **ksmtuned** service must be instructed with the ***retune*** parameter to run when new guests are created or destroyed.

Both of these services are controlled with the standard service management tools.

7.3.1. The KSM Service

The **ksm** service is included in the *qemu-kvm* package. KSM is off by default on Red Hat Enterprise Linux 7.

When the **ksm** service is not started, KSM shares only 2000 pages. This default is low and provides limited memory saving benefits.

When the **ksm** service is started, KSM will share up to half of the host system's main memory. Start the **ksm** service to enable KSM to share more memory.

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

The **ksm** service can be added to the default startup sequence. Make the **ksm** service persistent with the systemctl command.

```
# systemctl enable ksm
```

7.3.2. The KSM Tuning Service

The **ksmtuned** service does not have any options. The **ksmtuned** service loops and adjusts **ksm**. The **ksmtuned** service is notified by libvirt when a guest virtual machine is created or destroyed.

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

The **ksmtuned** service can be tuned with the ***retune*** parameter. The ***retune*** parameter instructs **ksmtuned** to run tuning functions manually.

The **/etc/ksmtuned.conf** file is the configuration file for the **ksmtuned** service. The file output below is the default **ksmtuned.conf** file:

```
# 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
```

Within the **/etc/ksmtuned.conf** file, **npages** sets how many pages ksm will scan before **ksmd**

goes to sleep. It will be set at **/sys/kernel/mm/ksm/pages_to_scan**.

`thres` sets the activation threshold, in kbytes. A KSM cycle is triggered when the `thres` value added to the sum of all **qemu-kvm** processes RSZ exceeds total system memory. This parameter is the equivalent in kbytes of the percentage defined in parameter `KSM_THRES_COEF`.

7.3.3. KSM Variables and Monitoring

KSM stores monitoring data in the **/sys/kernel/mm/ksm/** directory. Files in this directory are updated by the kernel and are an accurate record of KSM usage and statistics.

The variables in the list below are also configurable variables in the **/etc/ksmtuned.conf** file as noted below.

Files in /sys/kernel/mm/ksm/:

full_scans

Full scans run.

merge_across_nodes

Whether pages from different NUMA nodes can be merged.

pages_shared

Total pages shared.

pages_sharing

Pages presently shared.

pages_to_scan

Pages not scanned.

pages_unshared

Pages no longer shared.

pages_volatile

Number of volatile pages.

run

Whether the KSM process is running.

sleep_millisecs

Sleep milliseconds.

These variables can be manually tuned using the **virsh node-memory-tune** command. For example, the following specifies the number of pages to scan before the shared memory service goes to sleep:

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

KSM tuning activity is stored in the **/var/log/ksmtuned** log file if the ***DEBUG=1*** line is added to the **/etc/ksmtuned.conf** file. The log file location can be changed with the ***LOGFILE*** parameter. Changing the log file location is not advised and may require special configuration of SELinux settings.

7.3.4. Deactivating KSM

KSM has a performance overhead which may be too large for certain environments or host systems. KSM may also introduce side channels that could be potentially used to leak information across guests. In the case that is a concern, KSM can be disabled on per-guest basis.

KSM can be deactivated by stopping the **ksmtuned** and the **ksm** service. Stopping the services deactivates KSM, but this action does not persist after restarting. It should also be noted that when KSM is disabled, any memory pages that were shared prior to deactivating KSM are still shared. To deactivate KSM, run the following in a terminal as root:

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

Persistently deactivate KSM with the **systemctl** command. To disable the services, run the following commands:

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

To delete all of the PageKSM in the system, run the following command in a terminal as root:

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

After this runs, the **khugepaged** daemon can rebuild transparent hugepages on the KVM guest physical memory, whereas running # **echo 0 >/sys/kernel/mm/ksm/run** stops KSM, but does not unshare all the previously created KSM pages (this is the same as the # **systemctl stop ksmtuned** command).

