Red Hat Enterprise Linux 7

虚拟化调整和优化指南

在 RHEL 中为主机系统和虚拟客户机使用 KVM 性能功能

Last Updated: 2023-03-25

Red Hat Enterprise Linux 7 虚拟化调整和优化指南

在 RHEL 中为主机系统和虚拟客户机使用 KVM 性能功能

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.

法律通告

摘要

Red Hat Enterprise Linux 虚拟化调整和优化指南涵盖 KVM 和虚拟化性能。在本指南中，您可以找到关于为主机系统和虚拟客户机充分利用 KVM 性能功能和选项的建议。

目录

第 1 章 简介

1.1. 为何在虚拟化中进行性能优化问题

在 KVM 虚拟化中，客户机由主机上的进程表示。这意味着，处理主机的功能、内存和其他资源用于模拟客户机的虚拟硬件的功能和功能。

但是，客户机硬件在使用比主机的资源更不用时。因此，可能需要调整分配的主机资源的数量，以便虚拟机以预期速度执行其任务。此外，各种类型的虚拟硬件可能会具有不同的开销级别，因此适当的虚拟硬件配置可能会对客户机性能产生重大影响。最后，根据具体配置，特定配置可让虚拟机更有效地使用主机资源。

1.2. KVM 性能架构概述

以下点概述 KVM 与系统性能有关，以及流程和线程管理：

- 在使用 KVM 时，guest 作为 Linux 进程在主机上运行。
- 虚拟 CPU(vCPU)作为常规线程实施，由 Linux 调度程序处理。
- 客户机不会自动继承内核中的 NUMA 和巨页等功能。
- 主机中的磁盘和网络 I/O 设置对性能有显著影响。
- 网络流量通常通过基于软件的网桥进行传输。
- 根据设备及其模型，由于该特定硬件模拟，可能会有大量开销。

1.3. 虚拟化性能功能和改进

Red Hat Enterprise Linux 7 中的虚拟化性能

以下功能提高 Red Hat Enterprise Linux 7 中的虚拟化性能：

自动 NUMA Balancing

自动 NUMA 平衡提高了在 NUMA 硬件系统上运行的应用程序的性能，无需任何 Red Hat Enterprise Linux 7 客户机所需的手动调整。自动 NUMA 平衡移动任务（可以是线程或进程），更接近他们访问的内存。这可启用零配置的良好性能。然而，在某些情况下，提供更准确的客户机配置或将 guest 设置为托管 CPU 和内存的影响可能会提供更好的结果。

有关自动 NUMA 平衡的更多信息，请参阅 第 9.2 节 “自动 NUMA Balancing”。

virtio 型号

任何具有 virtio 模型的虚拟硬件都没有模拟所有特定硬件的开销。VirtIO 设备拥有较低的开销，因为事实上专为虚拟化环境而设计。然而，并非所有客户机操作系统都支持此类模型。

多队列 virtio-net

种联网方法，它允许数据包发送/接收处理，以便通过客户机的可用 vCPU 数量进行扩展。

有关多队列 virtio-net 的详情，请参考 第 5.4.2 节 “multi-Queue virtio-net”。

bridge Zero Copy Transmit

零复制传输模式可降低主机 CPU 开销，从而降低在客户机网络和外部网络之间传输大型数据包的开销，且不会影响吞吐量。在 Red Hat Enterprise Linux 7 虚拟机上完全支持桥接零副本传输，但默认禁用。

有关零副本传输的详情请参考 第 5.4.1 节 “bridge Zero Copy Transmit”。

APIC 虚拟化(APICv)

更新的 Intel 处理器提供高级程序中断控制器(APICv)的硬件虚拟化。APICv 通过允许客户机直接访问 APIC 提高了虚拟化 AMD64 和 Intel 64 客户机性能，显著降低中断延迟以及 APIC 导致的虚拟机数量。此功能默认使用新的 Intel 处理器，提高 I/O 性能。

EOI Acceleration

较旧芯片组上的高带宽 I/O 结束中断加速，无需虚拟 APIC 功能。

多队列 virtio-scsi

提高了 virtio-scsi 驱动程序中的多队列支持存储性能和可扩展性。这可让每个虚拟 CPU 有一个单独的队列和中断，以便在不影响其他 vCPU 的情况下使用。

有关多队列 virtio-scsi 的详情，请参考 第 7.4.2 节 “multi-Queue virtio-scsi”。

半虚拟化的 Ticketlocks

半虚拟化票据锁定(pvticketlocks)提高了在具有超额订阅 CPU 的 Red Hat Enterprise Linux 7 主机上运行的红帽企业 Linux 7 虚拟机的性能。

半虚拟页面故障

当虚拟页面错误试图访问主机交换后的页面时，会将半虚拟化页面错误注入到客户机中。这可提高在主机内存被过量使用且客户机内存被交换时的 KVM 客户机性能。

半虚拟时间 vsyscall Optimization

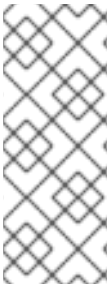
gettimeofday 和 **clock_gettime** 系统调用通过 **vsyscall** 机制在用户空间中执行。在以前的版本中，发出这些系统调用需要系统切换到内核模式，然后返回到用户空间。这可以大大提高某些应用程序的性能。

Red Hat Enterprise Linux 中的虚拟化性能功能

- CPU/Kernel
  - NUMA - 非一致性内存访问.有关 NUMA 的详情，请参阅 第 9 章 *NUMA*。
  - CFS - 完全公平调度程序.先进的面向类的调度程序。
  - RCU - 读取复制更新.更好地处理共享线程数据。
  - 最多 160 个虚拟 CPU(vCPU)。
- memory
  - 用于内存密集型环境的巨页和其他优化。详情请查看 第 8 章 *memory*。
- Networking
  - vhost-net - 基于内核的快速 VirtIO 解决方案。

- SR-IOV - 用于接近原生的网络性能级别。

- 块 I/O
  - AIO - 对线程的支持与其他 I/O 操作重叠。
  - MSI - PCI 总线设备中断生成。
  - 磁盘 I/O 节流 - 控制客户机磁盘 I/O 请求，以防止过度利用主机资源。详情请查看 第 7.4.1 节 “磁盘 I/O Throttling”。



注意

有关虚拟化支持、限制和功能的详情，请查看 *Red Hat Enterprise Linux 7 虚拟化入门指南* 以及以下 URL：

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

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

第 2 章 性能监控工具

本章论述了用于监控客户机虚拟机环境的工具。

2.1. PERF KVM

您可以使用 **perf** 命令和 **kvm** 选项从主机收集和分析客户机操作系统统计信息。perf 软件包提供 **perf** 命令。它通过运行以下命令来安装：

```
# yum install perf
```

要在主机中使用 **perf kvm**，您必须有权访问客户机中的 **/proc/modules** 和 **/proc/kallsyms** 文件。请参阅 过程 2.1, “将 /proc 文件从 guest 复制到主机”，将文件传送到主机中，并对文件运行报告。

过程 2.1. 将 /proc 文件从 guest 复制到主机

重要

如果您直接复制所需的文件（例如，使用 **scp**），您将只复制零长度的文件。这个步骤描述了如何先将客户机中的文件保存到临时位置（使用 **cat** 命令），然后将它们复制到主机，供 **perf kvm** 使用。

1. **登录到客户端并保存文件**
   登录到客户端并将 **/proc/modules** 和 **/proc/kallsyms** 保存到临时位置 **/tmp** 中：

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

2. **将临时文件复制到主机**
   从 guest 注销后，运行以下命令的 **scp** 命令将保存的文件复制到主机。如果主机名和 TCP 端口不同，您应该替换它们：

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

   现在，主机上 guest（**guest-kallsyms** 和 **guest-modules**）有两个文件，可供 **perf kvm** 使用。

3. **使用 perf kvm 记录和报告事件**
   使用在前面的步骤中获取的文件，现在有可能记录和报告客户机、主机或两者中的事件。

   运行以下命令：

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

注意

如果命令中同时使用 **--host** 和 **--guest**，则输出将存储在 **perf.data.kvm** 中。如果只使用 **--host**，则该文件将命名为 **perf.data.host**。同样，如果只使用 **--guest**，则该文件将命名为 **perf.data.guest**。