Chapter 8. NUMA

Historically, all memory on x86 systems is equally accessible by all CPUs. Known as Uniform Memory Access (UMA), access times are the same no matter which CPU performs the operation.

This behavior is no longer the case with recent x86 processors. In Non-Uniform Memory Access (NUMA), system memory is divided across NUMA *nodes*, which correspond to sockets or to a particular set of CPUs that have identical access latency to the local subset of system memory.

This chapter describes memory allocation and NUMA tuning configurations in virtualized environments.

8.1. NUMA Memory Allocation Policies

Three policy types define how memory is allocated from the nodes in a system:

Strict

The default operation is for allocation to fall back to other nodes if the memory cannot be allocated on the target node. Strict policy means that the allocation will fail if the memory cannot be allocated on the target node.

Interleave

Memory pages are allocated across nodes specified by a nodemask, but are allocated in a round-robin fashion.

Preferred

Memory is allocated from a single preferred memory node. If sufficient memory is not available, memory can be allocated from other nodes.

XML configuration enables the desired policy:

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

8.2. Automatic NUMA Balancing

Automatic NUMA balancing improves the performance of applications running on NUMA hardware systems. It is enabled by default on Red Hat Enterprise Linux 7 systems.

An application will generally perform best when the threads of its processes are accessing memory on the same NUMA node as the threads are scheduled. Automatic NUMA balancing moves tasks (which can be threads or processes) closer to the memory they are accessing. It also moves application data to memory closer to the tasks that reference it. This is all done automatically by the kernel when automatic NUMA balancing is active.

Automatic NUMA balancing uses a number of algorithms and data structures, which are only active and allocated if automatic NUMA balancing is active on the system:

- Periodic NUMA unmapping of process memory
- NUMA hinting fault

- Migrate-on-Fault (MoF) - moves memory to where the program using it runs
- task_numa_placement - moves running programs closer to their memory

8.2.1. Configuring Automatic NUMA Balancing

Automatic NUMA balancing is enabled by default in Red Hat Enterprise Linux 7, and will automatically activate when booted on hardware with NUMA properties.

Automatic NUMA balancing is enabled when both of the following conditions are met:

- **# numactl --hardware** shows multiple nodes, and
- **# cat /sys/kernel/debug/sched_features** shows **NUMA** in the flags.

Manual NUMA tuning of applications will override automatic NUMA balancing, disabling periodic unmapping of memory, NUMA faults, migration, and automatic NUMA placement of those applications.

In some cases, system-wide manual NUMA tuning is preferred.

To disable automatic NUMA balancing, use the following command:

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

To enable automatic NUMA balancing, use the following command:

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

8.3. libvirt NUMA Tuning

Generally, best performance on NUMA systems is achieved by limiting guest size to the amount of resources on a single NUMA node. Avoid unnecessarily splitting resources across NUMA nodes.

Use the **numastat** tool to view per-NUMA-node memory statistics for processes and the operating system.

In the following example, the **numastat** tool shows four virtual machines with suboptimal memory alignment across NUMA nodes:

```
# 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
```

Run **numad** to align the guests' CPUs and memory resources automatically.

Then run **numastat -c qemu-kvm** again to view the results of running **numad**. The following output shows that resources have been aligned:

```
# 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
```

Note

Running **numastat** with **-c** provides compact output; adding the **-m** option adds system-wide memory information on a per-node basis to the output. Refer to the **numastat** man page for more information.

8.3.1. Monitoring Memory per host NUMA Node

You can use the **nodestats.py** script to report the total memory and free memory for each NUMA node on a host. This script also reports how much memory is strictly bound to certain host nodes for each running domain. For example:

```
# ./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
```

This example shows four host NUMA nodes, each containing approximately 4GB of RAM in total (***MemTotal***). Nearly all memory is consumed on each domain (***MemFree***). There are four domains (virtual machines) running: domain *'rhel7-0'* has 1.5GB memory which is not pinned onto any specific host NUMA node. Domain *'rhel7-2'* however, has 4GB memory and 4 NUMA nodes which are pinned 1:1 to host nodes.

To print host NUMA node statistics, create a **nodestats.py** script for your environment. An example script can be found the *libvirt-python* package files in **./examples/nodestats.py**.

8.3.2. NUMA vCPU Pinning

vCPU pinning provides similar advantages to task pinning on bare metal systems. Since vCPUs run as user-space tasks on the host operating system, pinning increases cache efficiency. One example of this is an environment where all vCPU threads are running on the same physical socket, therefore sharing a L3 cache domain.

Note

In Red Hat Enterprise Linux versions 7.0 to 7.2, it is only possible to pin active vCPUs. However, with Red Hat Enterprise Linux 7.3, pinning inactive vCPUs is available as well.

Combining vCPU pinning with **numatune** can avoid NUMA misses. The performance impacts of NUMA misses are significant, generally starting at a 10% performance hit or higher. vCPU pinning and **numatune** should be configured together.

If the virtual machine is performing storage or network I/O tasks, it can be beneficial to pin all vCPUs and memory to the same physical socket that is physically connected to the I/O adapter.

Note

The **lstopo** tool can be used to visualize NUMA topology. It can also help verify that vCPUs are binding to cores on the same physical socket. Refer to the following Knowledgebase article for more information on **lstopo**: https://access.redhat.com/site/solutions/62879.

Important

Pinning causes increased complexity where there are many more vCPUs than physical cores.

The following example XML configuration has a domain process pinned to physical CPUs 0-7. The vCPU thread is pinned to its own cpuset. For example, vCPU0 is pinned to physical CPU 0, vCPU1 is pinned to physical CPU 1, and so on:

```
<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>
```

There is a direct relationship between the vcpu and vcpupin tags. If a vcpupin option is not specified, the value will be automatically determined and inherited from the parent vcpu tag option. The following configuration shows **<vcpupin>** for **vcpu 5** missing. Hence, **vCPU5** would be pinned to physical CPUs 0-7, as specified in the parent tag **<vcpu>**:

```
<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>
```

Important

<vcpupin>, **<numatune>**, and **<emulatorpin>** should be configured together to achieve optimal, deterministic performance. For more information on the **<numatune>** tag, see Section 8.3.3, “Domain Processes”. For more information on the **<emulatorpin>** tag, see Section 8.3.5, “Using emulatorpin”.

8.3.3. Domain Processes

As provided in Red Hat Enterprise Linux, libvirt uses libnuma to set memory binding policies for domain processes. The nodeset for these policies can be configured either as *static* (specified in the domain XML) or *auto* (configured by querying numad). Refer to the following XML configuration for examples on how to configure these inside the **<numatune>** tag:

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

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

libvirt uses **sched_setaffinity(2)** to set CPU binding policies for domain processes. The cpuset option can either be *static* (specified in the domain XML) or *auto* (configured by querying numad). Refer to the following XML configuration for examples on how to configure these inside the **<vcpu>** tag:

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

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

There are implicit inheritance rules between the placement mode you use for **<vcpu>** and **<numatune>**:

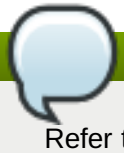
- The placement mode for **<numatune>** defaults to the same placement mode of **<vcpu>**, or to *static* if a **<nodeset>** is specified.
- Similarly, the placement mode for **<vcpu>** defaults to the same placement mode of **<numatune>**, or to *static* if **<cpuset>** is specified.

This means that CPU tuning and memory tuning for domain processes can be specified and defined separately, but they can also be configured to be dependent on the other's placement mode.

It is also possible to configure your system with numad to boot a selected number of vCPUs without pinning all vCPUs at startup.