按 Ctrl-C 停止记录。

4. **报告事件**
以下示例使用由记录进程获取的 文件，并将输出重定向到新文件，从而 **分析**。

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

查看 **分析** 文件的内容，以检查记录的事件：

```
# 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. 虚拟性能监控单元(VPMU)

虚拟性能监控单元(vPMU)显示表明 guest 虚拟机运行情况的统计信息。

虚拟性能监控单元允许用户识别其客户机虚拟机中可能出现性能问题的来源。vPMU 基于 Intel 的 PMU（性能监控单元），且只能在 Intel 机器上使用。

这个功能只支持运行 Red Hat Enterprise Linux 6 或 Red Hat Enterprise Linux 7 的客户机虚拟机，且默认禁用。

要验证您的系统是否支持 vPMU，请运行以下命令检查主机 CPU 上的 **arch_perfmon** 标志：

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

要启用 vPMU，在客户机 XML 中将 ***cpu mode*** 指定为 **host-passthrough** ：

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

启用 vPMU 后，通过从客户机虚拟机运行 **perf** 命令来显示虚拟机的性能统计信息。

2.3. 监控虚拟机管理器中的性能

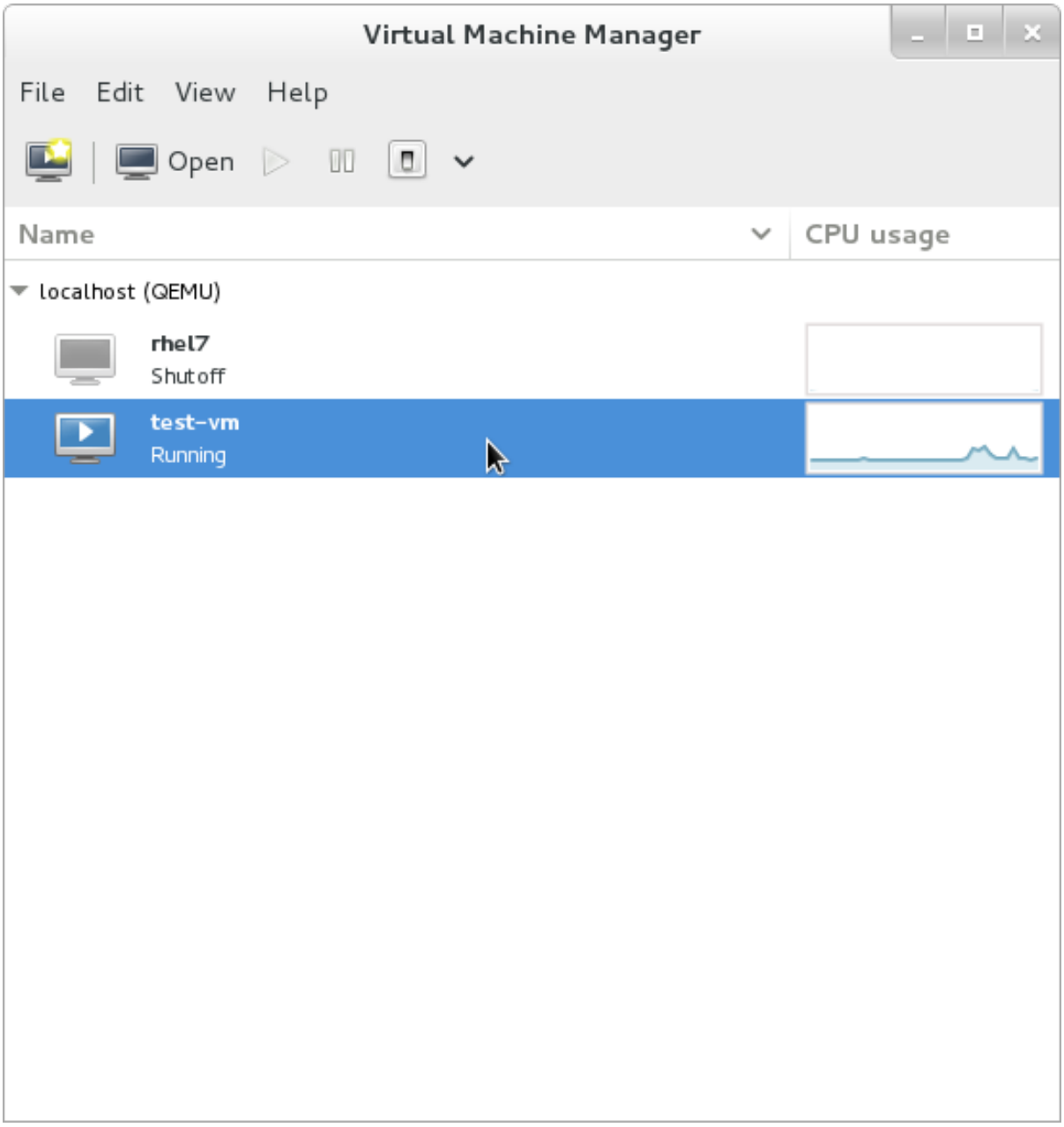
您可以使用 Virtual Machine Monitor 来查看系统上任何虚拟机的性能信息。您还可以配置虚拟机管理器中显示的性能信息。

2.3.1. 查看虚拟机管理器中的性能概述

使用 Virtual Machine Manager 查看虚拟机的性能概述：

1. 在 Virtual Machine Manager 主窗口中，突出显示您要查看的虚拟机。

   图 2.1. 选择要显示的虚拟机



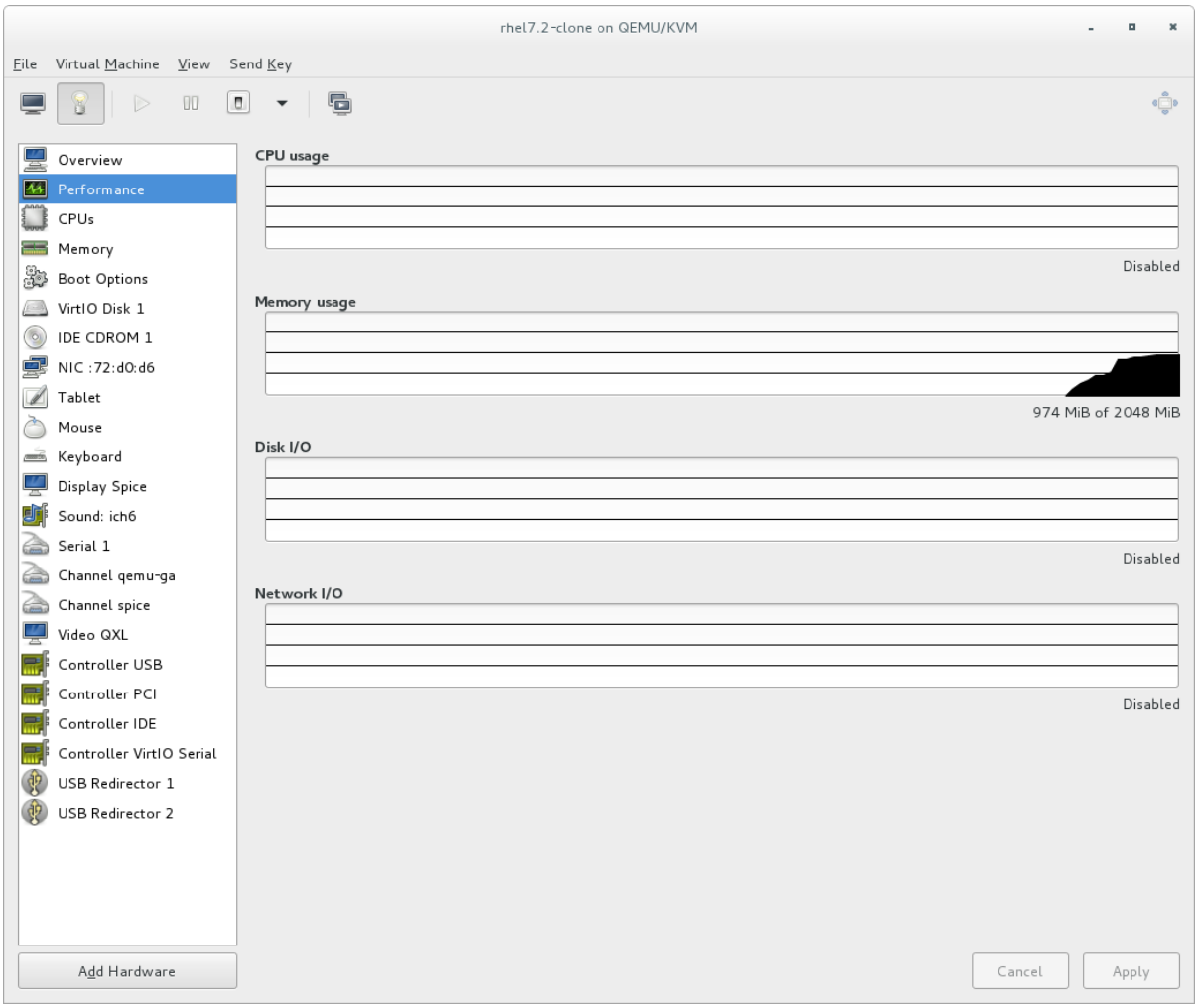
2. 在 Virtual Machine Manager **Edit** 菜单中，选择 **Virtual Machine Details**。

   当 Virtual Machine 详情窗口打开时，可能会显示控制台。如果发生这种情况，请单击" **查看** "，然后选择" **详细信息** "。默认会首先打开 Overview 窗口。

3. 从左侧的导航窗格中选择 **Performance**。

   性能 视图显示客户机性能的摘要，包括 CPU 和内存使用情况以及磁盘以及网络输入和输出。

图 2.2. 显示客户机性能详情



2.3.2. 性能监控

可以通过 **virt-manager** 的首选项修改性能监控首选项。

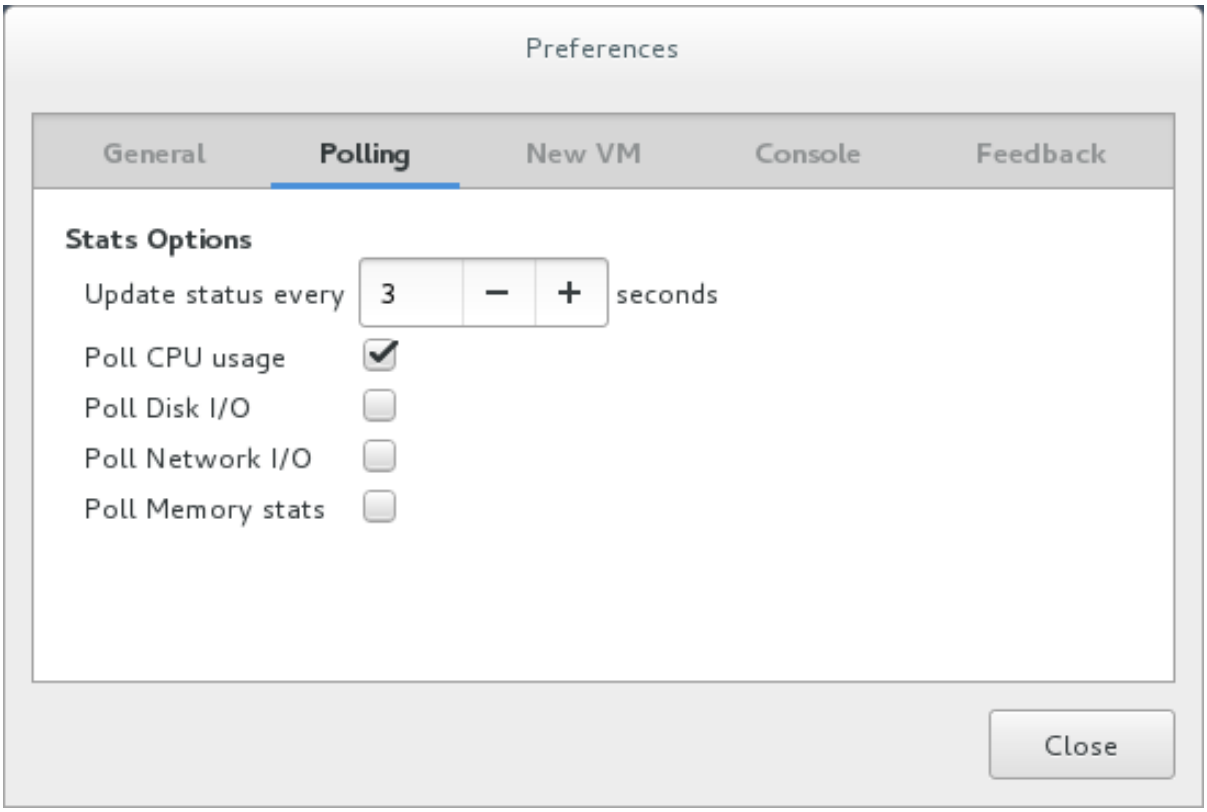
配置性能监控：