For example, to enable only 8 vCPUs at boot on a system with 32 vCPUs, configure the XML similar to the following:

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



Note

Refer to the following URLs for more information on vcpu and numatune: http://libvirt.org/formatdomain.html#elementsCPUAllocation and http://libvirt.org/formatdomain.html#elementsNUMATuning

8.3.4. Domain vCPU Threads

In addition to tuning domain processes, libvirt also permits the setting of the pinning policy for each vcpu thread in the XML configuration. Set the pinning policy for each vcpu thread inside the **<cputune>** tags:

```
<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>
```

In this tag, libvirt uses either cgroup or **sched_setaffinity(2)** to pin the vcpu thread to the specified cpuset.

Note

For more details on **<cputune>**, refer to the following URL:
http://libvirt.org/formatdomain.html#elementsCPUTuning

In addition, if you need to set up a virtual machines with more vCPU than a single NUMA node, configure the host so that the guest detects a NUMA topology on the host. This allows for 1:1 mapping of CPUs, memory, and NUMA nodes. For example, this can be applied with a guest with 4 vCPUs and 6 GB memory, and a host with the following NUMA settings:

```
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
```

In this scenario, use the following Domain XML setting:

```
<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"/>
      <memnode cellid="0" mode="strict" nodeset="1"/>
      <memnode cellid="1" mode="strict" nodeset="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>
```

8.3.5. Using emulatorpin

Another way of tuning the domain process pinning policy is to use the **<emulatorpin>** tag inside of **<cputune>**.

The **<emulatorpin>** tag specifies which host physical CPUs the *emulator* (a subset of a domain, not including vCPUs) will be pinned to. The **<emulatorpin>** tag provides a method of setting a precise affinity to emulator thread processes. As a result, vhost threads run on the same subset of physical CPUs and memory, and therefore benefit from cache locality. For example:

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

Note

In Red Hat Enterprise Linux 7, automatic NUMA balancing is enabled by default. Automatic NUMA balancing reduces the need for manually tuning **<emulatorpin>**, since the vhost-net emulator thread follows the vCPU tasks more reliably. For more information about automatic NUMA balancing, see Section 8.2, “Automatic NUMA Balancing”.

8.3.6. Tuning vCPU Pinning with virsh

Important

These are example commands only. You will need to substitute values according to your environment.

The following example **virsh** command will pin the vcpu thread *rhel7* which has an ID of 1 to the physical CPU 2:

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

You can also obtain the current vcpu pinning configuration with the **virsh** command. For example:

```
% virsh vcpupin rhel7
```

8.3.7. Tuning Domain Process CPU Pinning with virsh

Important

These are example commands only. You will need to substitute values according to your environment.

The **emulatorpin** option applies CPU affinity settings to threads that are associated with each domain process. For complete pinning, you must use both **virsh vcpupin** (as shown previously) and **virsh emulatorpin** for each guest. For example:

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

8.3.8. Tuning Domain Process Memory Policy with virsh

Domain process memory can be dynamically tuned. Refer to the following example command:

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

More examples of these commands can be found in the **virsh** man page.

8.3.9. Guest NUMA Topology

Guest NUMA topology can be specified using the **<numa>** tag inside the **<cpu>** tag in the guest virtual machine's XML. Refer to the following example, and replace values accordingly:

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

Each **<cell>** element specifies a NUMA cell or a NUMA node. **cpus** specifies the CPU or range of CPUs that are part of the node, and **memory** specifies the node memory in kibibytes (blocks of 1024 bytes). Each cell or node is assigned a **cellid** or **nodeid** in increasing order starting from 0.

Important

When modifying the NUMA topology of a guest virtual machine with a configured topology of CPU sockets, cores, and threads, make sure that cores and threads belonging to a single socket are assigned to the same NUMA node. If threads or cores from the same socket are assigned to different NUMA nodes, the guest may fail to boot.

8.3.10. Assigning Host Huge Pages to Multiple Guest NUMA Nodes

In Red Hat Enterprise Linux 7.1 and above, huge pages from the host can be allocated to multiple guest NUMA nodes. This can optimize memory performance, as guest NUMA nodes can be moved to host NUMA nodes as required, while the guest can continue to use the huge pages allocated by the host.

After configuring the guest NUMA node topology (see Section 8.3.9, “Guest NUMA Topology” for details), specify the huge page size and the guest NUMA nodeset in the ***<memoryBacking>*** element in the guest XML. The ***page size*** and ***unit*** refer to the size of the huge pages from the host. The ***nodeset*** specifies the guest NUMA node (or several nodes) to which huge pages will be assigned.

In the following example, guest NUMA nodes 0-5 (except for NUMA node 4) will use 1GB huge pages, and guest NUMA node 4 will use 2MB huge pages, regardless of guest NUMA node placement on the host. To use 1GB huge pages in guests, the host must be booted first with 1GB huge pages enabled; see Section 7.2.3, “Huge Pages and Transparent Huge Pages (THP)” for instructions on enabling 1GB huge pages.

```
<memoryBacking>
        <hugepages/>
          <page size="1" unit="G" nodeset="0-3,5"/>
          <page size="2" unit="M" nodeset="4"/>
        </hugepages>
</memoryBacking>
```

This allows for greater control over huge pages in a situation where it is useful to merge some guest NUMA nodes onto a single host NUMA node, but continue to use different huge page sizes. For example, even if guest NUMA nodes 4 and 5 are moved to the host's NUMA node 1, both continue to use different sizes of huge pages.

Note

When using ***strict*** memory mode, the guest will fail to start when there are not enough huge pages available on the NUMA node. See Section 8.3.3, “Domain Processes” for a configuration example of the ***strict*** memory mode option within the **<numatune>** tag.

8.3.11. NUMA Node Locality for PCI Devices

When starting a new virtual machine, it is important to know both the host NUMA topology and the PCI device affiliation to NUMA nodes, so that when PCI passthrough is requested, the guest is pinned onto the correct NUMA nodes for optimal memory performance.

For example, if a guest is pinned to NUMA nodes 0-1, but one of its PCI devices is affiliated with node 2, data transfer between nodes will take some time.

In Red Hat Enterprise Linux 7.1 and above, libvirt reports the NUMA node locality for PCI devices in the guest XML, enabling management applications to make better performance decisions.

This information is visible in the **sysfs** files in **/sys/devices/pci*/*/numa_node**. One way to verify these settings is to use the **lstopo** tool to report **sysfs** data:

```
# 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"
```

This output shows:

- NICs **em*** and disks **sd*** are connected to NUMA node 0 and cores 0,2,4,6,8,10,12,14.
- NICs **p1*** and **ib*** are connected to NUMA node 1 and cores 1,3,5,7,9,11,13,15.

8.4. NUMA-Aware Kernel SamePage Merging (KSM)

Kernel SamePage Merging (KSM) allows virtual machines to share identical memory pages. KSM can detect that a system is using NUMA memory and control merging pages across different NUMA nodes.

Use the **sysfs /sys/kernel/mm/ksm/merge_across_nodes** parameter to control merging of pages across different NUMA nodes. By default, pages from all nodes can be merged together. When this parameter is set to zero, only pages from the same node are merged.

Generally, unless you are oversubscribing the system memory, you will get better runtime performance by disabling KSM sharing.

Important

When KSM merges across nodes on a NUMA host with multiple guest virtual machines, guests and CPUs from more distant nodes can suffer a significant increase of access latency to the merged KSM page.

To instruct the hypervisor to disable share pages for a guest, add the following to the guest's XML:

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

For more information about tuning memory settings with the **<memoryBacking>** element, see Section 7.2.2, “Memory Tuning with virsh”.

Appendix A. Revision History

Revision 1.0-23	**Thu Aug 18 2016**	**Jiri Herrmann**

Preparing document for 7.3 Beta publication

Revision 1.0-22	**Mon Dec 21 2015**	**Laura Novich**

Republished Guide to fix several bugs

Revision 1.0-19	**Thu Oct 08 2015**	**Jiri Herrmann**

Cleaned up Revision History