1. 在" **编辑** "菜单中，选择 **"首选项** "。

   此时会出现 **Preferences** 窗口。

2. 在 **Polling** 选项卡中指定时间（以秒为单位）或 stats 轮询选项。

图 2.3. 配置性能监控

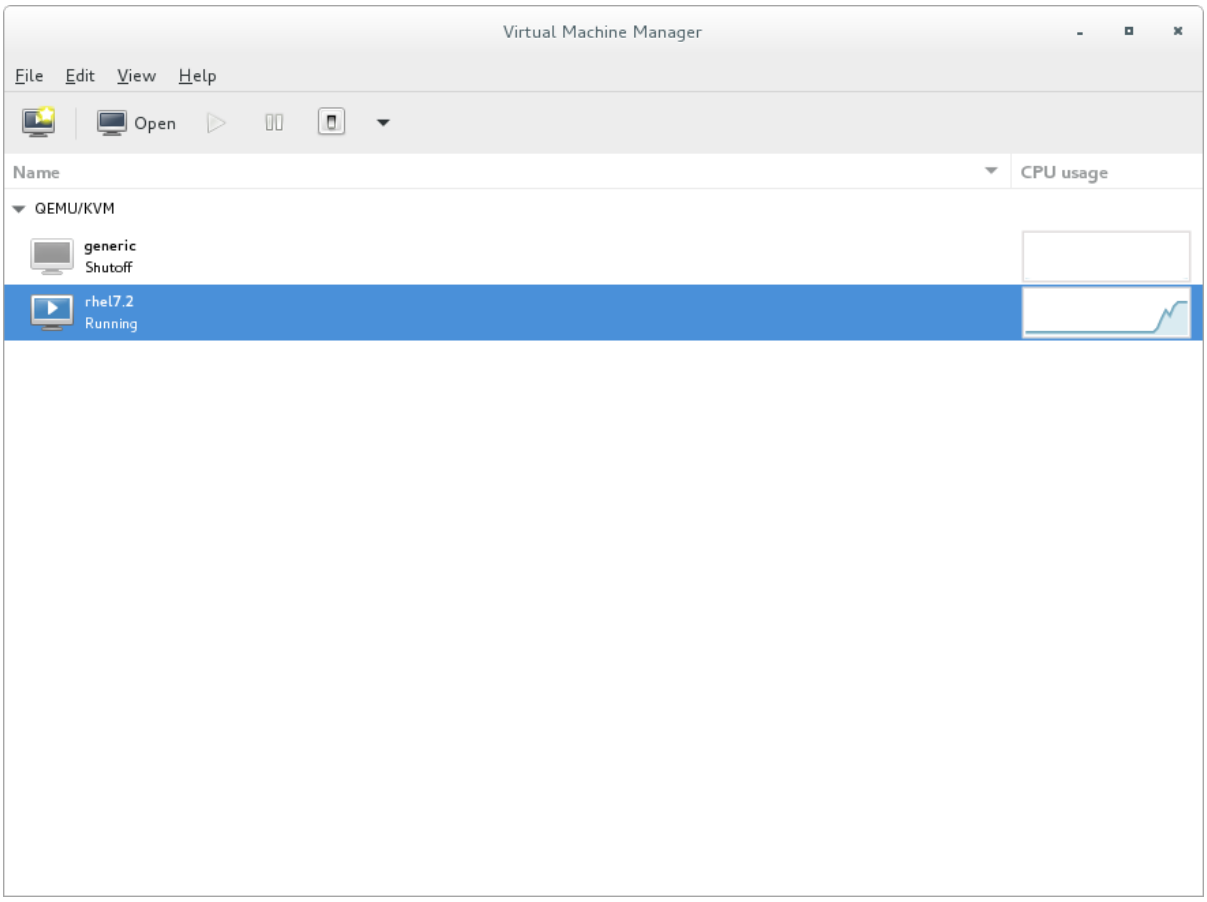


2.3.3. 显示客户机的 CPU 用量

查看系统中所有客户端的 CPU 使用量：

1. 在 **View** 菜单中，选择 **Graph**，然后选择 **Guest CPU Usage** 复选框。
2. Virtual Machine Manager 显示系统上所有虚拟机的 CPU 使用量图。

图 2.4. 客户机 CPU 用量图

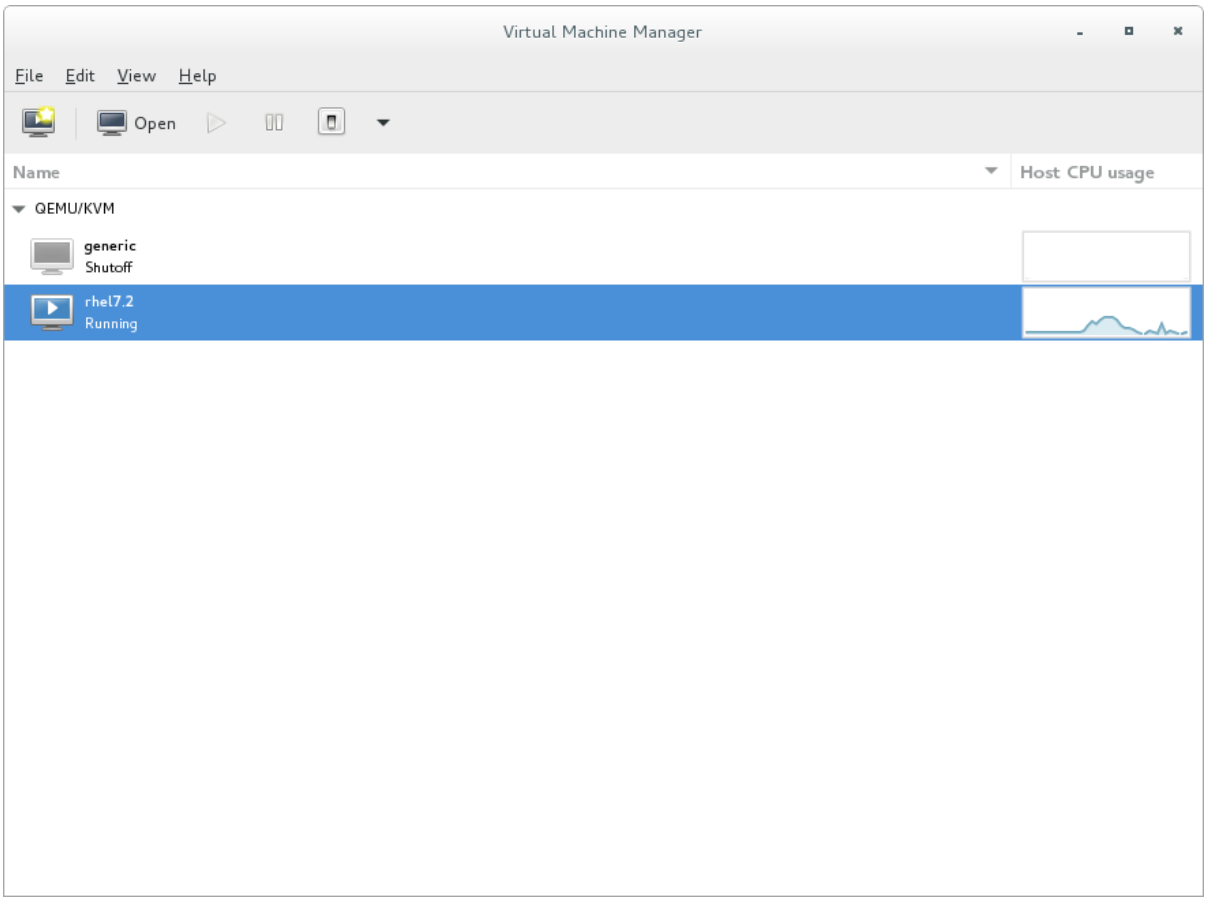


2.3.4. 显示主机的 CPU 用量

查看系统中所有主机的 CPU 使用量：

1. 在 **View** 菜单中，选择 **Graph**，然后选择 **Host CPU Usage** 复选框。
2. 虚拟机管理器显示系统中主机 CPU 用量图表。

图 2.5. 主机 CPU 用量图



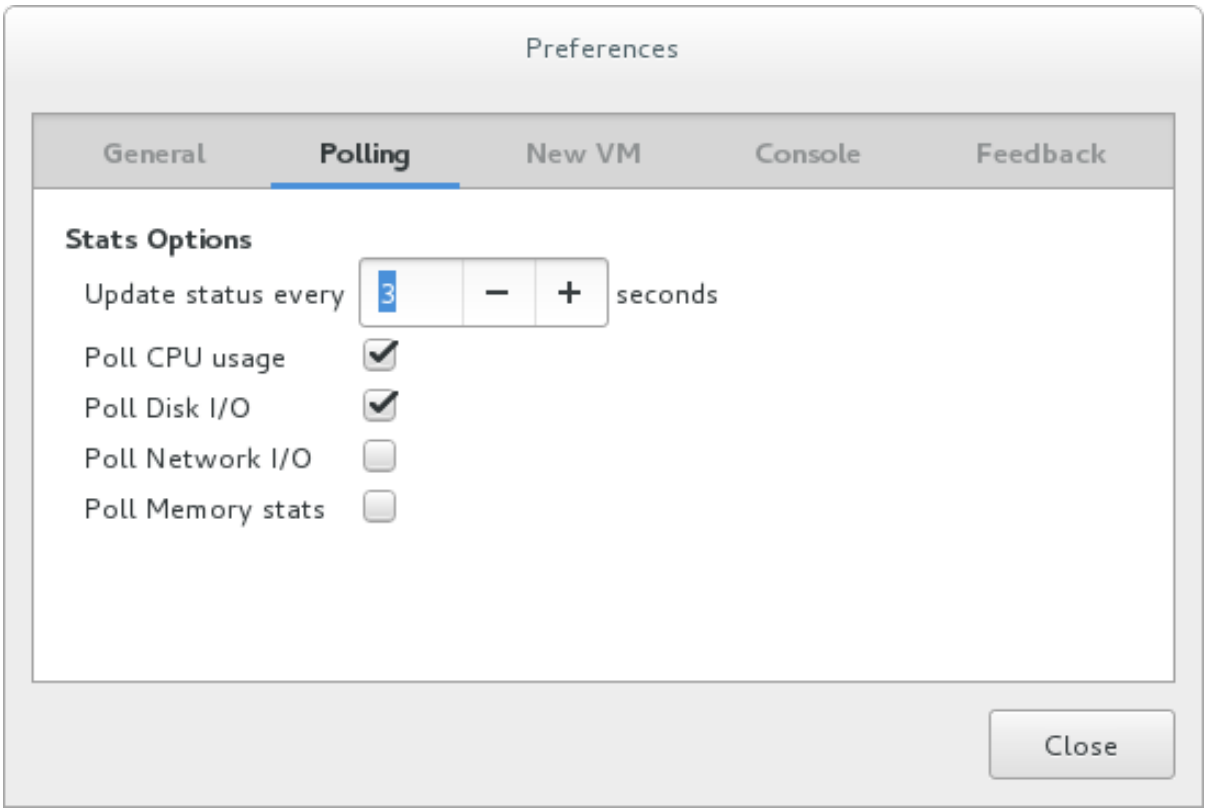
2.3.5. 显示磁盘 I/O

查看系统中所有虚拟机的磁盘 I/O：

1. 确保启用了磁盘 I/O 统计数据集合。为此，请从"**编辑**"菜单中选择"**首选项**"并单击"**轮询**"选项卡。

2. 选择 **Disk I/O** 复选框。

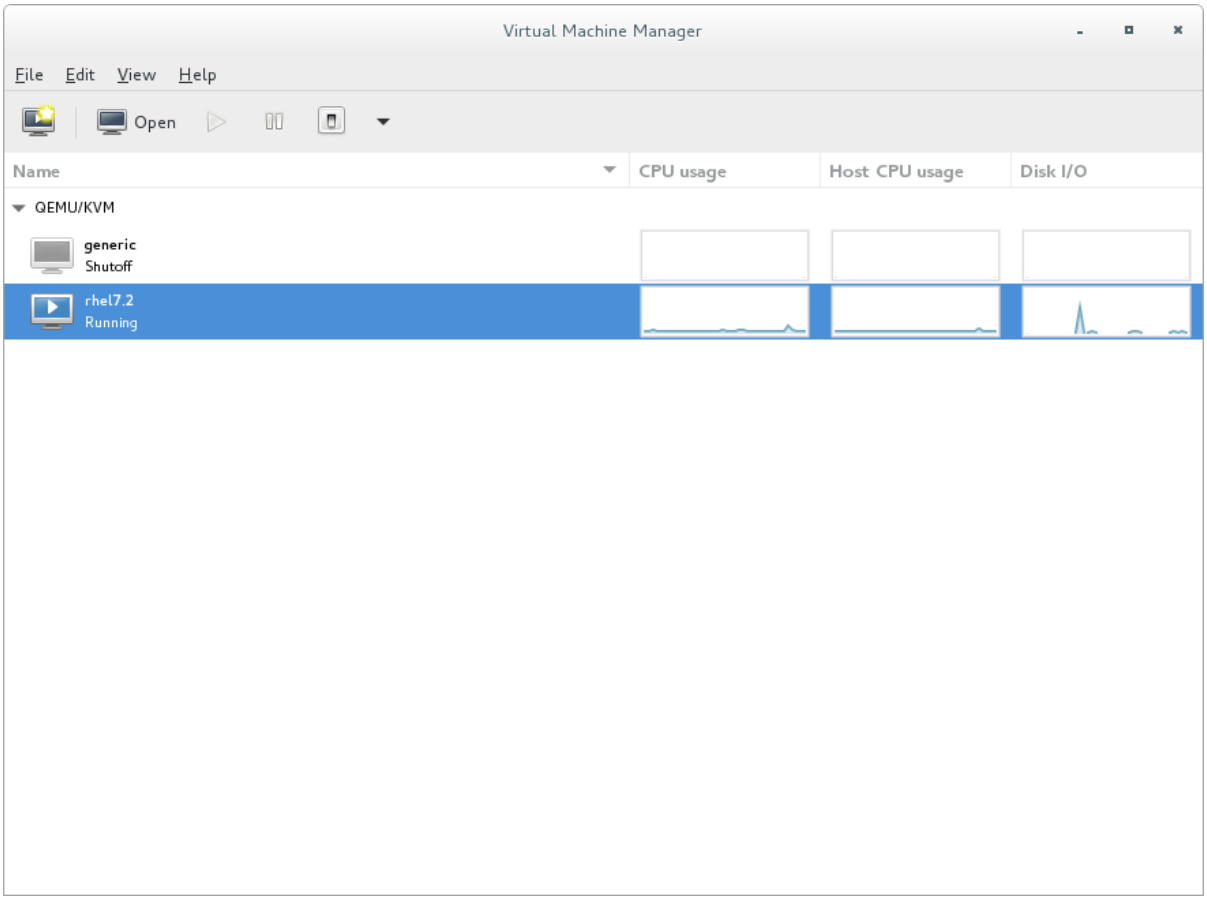
图 2.6. 启用磁盘 I/O



3. 要启用 Disk I/O 显示，从 **View** 菜单中选择 **Graph**，然后选择 **Disk I/O** 复选框。

4. 虚拟机管理器显示系统上所有虚拟机的磁盘 I/O 图表。

图 2.7. 显示磁盘 I/O



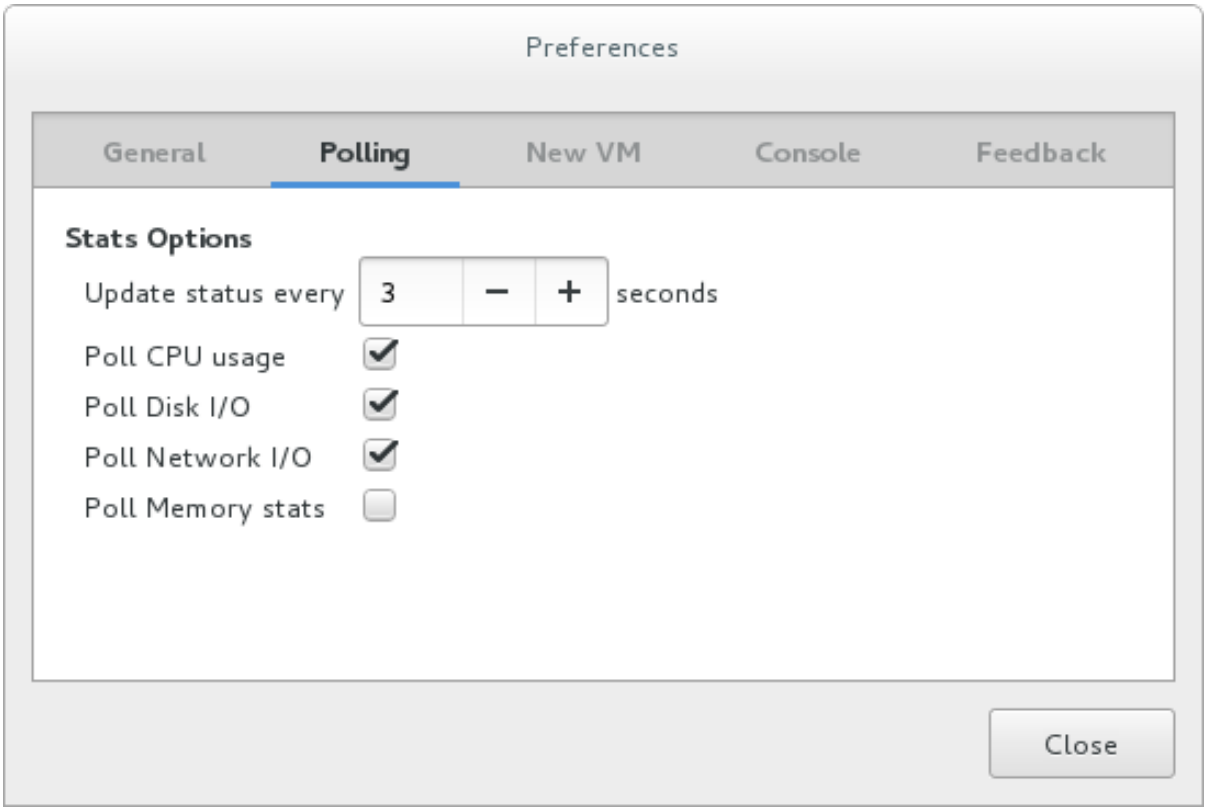
2.3.6. 显示网络 I/O

查看系统中所有虚拟机的网络 I/O：

1. 确保已启用网络 I/O 统计数据集合。为此，请从" **编辑** "菜单中选择" **首选项"** 并单击" **轮询**"选项卡。

2. 选中 **Network I/O** 复选框。

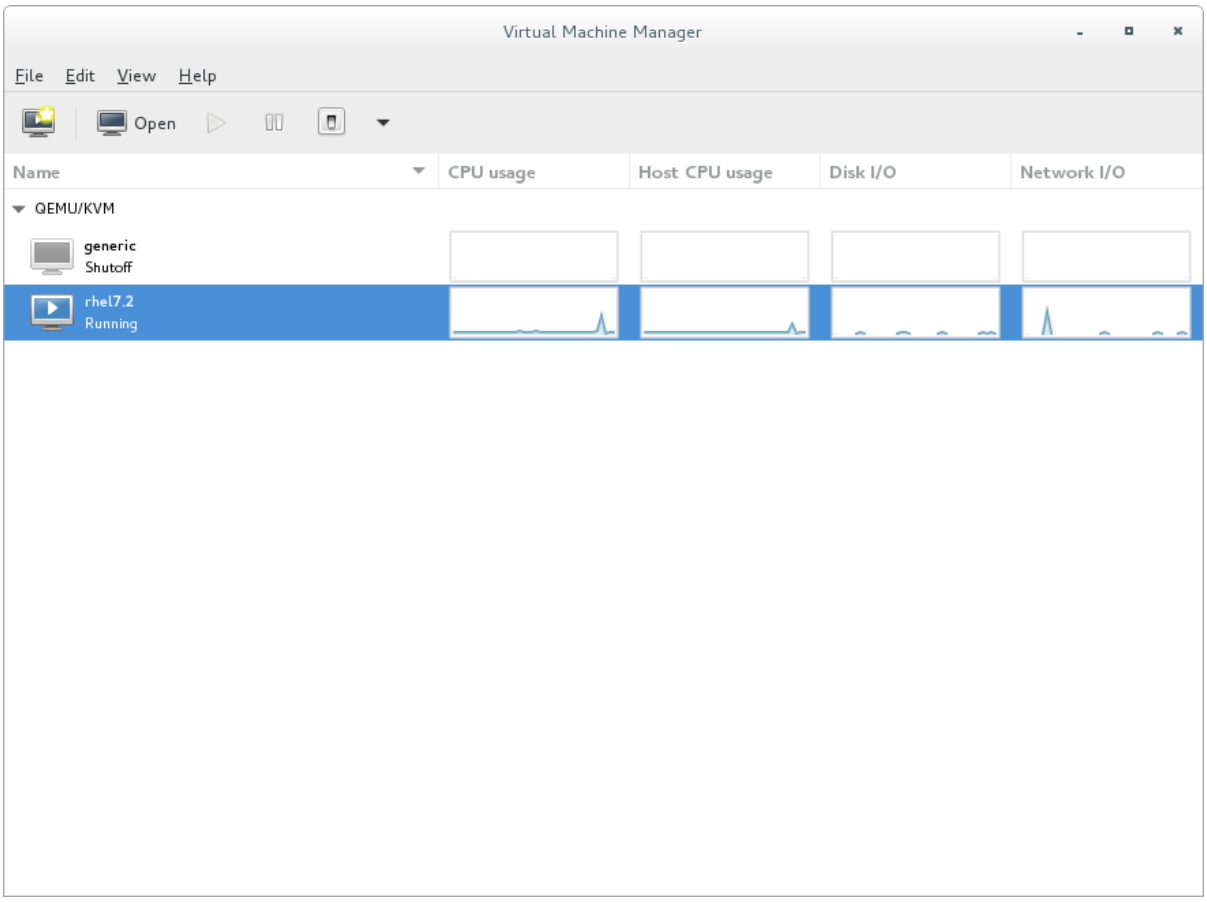
   图 2.8. 启用网络 I/O



3. 要显示 Network I/O 统计信息，请从 **View** 菜单中选择 **Graph**，然后选择 **Network I/O** 复选框。

4. 虚拟机管理器显示系统上所有虚拟机的网络 I/O 图表。

图 2.9. 显示网络 I/O

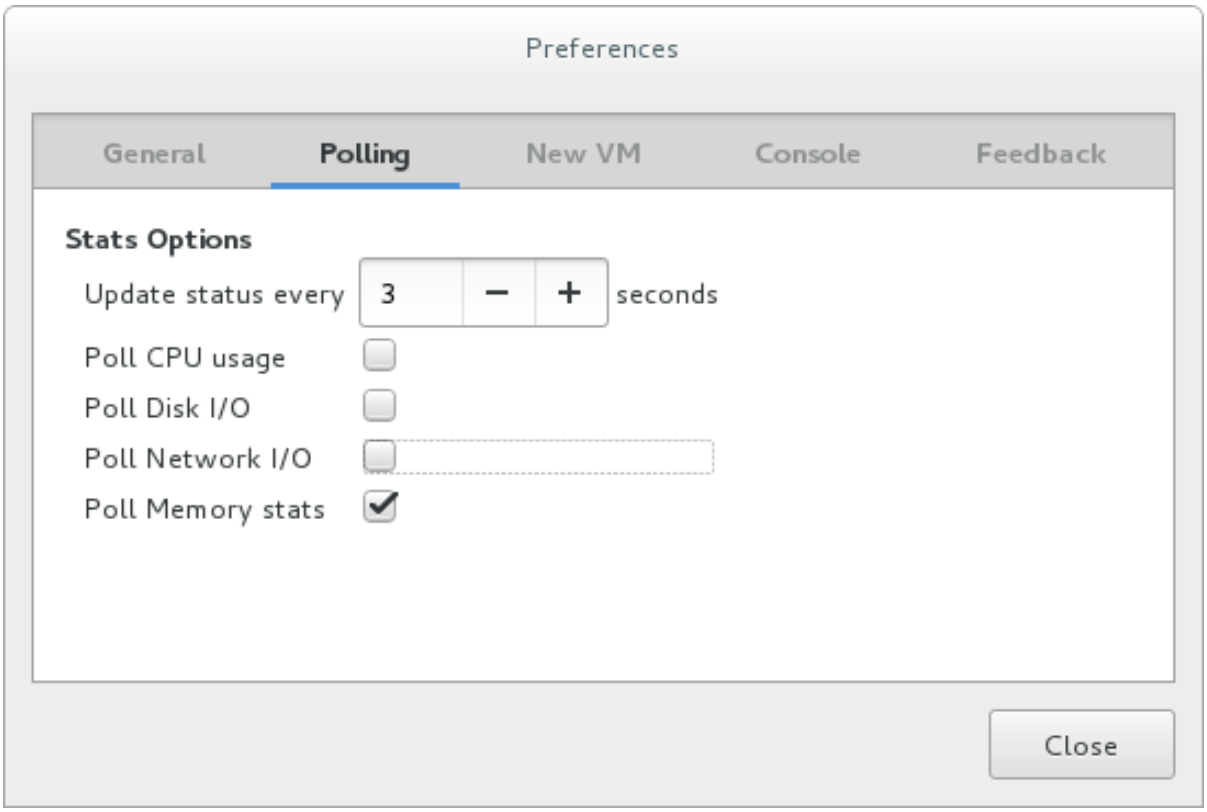


2.3.7. 显示内存使用情况

查看系统中所有虚拟机的内存用量：

1. 确保启用了内存用量统计集合。为此，请从"**编辑**"菜单中选择"**首选项**"并单击"**轮询**"选项卡。
2. 选择 **Poll Memory stats** 复选框。

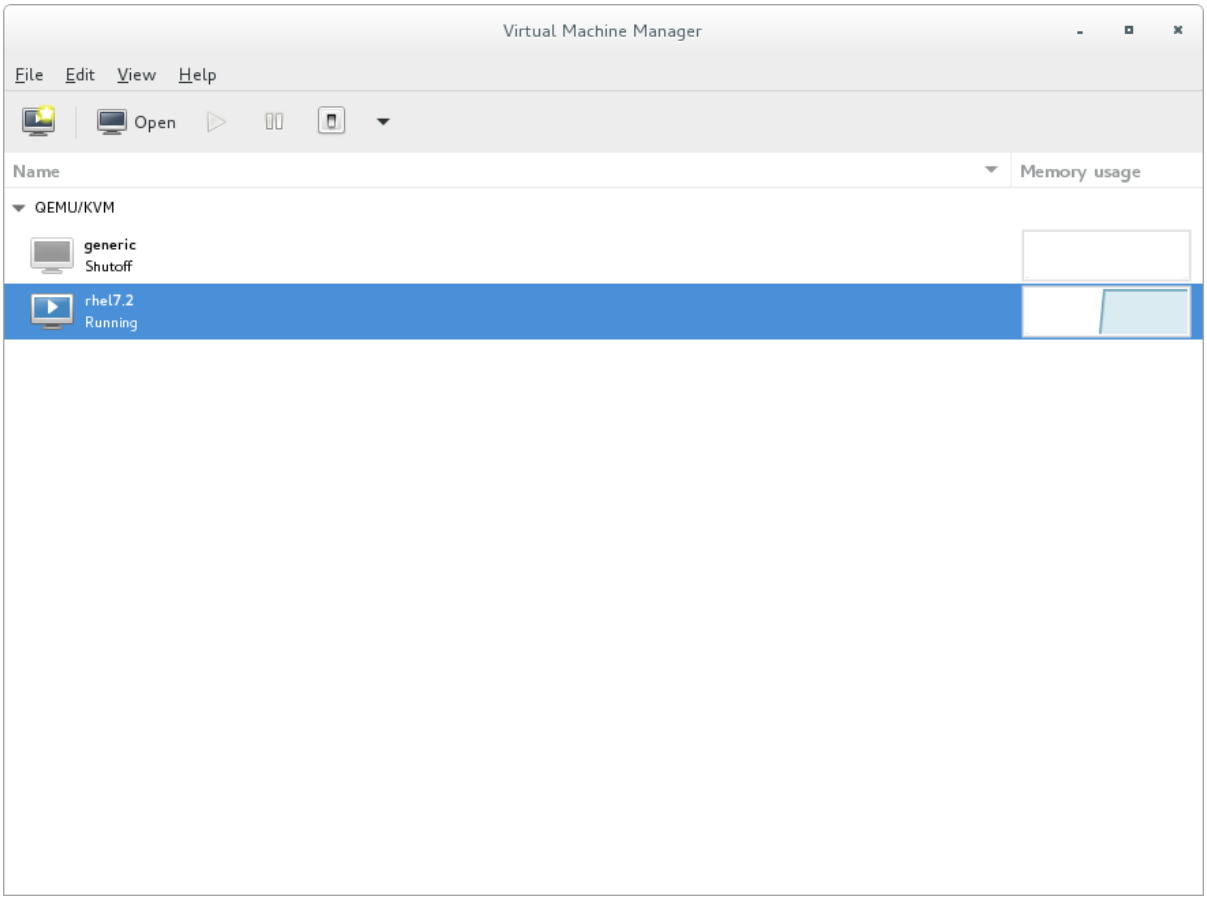
图 2.10. 启用内存用量



3. 要显示内存使用情况，请从 **View** 菜单中选择 **Graph**，然后选择 **Memory Usage** 复选框。

4. Virtual Machine Manager 列出了系统中所有虚拟机使用的内存百分比（以 MB 为单位）。

图 2.11. 显示内存用量



第 3 章 使用 VIRT-MANAGER 优化虚拟化性能

本章论述了 **virt-manager** 中的性能调优选项，这是用于管理 guest 虚拟机的桌面工具。

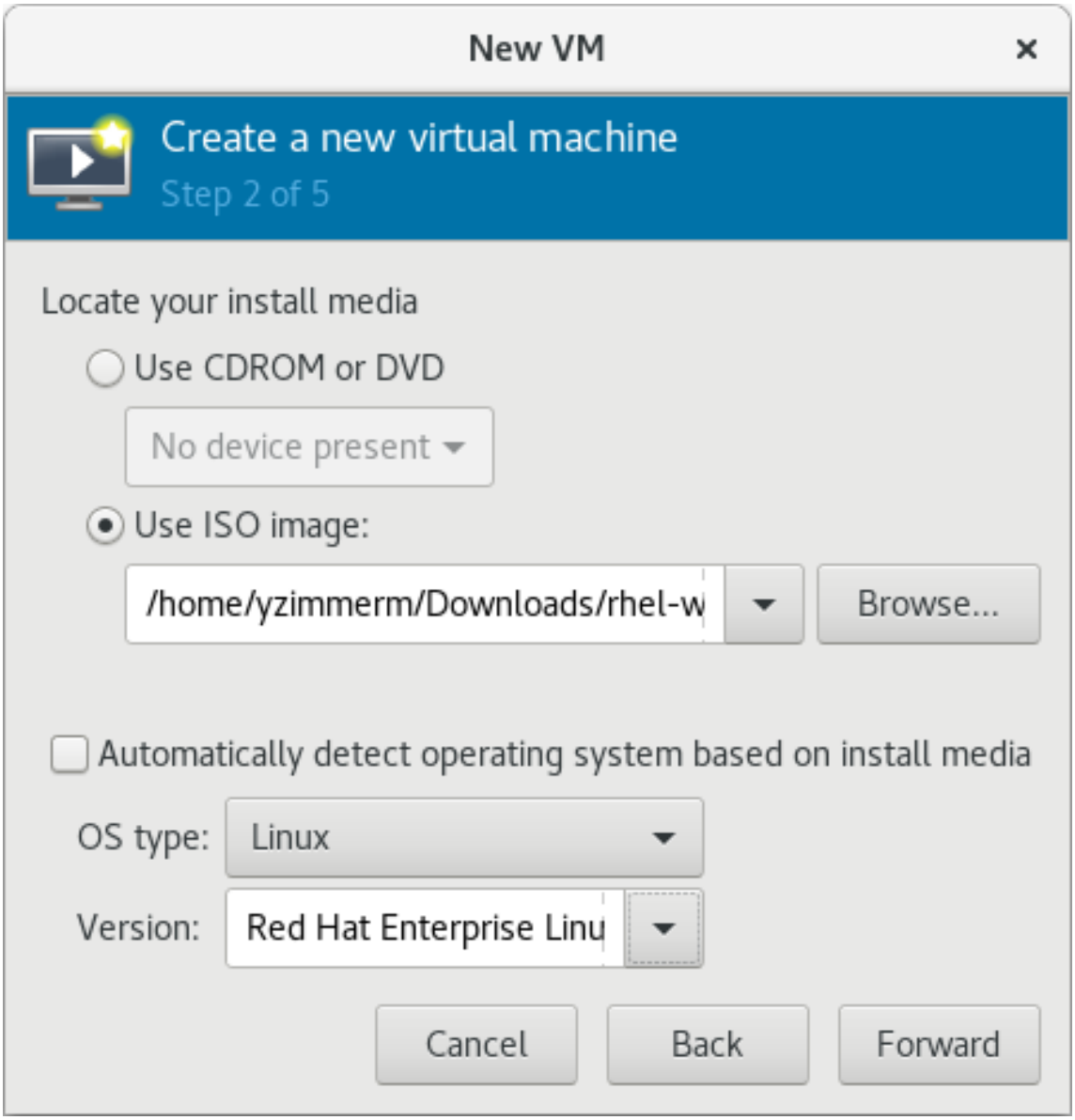
3.1. 操作系统详情和设备

3.1.1. 指定虚拟客户机详情

virt-manager 工具根据为新客户机虚拟机选择的操作系统类型和版本提供不同的配置集。在创建 guest 时，您应该提供尽可能多的详细信息；这可以通过启用特定类型的功能来提高性能。

请参阅 **virt-manager** 工具的示例屏幕截图。在创建新客户端虚拟机时，请始终指定预期的 **操作系统类型和版本** ：

图 3.1. **提供操作系统类型和版本**

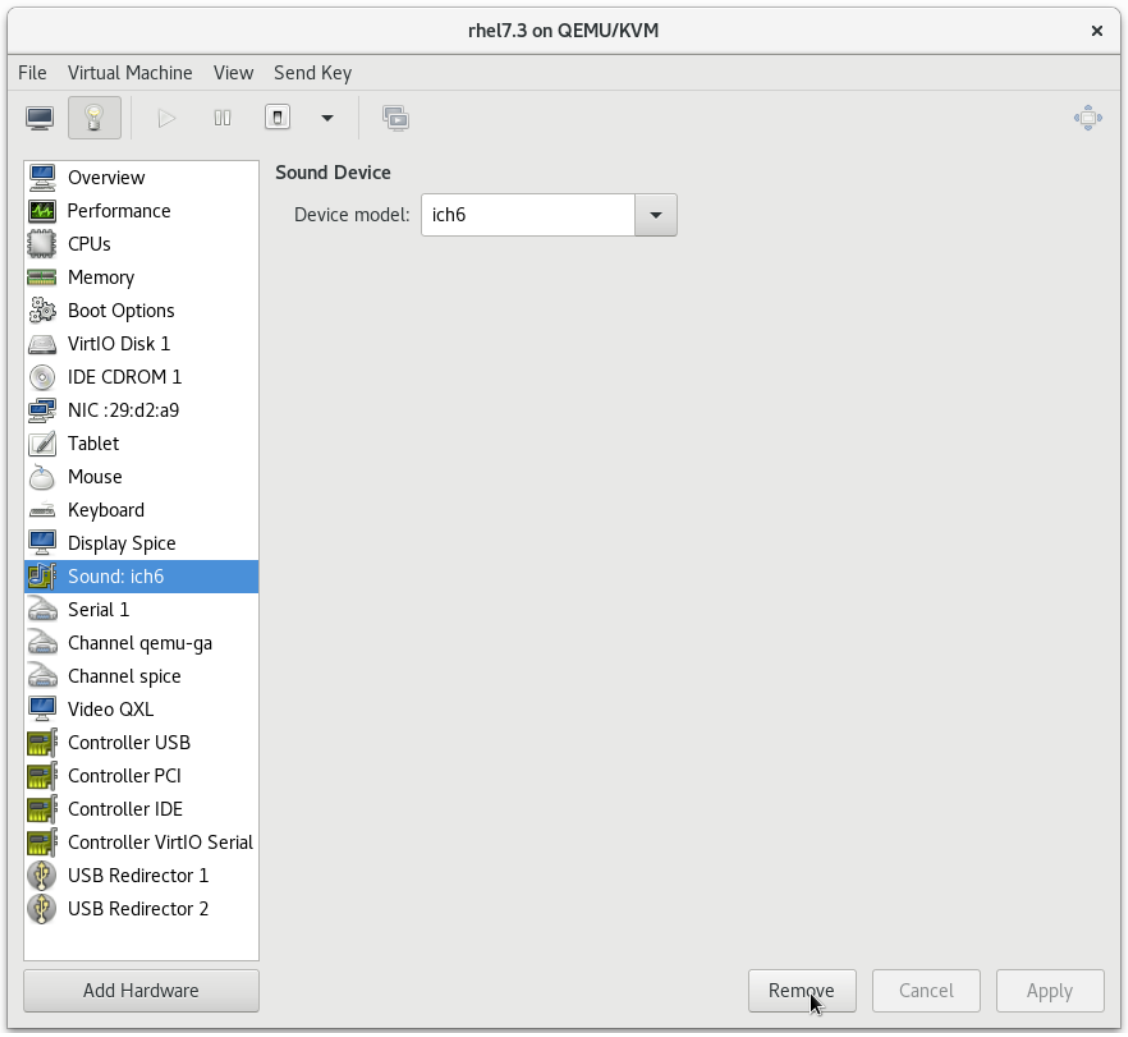


3.1.2. 删除未使用的设备

删除未使用或不必要的设备可以提高性能。例如，作为 Web 服务器的客户机不大需要音频功能或附加表格。

请参阅 **virt-manager** 工具的示例屏幕截图。点击 **Remove** 按钮以删除不必要的设备：

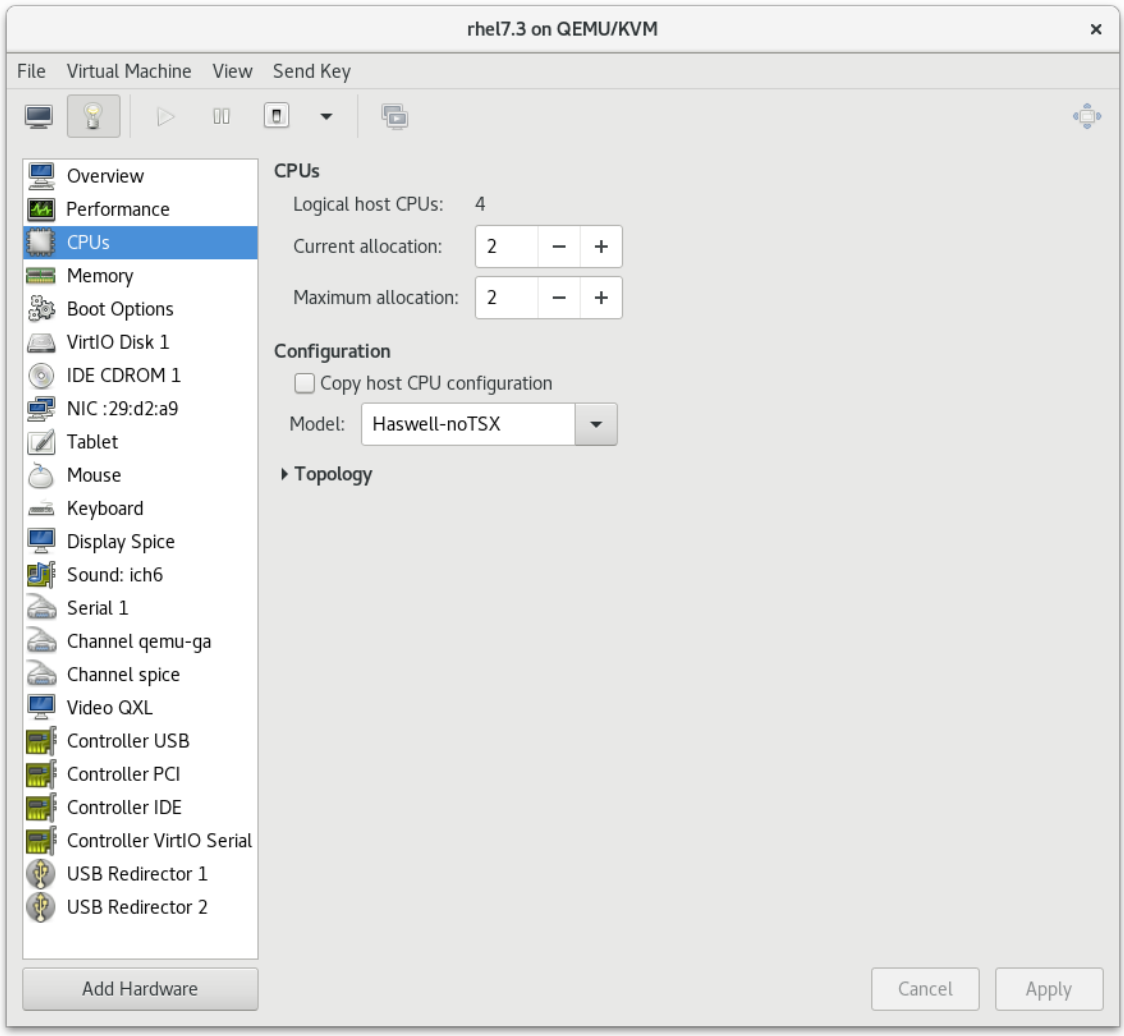
图 3.2. 删除未使用的设备



3.2. CPU 性能选项

您的客户机虚拟机可使用几个与 CPU 相关的选项。正确配置了，这些选项可能会对性能产生较大影响。下图显示了可供您的客户机使用的 CPU 选项。本节的其余部分显示并解释这些选项的影响。

图 3.3. CPU 性能选项



3.2.1. 选项：可用的 CPU

使用这个选项调整客户端可用的虚拟 CPU(vCPU)量。如果主机上分配了多于 （*过量使用*）的信息，则会显示警告，如下所示：

图 3.4. CPU 过量使用

如果系统中所有客户机的 vCPU 大于系统中主机 CPU 的数量，则 CPU 会被过量使用。如果 vCPU 总数大于主机 CPU 的数量，则可使用一个或多个客户机过量使用 CPU。

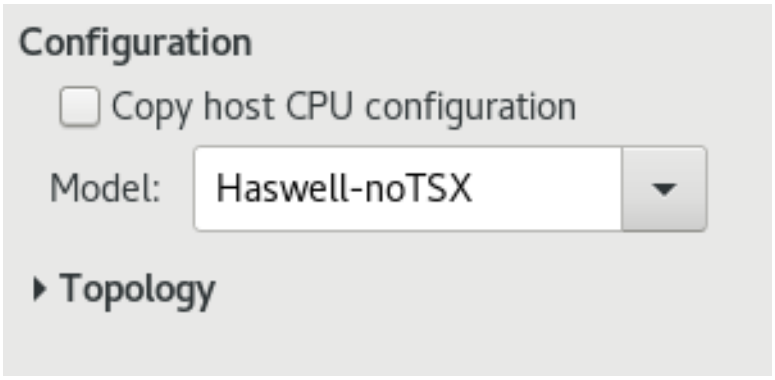
重要

与内存过量使用一样，CPU 过量使用可能会对性能造成负面影响，例如在出现大量或无法预计的虚拟机工作负载的情况下。有关过量使用的详情，*请参阅虚拟化部署和管理指南*。

3.2.2. 选项：CPU 配置

使用这个选项根据预期的 CPU 型号选择 CPU 配置类型。点 *Copy host CPU configuration* 复选框来检测并应用物理主机的 CPU 模型和配置，或者展开列表来查看可用选项。选择 CPU 配置后，其可用 CPU 功能/扩展会显示，可在 *CPU 功能* 列表中单独启用/禁用。

图 3.5. CPU 配置选项



注意

建议手动配置复制主机 CPU 配置。

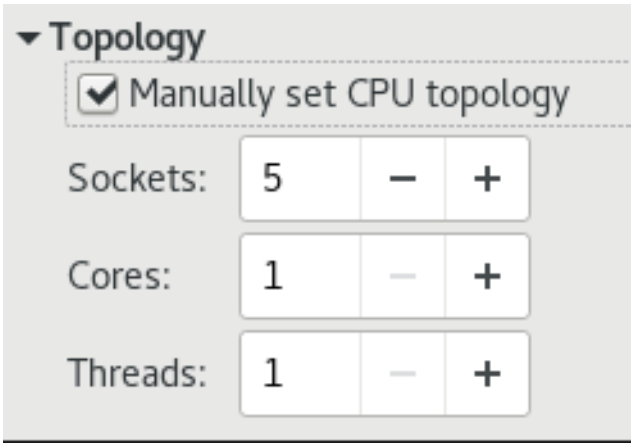
注意

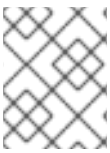
或者，在主机机器上运行 **virsh capabilities** 命令以查看系统的虚拟化功能，包括 CPU 类型和 NUMA 功能。

3.2.3. 选项：CPU Topology

使用这个选项将特定的 CPU 拓扑（插槽、内核、线程）应用到客户机虚拟机的虚拟 CPU。

图 3.6. CPU 拓扑选项





注意

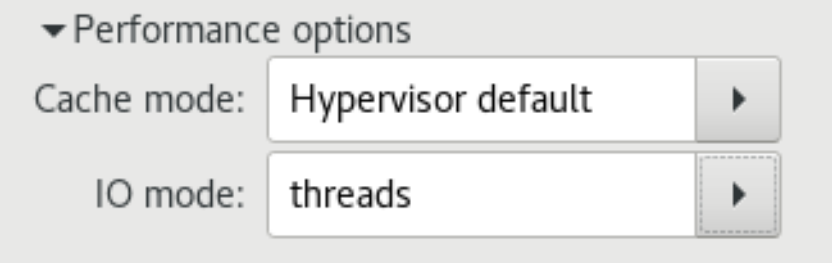
虽然您的环境可能会指定其他要求，但选择任何指定数量的插槽，但只有一个核心和一个线程通常会获得最佳性能结果。

3.3. 虚拟磁盘性能选项

可以在安装过程中为您的客户端虚拟机提供一些与虚拟磁盘相关的选项，这会影响性能。下图显示了可供您的客户机使用的虚拟磁盘选项。

可以在 virt-manager 的 **Virtual Disk** 部分中选择缓存模式、IO 模式和 IO 调整。在 **Performance** 选项下的字段中设置这些参数，如下镜像中所示：

图 3.7. 虚拟磁盘性能选项



重要

当在 virt-manager 中设置虚拟磁盘性能选项时，必须重启虚拟机才能使设置生效。

有关这些设置的描述以及在客户机 XML 配置中编辑这些设置的说明，请参阅 第 7.2 节 “Caching” 和 第 7.3 节 “I/O 模式”。

第 4 章 TUNED 和 TUNED-ADM

本章论述了使用 **tuned** 守护进程来调优虚拟环境中的系统设置。

Tuned 是一个调优的配置文件 交付机制，可针对某些工作负载特性调整 Red Hat Enterprise Linux，如 CPU 密集型任务的要求，或者存储/网络吞吐量响应。它提供了一些预配置调优配置文件，以增强性能并减少在很多特定用例中的能耗。编辑这些配置集或创建新配置集以创建根据您的环境定制的性能解决方案。

作为 **tuned** 的一部分提供的虚拟化相关配置集包括：

virtual-guest

基于 ***throughput-performance*** 配置集， ***virtual-guest*** 也会降低虚拟内存的交换性。

创建 Red Hat Enterprise Linux 7 客户机虚拟机时会自动选择 ***virtual-guest*** 配置集。它是虚拟机的建议配置集。

此配置集在红帽企业 Linux 6.3 及更新的版本中提供，但必须在安装虚拟机时手动选择。

virtual-host

基于 ***throughput-performance*** 配置集， ***virtual-host*** 还可提高脏页面的主动回写。这个配置集是虚拟化主机的推荐配置集，包括 KVM 和 Red Hat Virtualization(RHV)主机。

在 Red Hat Enterprise Linux 7 安装中，默认安装 tuned 软件包并启用 **tuned** 服务。

要列出所有可用配置集并确定当前活跃的配置集，请运行：

```
# 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
```

也可以创建 **自定义调优配置集** 来封装一组调优参数。有关创建 **自定义调优配置集** 的说明，请参阅 **tuned.conf** man page。

要只显示当前活跃的配置集，请运行：

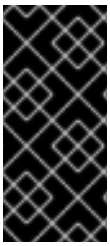
```
tuned-adm active
```

要切换到其中一个可用配置集，请运行：

```
tuned-adm profile profile_name
```

例如，要切换到 **virtual-host** 配置集，请运行：

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



重要

在 Red Hat Enterprise Linux 7.1 及更高版本中设置调优配置集后，确保为重启后要应用配置的配置集启用 **tuned** 服务：

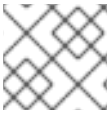
```
# systemctl enable tuned
```

在某些情况下，最好禁用 **tuned** 来使用手动设置的参数。要禁用当前会话的所有调整，请运行：

```
# tuned-adm off
```

要永久禁用 **tuned** 并恢复它执行的所有更改，请运行：

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



注意

有关 **tuned** 的详情，请查看 *Red Hat Enterprise Linux 7 性能调优指南*。

第 5 章 NETWORKING

本章论述了虚拟化环境的网络优化主题。

5.1. 网络调优提示

- 使用多个网络避免在单个网络上拥塞。例如，具有用于管理、备份或实时迁移的专用网络。
- 红帽建议不要在同一网络段中使用多个接口。但是，如果这是不可避免的，您可以使用 ***arp_filter*** 来防止 ARP Flux，一个不可取的情况在主机和客户机中发生，并由机器响应多个网络接口的 ARP 请求： **echo 1 > /proc/sys/net/ipv4/conf/all/arp_filter** 或 edit **/etc/sysctl.conf**.

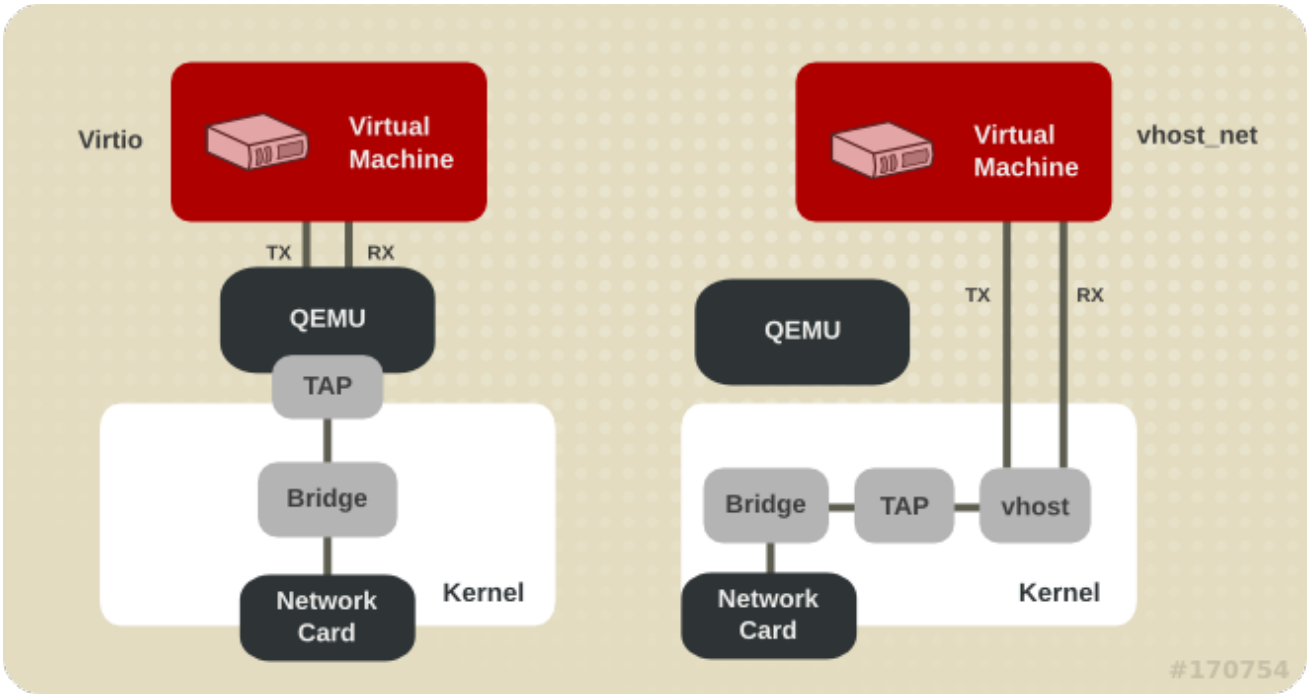
注意

有关 ARP Flux 的更多信息，请参见 http://linux-ip.net/html/ether-arp.html#ether-arp-flux

5.2. VIRTIO 和 VHOST_NET

下图显示了在 Virtio 和 vhost_net 构架中对内核的参与。

图 5.1. virtio 和 vhost_net 架构

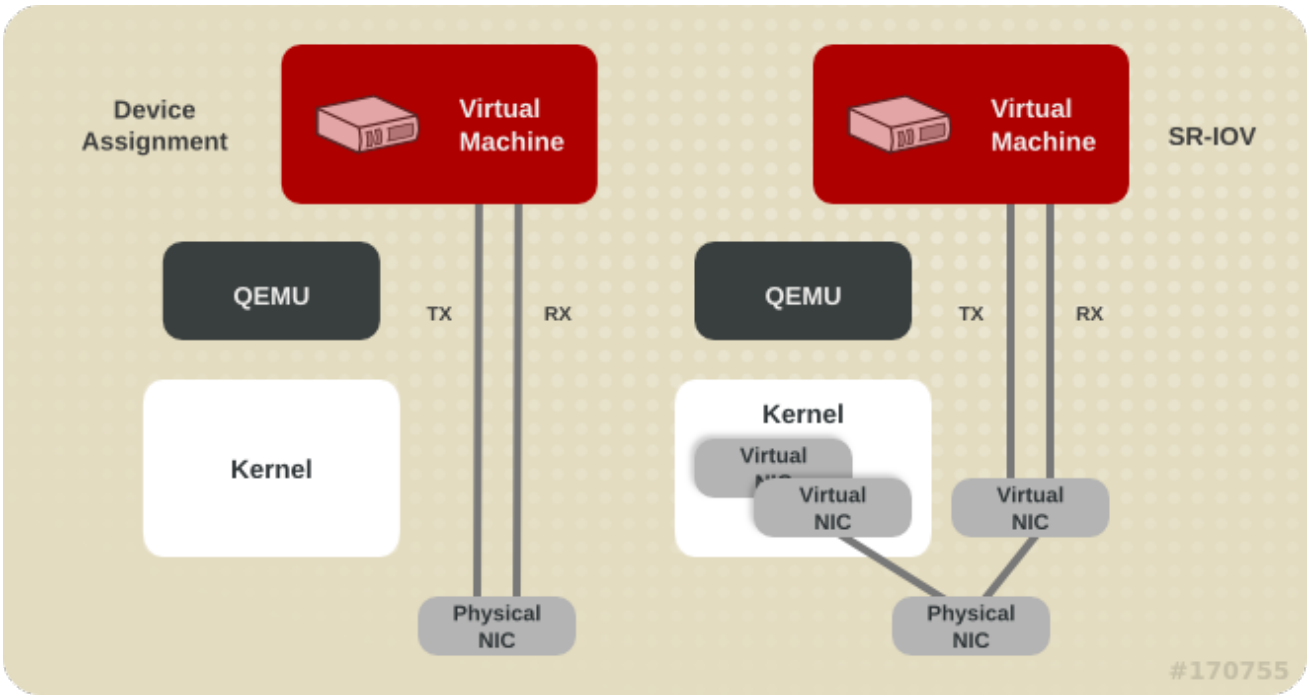


vhost_net 将 Virtio 驱动程序的一部分从用户空间移至内核。这可减少复制操作，从而减少延迟和 CPU 用量。

5.3. 设备分配和 SR-IOV

下图显示了在 Device Assignment 和 SR-IOV 架构中参与内核。

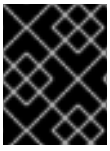
图 5.2. 设备分配和 SR-IOV



设备分配为客户机提供整个设备。SR-IOV 需要支持驱动程序和硬件，包括 NIC 和系统板，并允许创建多个虚拟设备并将其传递给不同的虚拟机。客户端需要特定于厂商的驱动程序，但 SR-IOV 提供了任何网络选项的最低延迟。

5.4. 网络调优技术

这部分论述了在虚拟环境中调优网络性能的技术。



重要

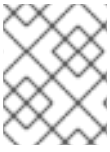
红帽企业 Linux 7 管理程序和虚拟机以及运行 Red Hat Enterprise Linux 6.6 及更高版本的虚拟机支持以下功能：

5.4.1. bridge Zero Copy Transmit

零复制传输模式对大型数据包大小有效。它通常在客户机网络和外部网络间传输大型数据包时，主机 CPU 开销降低了 15%，而不影响吞吐量。

它不会影响虚拟客户机、guest-to-host 或小数据包工作负载的性能。

在 Red Hat Enterprise Linux 7 虚拟机上完全支持桥接零副本传输，但默认禁用。要启用零复制传输模式，将 vhost_net 模块的 ***experimental_zcopytx*** 内核模块参数设置为 1。具体步骤请查看 虚拟化部署和管理指南。



注意

通常会在传输过程中创建额外的数据复制，作为威胁缓解技术，以防止拒绝服务以及信息泄露攻击。启用零复制会禁用此威胁的缓解方案技术。

如果观察性能回归，或者主机 CPU 使用率不是问题，可以通过将 ***experimental_zcopytx*** 设置为 0 来禁用零副本传输模式。

5.4.2. multi-Queue virtio-net

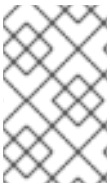
多队列 virtio-net 提供了一种在 vCPU 数量增加时扩展网络性能的方法，允许它们一次通过多个 virtqueue 对传输数据包。

当今的高端服务器具有更多处理器，在它们上运行的客户机通常会增加 vCPU 数量。在单个队列 virtio-net 中，客户机中协议堆栈的规模受到限制，因为网络性能不会随着 vCPU 数量增加而扩展。客户机无法并行传输或检索数据包，因为 virtio-net 只有一个 TX 和 RX 队列。

多队列支持通过允许并行数据包处理消除这些瓶颈。

多队列 virtio-net 在以下情况下，提供最高性能优势：

- 流量数据包相对较大。
- 客户机同时在许多连接上处于活动状态，客户机在客户机之间、客户机到主机或向外部系统运行的流量。
- 队列数量等于 vCPU 数量。这是因为多队列支持优化 RX 中断关联和 TX 队列选择，以便使特定的队列私有到特定的 vCPU。



注意

目前，设置多队列 virtio-net 连接会对传出流量的性能产生负面影响。具体来说，这可能会在传输控制协议(TCP)流上发送 1,500 字节的数据包时发生。如需更多信息，请参阅红帽知识库。

5.4.2.1. 配置多队列 virtio-net

要使用多队列 virtio-net，请在客户机 XML 配置中添加以下内容（其中 *N* 的值从 1 到 256，因为内核支持多队列 tap 设备）支持 256 个队列：

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

当在客户机中运行带有 *N* virtio-net 队列的虚拟机时，使用以下命令（ *M* 的值从 1 到 *N*）启用多队列支持：

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

5.5. 批量网络数据包

在传输大量路径的配置中，在将数据包提交到内核前，进行批处理数据包可能会提高缓存利用率。

要配置可以批处理的最大数据包数，其中 *N* 是要批处理的最大数据包数：

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

要为 type='bridge' 或 type='network' 接口提供对 **tun/tap** rx 批处理的支持，请向域 XML 文件添加类似如下的片段。

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

第 6 章 I/O 调度

您可以使用输入/输出(I/O)调度程序在 Red Hat Enterprise Linux 7 是 虚拟化主机 以及虚拟化 guest 时提高磁盘性能。

6.1. 使用 RED HAT ENTERPRISE LINUX 作为虚拟化主机进行 I/O 调度

当使用 Red Hat Enterprise Linux 7 作为虚拟机的主机时，默认的 **截止时间调度程序** 通常是理想的选择。这个调度程序几乎在所有工作负载上执行良好。

但是，如果最大化 I/O 吞吐量比降低客户机工作负载上的 I/O 延迟更重要，那么使用 **cfq** 调度程序可能会很有用。

6.2. 使用 RED HAT ENTERPRISE LINUX 作为虚拟机进行 I/O 调度

您可以在 Red Hat Enterprise Linux 客户机虚拟机上使用 I/O 调度，无论运行客户机的系统管理程序是什么。以下是应考虑的好处和问题列表：

- 红帽企业 Linux 虚拟机通常受益于使用 **noop** 调度程序。在向虚拟机监控程序发送 I/O 之前，调度程序将客户端操作系统中的小请求合并到较大的请求中。这可让管理程序更有效地处理 I/O 请求，这可显著改进虚拟机的 I/O 性能。
- 根据工作负载 I/O 以及连接存储设备的方式，**截止时间** 等调度程序可能比不透明地使用。红**帽建议对性能测试来验证哪个调度程序能提供最佳性能影响。**
- **使用 iSCSI、SR-IOV 或物理设备透传访问的客户机不应使用 noop 调度程序。这些方法不允许主机优化对底层物理设备的 I/O 请求。**

注意

在虚拟环境中，在主机和客户机层上调度 I/O 有时非常有用。如果多个客户端使用由主机操作系统管理的文件系统或块设备上的存储，主机可能会更有效地调度 I/O，因为它知道所有客户机的请求。另外，主机知道存储的物理布局，这可能不会线性映射到客户机的虚拟存储。

所有调度程序性能优化都应该在正常操作条件下进行测试，因为同构基准通常无法准确比较虚拟环境中使用共享资源的系统的性能。

6.2.1. 为 Red Hat Enterprise Linux 7 配置 I/O 调度程序

Red Hat Enterprise Linux 7 系统中使用的默认调度程序是 截止时间。但是，在 Red Hat Enterprise Linux 7 虚拟机中，通过执行以下操作将调度程序更改为 noop 会大：

1. **在 /etc/default/grub 文件中，将 GRUB_CMDLINE_LINUX 行上的 elevator=deadline 字符**

串更改为 **elevator=noop**。如果没有 **elevator=** 字符串，请在行末尾添加 **elevator=noop**。

以下显示了在成功更改后的 **/etc/default/grub** 文件。

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

2. 重建 **/boot/grub2/grub.cfg** 文件。

- 在基于 **BIOS** 的机器上：

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

- 在基于 **UEFI** 的机器上：

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

第 7 章 块 I/O

本章论述了在虚拟环境中优化 I/O 设置。

7.1. 块 I/O 调优

通过 virsh blkiotune 命令，管理员可以在客户机 XML 配置的 <blkio> 元素中手动设置或显示客户机虚拟机的块 I/O 参数。

显示虚拟机的当前 <blkio> 参数：

```
# virsh blkiotune virtual_machine
```

要设置虚拟机的 <blkio> 参数，请使用 virsh blkiotune 命令并根据您的环境替换选项值：

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

参数包括：

weight

I/O 权重，范围为 100 到 1000。

增加设备的 I/O 权重会增加设备的 I/O 带宽的优先级，从而为它提供更多主机资源。同样的，降低设备的权重可使其消耗较少的主机资源。

device-weights

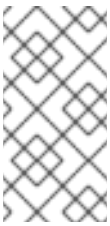
单个字符串列出一个或多个设备/加权对，格式为/path/to/device,weight,/path/to/device,weight。每个权重必须在范围 100-1000 中，或值 0 才能从每个设备列表中删除该设备。只有字符串中列出的设备才会被修改，对其它设备的现有每个设备权重保持不变。

config

添加 --config 选项以便在下次引导时生效。

live

添加 --live 选项，以将更改应用到正在运行的虚拟机。



注意

live 选项要求虚拟机监控程序支持此操作。并非所有虚拟机监控程序都允许实时更改最大内存限值。

current

添加 --current 选项，以将更改应用到当前虚拟机。

例如，以下将 *liftbrul* 虚拟机中的 /dev/sda 设备的权重改为 500。

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

注意

使用 virsh help blkiotune 命令获取有关使用 virsh blkiotune 命令的更多信息。

7.2. CACHING

可使用 virt-manager 在 guest 安装过程中配置缓存选项，或者通过编辑客户机 XML 配置在现有 guest 虚拟机上进行配置。

表 7.1. 缓存选项

缓存选项	描述
cache=none	客户机上的 I/O 不会缓存在主机上，但可能会保存在回写磁盘缓存中。这个选项用于具有较大 I/O 要求的虚拟机。这个选项通常是最佳选择，是支持迁移的唯一选择。

缓存选项	描述
Cache=writethrough	客户机上的 I/O 会缓存在主机上，但通过写入物理介质。这个模式速度较慢，且容易扩展问题。最适合用于具有较低 I/O 要求的少量虚拟机。建议在不需要迁移的情况下不支持回写缓存（如 Red Hat Enterprise Linux 5.5 及更早版本）的客户机。
Cache=writeback	客户机上的 I/O 会缓存在主机上。
Cache=directsync	与 ***writethrough*** 类似，但客户机中的 I/O 会绕过主机页面缓存。
Cache=unsafe	主机可能会缓存所有磁盘 I/O，并忽略来自 guest 的同步请求。
cache=default	如果没有指定缓存模式，则会选择系统的默认设置。

在 virt-manager 中，可在 虚拟磁盘下指定缓存模式。有关使用 virt-manager 更改缓存模式的详情，请参考 第 3.3 节 “虚拟磁盘性能选项”

要在客户机 XML 中配置缓存模式，请编辑 cache 标签中的 driver 设置以指定缓存选项。例如，要将缓存设置为 *writeback* ：

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

7.3. I/O 模式

在 guest 安装过程中，可以使用 virt-manager 配置 I/O 模式选项，或者通过编辑客户机 XML 配置在现有 guest 虚拟机上进行配置。

表 7.2. IO 模式选项

IO 模式选项	描述
IO=native	Red Hat Virtualization(RHV)环境的默认环境。这个模式指的是具有直接 I/O 选项的内核异步 I/O。
IO=threads	默认为基于主机用户模式的线程。

IO 模式选项	描述
IO=default	Red Hat Enterprise Linux 7 中的默认设置是线程模式。

在 virt-manager 中，可在 虚拟磁盘 下指定 I/O 模式。有关使用 virt-manager 更改 I/O 模式的详情，请参考 第 3.3 节 “虚拟磁盘性能选项”

要在客户机 XML 中配置 I/O 模式，请编辑 io 标签中的 driver 设置，指定 *native*、*threads* 或 *default*。例如，要将 I/O 模式设置为 *threads* ：

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

7.4. 块 I/O 调优技术

本节介绍了在虚拟环境中调优块 I/O 性能的更多技术。

7.4.1. 磁盘 I/O Throttling

当多个虚拟机同时运行时，它们可以通过过度磁盘 I/O 影响到系统性能。KVM 中的磁盘 I/O 节流提供了设置从虚拟机发送到主机机器的磁盘 I/O 请求的限制。这可阻止虚拟机过度利用共享资源并影响其他虚拟机的性能。

在各种情况下，磁盘 I/O 节流非常有用，例如，属于不同客户的客户机虚拟机在同一主机上运行，或者为不同的客户机提供服务质量保障时。磁盘 I/O 节流还可用来模拟较慢的磁盘。

I/O 节流可以独立于附加到客户机的每个块设备应用，并支持对吞吐量和 I/O 操作的限制。使用 virsh blkdeviotune 命令为虚拟机设置 I/O 限制：

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

device 为附加到虚拟机的一个磁盘设备指定唯一的目标名称(<target dev='name'/>)或源文件(<source file='name'/>)。*使用 virsh domblklist 命令输入磁盘设备名称列表。*

可选参数包括：

total-bytes-sec

吞吐量总吞吐量上限（以字节/秒为单位）。

read-bytes-sec

每秒的读取吞吐量限制（以字节为单位）。

write-bytes-sec

每秒写入吞吐量限制（以字节为单位）。

total-iops-sec

每秒总 I/O 操作限制。

read-iops-sec

每秒读取 I/O 操作限制。

write-iops-sec

每秒写入 I/O 操作限制。

例如，要将 virtual_machine 上的 vda 节流为每秒 1000 个 I/O 操作，每秒 50 MB，运行这个命令：

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

7.4.2. multi-Queue virtio-scsi

多队列 virtio-scsi 在 virtio-scsi 驱动程序中提供了改进的存储性能和可扩展性。它可让每个虚拟 CPU 有一个单独的队列和中断，以便在不影响其他 vCPU 的情况下使用。

7.4.2.1. 配置多队列 virtio-scsi

在 Red Hat Enterprise Linux 7 中默认禁用多队列 virtio-scsi。

要在客户机中启用多队列 virtio-scsi 支持，请在客户机 XML 配置中添加以下内容，其中 N 是 vCPU 队列总数 ：

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

第 8 章 MEMORY

本章论述了虚拟化环境的内存优化选项。

8.1. 内存调整提示

要优化虚拟化环境中的内存性能，请考虑以下信息：

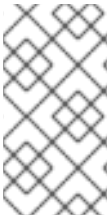
- 不要将更多资源分配给 guest，超过其使用的资源。
- 如果可能，将客户机分配到一个 NUMA 节点，为该 NUMA 节点上提供资源就足够。有关使用 NUMA 的更多信息，请参阅 第 9 章 NUMA。

8.2. 虚拟机的内存调整

8.2.1. 内存监控工具

内存用量可以使用裸机环境中使用的工具监控虚拟机。有助于监控内存用量和诊断内存相关问题的工具包括：

- top
- vmstat
- numastat
- /proc/



注意

有关使用这些性能工具的详情，请查看 **Red Hat Enterprise Linux 7 性能调优指南** 和 **man page**。

8.2.2. 使用 virsh 进行内存调优

guest XML 配置中的可选 **<memtune>** 元素允许管理员手动配置客户机虚拟机内存设置。如果省略 **<memtune>**，则虚拟机会根据在虚拟机创建过程中如何分配和分配的内存。

使用 **virsh memtune** 命令显示或设置虚拟机 **<memtune>** 元素中的内存参数，根据您的环境替换值：

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

可选参数 包括：

hard_limit

虚拟机可以使用的最大内存，单位为 **kibibytes**（1024 字节块）。

警告

设置此限制太低，可能会导致虚拟机被内核终止。

soft_limit

要在内存争用期间强制实施的内存限值，以 **kibibytes**（1024 字节的块）。

swap_hard_limit

最大内存加上交换虚拟机可以使用的最大内存，单位为 **kibibytes**（1024 字节的块）。**swap_hard_limit** 值必须大于 **hard_limit** 值。

min_guarantee

保证虚拟机的最小内存分配量（单位为 1024 字节）。

注意

有关使用 virsh memtune 命令的详情，请参考 # virsh help memtune。

可选的 <memoryBacking> 元素可以包含多个影响主机页面支持虚拟内存页的元素。

设置 locked 可防止主机交换属于客户机的内存页面。将以下内容添加到客户机 XML 中，以锁定主机内存中的虚拟内存页面：

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

重要

设置 locked 时，必须在 hard_limit 元素中将 <memtune> 设置为为客户机配置的最大内存，以及 进程本身消耗的内存。

设置 nosharepages 可防止主机合并客户机间使用的相同内存。要指示管理程序禁用客户机的共享页面，请将以下内容添加到客户机的 XML 中：

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

8.2.3. 大内存页和透明大内存页

AMD64 和 Intel 64 CPU 通常在 4kB 页面中解决内存，但它们能够使用更大的 2MB 或 1GB 页面，称为 巨页。KVM 客户机可以使用巨页内存支持进行部署，从而通过根据 transaction Lookaside Buffer (TLB)增加 CPU 缓存点击来提高性能。

在 Red Hat Enterprise Linux 7 中默认启用的内核功能，巨页可能会显著提高性能，特别是大型内存和内存密集型工作负载。通过通过使用巨页，Red Hat Enterprise Linux 7 可以更有效地管理大量内存。为了提高管理巨页的有效性和便利，Red Hat Enterprise Linux 7 默认使用 Transparent Huge Pages (THP)。有关巨页和 THP 的更多信息，请参阅 性能调优指南。

Red Hat Enterprise Linux 7 系统支持 2MB 和 1GB 巨页，它们可在引导时或运行时分配。有关启用多个巨页大小的步骤，请参阅 第 8.2.3.3 节 “在引导时或运行时为客户机启用 1 GB 巨页”。

8.2.3.1. 配置 THG

透明大内存页(THP)是一种抽象层，可自动化创建、管理和使用巨页的大多数方面。默认情况下，它们自动优化系统设置以提高性能。

注意

使用 KSM 可以减少出现透明大内存页，因此建议在启用 THP 前禁用 KSM。更多信息请参阅 第 8.3.4 节 “取消激活 KSM”。

默认启用透明大内存页。要检查当前状态，请运行：

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

要启用默认使用透明大内存页，请运行：

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

这会将 /sys/kernel/mm/transparent_hugepage/enabled 设置为 always。

禁用透明大内存页 ：

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

透明大内存页支持不会阻止使用静态巨页。但是，如果没有使用静态巨页，KVM 将使用透明的巨页，而不是常规的 4kB 页面大小。

8.2.3.2. 配置静态大页

在某些情况下，最好对巨页进行更大的控制。 要在客户机上使用静态巨页，请使用 virsh edit 将以下内容添加到 guest XML 配置中：

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

这指示主机使用大页面将内存分配到虚拟客户机，而不使用默认的页面大小。

运行以下命令来查看当前的巨页值：

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

过程 8.1. 设置巨页

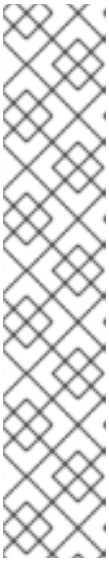
以下示例步骤显示了用于设置巨页的命令。

1. 查看当前的巨页值：

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

2. 巨页的增量为 2MB。要将巨页数量设置为 25000，请使用以下命令：

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



注意

要使设置持久，请在客户端机器上的 **/etc/sysctl.conf** 文件中添加以下行，**X** 是预期的巨页数：

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

之后，通过将 **transparent_hugepage=never** 添加到客户端 **/etc/grub2.cfg** 文件中的 **/kernel** 行的末尾，为内核引导参数添加 **transparent_hugepage=never**。

3. 挂载巨页：

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

4. 将以下几行添加到虚拟机 **XML** 配置的 **memoryBacking** 部分：

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

5. 重启 **libvirtd** ：

```
# systemctl restart libvirtd
```

6.

- 启动虚拟机：

```
# virsh start virtual_machine
```

- 如果虚拟机已在运行，重启它：

```
# virsh reset virtual_machine
```

7. 验证 /proc/meminfo 中的更改：

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

巨页不仅可能有益于主机，而且客户机总数必须小于主机中的可用值。

8.2.3.3. 在引导时或运行时为客户机启用 1 GB 巨页

Red Hat Enterprise Linux 7 系统支持 2MB 和 1GB 巨页，它们可在引导时或运行时分配。

过程 8.2. 在引导时分配 1GB 巨页

1. 要在引导时分配巨页大小，请使用以下命令来指定巨页数量。这个示例分配 4 个 1GB 巨页和 1024 2MB 巨页：

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

更改此命令行，以指定要在启动时分配的不同数量大页面。

注意

然后，在您在引导时分配 1GB 巨页时，还必须完成接下来的两个步骤。

2. **在主机上挂载 2MB 和 1GB 巨页：**

```
# 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. **将以下几行添加到虚拟机 XML 配置的 memoryBacking 部分：**

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

4. **重启 libvirtd 以在客户机上使用 1GB 巨页：**

```
# systemctl restart libvirtd
```

过程 8.3. 在运行时分配 1GB 巨页

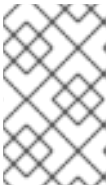
1GB 巨页也可以在运行时分配。通过运行时分配，系统管理员可以选择从哪个 NUMA 节点分配这些页面。但是，由于内存碎片，运行时页面分配可能会比引导时间分配失败更容易。

1. **要在运行时分配巨页的不同大小，请使用以下命令替换巨页数的值、NUMA 节点从中分配它们，以及巨页大小：**

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

这个示例从 node1 分配 4 个 1GB 巨页，并从 node3 分配 1024 2MB 巨页。

可使用上述命令随时更改这些巨页设置，具体取决于主机系统上的可用内存量。



注意

在运行时，您第一次分配 1GB 巨页时，还必须完成接下来的两个步骤。

2. 在主机上挂载 2MB 和 1GB 巨页：

```
# 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. 将以下几行添加到虚拟机 XML 配置的 memoryBacking 部分：

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

4. 重启 libvirtd 以在客户机上使用 1GB 巨页：

```
# systemctl restart libvirtd
```

8.3. 内核同页合并(KSM)

内核相同的页面合并(KSM)由 KVM 管理程序使用，允许 KVM 客户机共享相同的内存页。这些共享页面通常是常见的库或其他相同、高使用的数据。KSM 通过避免内存重复，可以增加相同或类似客户机操作系统的客户机密度。

共享内存的概念在现代操作系统中很常见。例如，在程序首次启动时，它会与父程序共享其所有内存。当子程序尝试修改此内存时，内核会分配一个新的内存区域，复制原始内容并允许程序修改这个新区域。这被称为 write（写时）的 copy。

KSM 是以相反方式使用此概念的 Linux 功能。KSM 使内核能够检查两个或多个已运行程序并比较它们的内存。如果任何内存区域或页面相同，则 KSM 会将多个相同内存页减至单个页面。然后，这个页会在 write 上标记为副本。如果 guest 虚拟机修改了页面的内容，则会为该 guest 创建一个新页面。

这对于使用 KVM 进行虚拟化非常有用。当客户机虚拟机启动时，它只会从主机 qemu-kvm 进程中继承内存。客户端运行后，客户机操作系统镜像的内容可在客户机运行相同的操作系统或应用程序时共享。

KSM 允许 KVM 请求这些相同客户机内存区域已经共享。

KSM 提供增强的内存速度和利用率。通过 KSM，通用进程数据存储在缓存或主内存中。这可减少 KVM 客户机的缓存丢失，这可以提高某些应用程序和操作系统的性能。其次，共享内存可减少客户机的总内存用量，从而可以增加密度并增加资源的利用率。

注意

在 Red Hat Enterprise Linux 7 中，KSM 是 NUMA 感知功能。这样，它在合并页面时可以考虑 NUMA 本地性，从而避免与将页面移动到远程节点中的页面相关的性能下降。红帽建议在使用 KSM 时避免多节点内存合并。如果使用 KSM，将 /sys/kernel/mm/ksm/merge_across_nodes 可调项改为 0，以避免在 NUMA 节点间合并页面。这可以通过 virsh node-memory-tune --shm-merge-across-nodes 0 命令实现。内核内存核算统计最终可在大量跨节点合并后相互冲突。因此，numad 在 KSM 守护进程合并大量内存后可能会变得混乱。如果您的系统有大量可用内存，可以通过关闭和禁用 KSM 守护进程来获得更高的性能。有关 NUMA 的更多信息，请参阅 第 9 章 NUMA"。

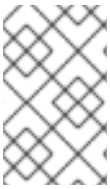
重要

确定 swap 大小足以用于提交的 RAM，即使不考虑 KSM。KSM 可减少相同或类似虚拟机的 RAM 使用量。可能会在没有足够 swap 空间的情况下使用 KSM 过量使用客户机，但不建议使用 guest 虚拟机内存使用可能会导致页面变得不共享。

Red Hat Enterprise Linux 使用两种单独的方法来控制 KSM：

- ksm 服务启动并停止 KSM 内核线程。
- ksmtuned 服务控制并调整 ksm 服务，动态管理同一页面合并。ksmtuned 启动 ksm 服务，并在不需要内存共享时停止 ksm 服务。当创建或销毁新 guest 时，ksmtuned 必须使用要运行的 retune 参数来指示。

这两个服务都使用标准服务管理工具进行控制。



注意

默认情况下，Red Hat Enterprise Linux 6.7 中会关闭 KSM。

8.3.1. KSM 服务

- ksm 服务包括在 qemu-kvm 软件包中。

- 当 ksm 服务没有启动时，内核相同页面合并(KSM)只共享 2000 个页面。此默认值提供有限的内存节省优势。

- 启动 ksm 服务后，KSM 将共享到主机系统主内存的一半。启动 ksm 服务，使 KSM 能够共享更多内存。

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

ksm 服务可以添加到默认启动序列中。使用 systemctl 命令使 ksm 服务持久。

```
# systemctl enable ksm
```

8.3.2. KSM 调整服务

ksmtuned 服务通过循环和调整 ksm，对内核相同的页面合并(KSM)配置进行微调。另外，当创建虚拟机或销毁时，libvirt 会通知 ksmtuned 服务。ksmtuned 服务没有选项。

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

可以使用 retune 参数调优 ksmtuned 服务，该参数指示 ksmtuned 手动运行调优功能。

/etc/ksmtuned.conf 文件是 ksmtuned 服务的配置文件。下面的文件输出是默认的 ksmtuned.conf 文件：

```
# 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
```

在 /etc/ksmtuned.conf 文件中，npages 设置 ksm 在 ksm d 守护进程变为 inactive 之前将扫描多少页。这个值也会在 /sys/kernel/mm/ksm/pages_to_scan 文件中设置。

KSM_THRES_CONST 值代表激活 ksm 的可用内存量。如果发生以下情况之一，则激活 ksmd ：

- 可用内存量低于阈值，在 KSM_THRES_CONST 中设置。
- 所提交的内存量加上阈值 KSM_THRES_CONST 超过总内存量。

8.3.3. KSM 变量和监控

内核相同页面合并(KSM)将监控数据存储在 /sys/kernel/mm/ksm/ 目录中。这个目录中的文件由内核更新，是 KSM 使用和统计数据的准确记录。

以下列表中的变量也是 /etc/ksmtuned.conf 文件中的可配置变量，如上述所述。

/sys/kernel/mm/ksm/ 中的文件：

full_scans

完整扫描运行.

merge_across_nodes

是否可以合并来自不同 NUMA 节点的页面。

pages_shared

共享的总页面。

pages_sharing

页当前共享。

pages_to_scan

页面未扫描。

pages_unshared

页面不再共享。

pages_volatile

易失性页面的数量。

run

KSM 进程是否在运行。

sleep_millisecs

sleep 毫秒.

可以使用 virsh node-memory-tune 命令手动调整这些变量。例如，下面指定了在共享内存服务进入休眠前要扫描的页数：

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

如果 DEBUG=1 行添加到 /etc/ksmtuned.conf 文件中，则 KSM 调整活动将存储在 /var/log/ksmtuned 日志文件中。日志文件位置可使用 LOGFILE 参数更改。不建议更改日志文件位置，可能需要特殊配置 SELinux 设置。

8.3.4. 取消激活 KSM

内核相同页面合并(KSM)的性能开销可能会对某些环境或主机系统而言过大。KSM 还可以引入侧信道，这些通道可能会用于跨客户机泄漏信息。如果出现这种情况，可以基于每个虚拟机禁用 KSM。

通过停止 ksmtuned 和 ksm 服务可停用 KSM。但是，重新启动后此操作不会保留。要取消激活 KSM，以 root 用户身份在终端中运行以下命令：

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

停止 ksmtuned 和 ksm 会取消激活 KSM，但重新启动后该操作不会保留。使用 systemctl 命令永久取消激活 KSM：

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

当 KSM 禁用时，在激活 KSM 之前共享的所有内存页面仍会共享。要删除系统中的所有 PageKSM，请使用以下命令：

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

执行后，khugepaged 守护进程可以在 KVM 客户机物理内存上重建透明巨页。使用 # echo 0 >/sys/kernel/mm/ksm/run 会停止 KSM，但不会共享之前创建的所有 KSM 页面（这与 # systemctl stop ksmtuned 命令相同）。

第 9 章 NUMA

过去，AMD64 和 Intel 64 系统中的所有内存都可被所有 CPU 平等地访问。无论 CPU 执行操作，已知一致性内存访问(UMA)的访问时间都是一样的。

最近 AMD64 和 Intel 64 处理器时，这个行为不再是这种情况。在非一致性内存访问(NUMA)中，系统内存被分成多个 NUMA 节点，对应于套接字或一组对系统内存本地子集相同的访问延迟的特定 CPU。

本章论述了虚拟环境中的内存分配和 NUMA 调优配置。

9.1. NUMA 内存分配策略

以下策略定义了如何从系统的节点分配内存：

Strict

严格策略意味着，如果无法在目标节点上分配内存，分配将失败。

指定 NUMA nodeset 列表，而不定义内存模式属性默认为 strict 模式。

Interleave

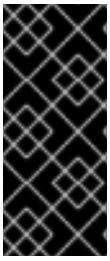
内存页面会在由节点设置指定的节点上分配，但以轮循方式分配。

Preferred

内存从单个首选内存节点分配。如果没有足够的内存可用，可以从其他节点分配内存。

要启用预期的策略，将其设置为域 XML 文件的 &lt ;memory mode > 元素的值：

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



重要

如果以 strict 模式过量使用内存，且客户机没有足够的 swap 空间，内核会终止一些客户机进程来检索额外的内存。红帽建议使用 preferred 分配并指定单个节点集（例如，nodeset='0'）以防止这种情况。

9.2. 自动 NUMA BALANCING

自动 NUMA 平衡提高了在 NUMA 硬件系统上运行的应用程序的性能。它在 Red Hat Enterprise Linux 7 系统中默认启用。

当进程线程正在调度到与线程相同的 NUMA 节点上时，应用程序通常会尽力执行最佳操作。自动 NUMA 平衡移动任务（可以是线程或进程）更接近他们访问的内存。它还将应用程序数据移动到更接近引用它的任务中。这在自动 NUMA 平衡激活时由内核自动完成。

自动 NUMA 平衡使用多种算法和数据结构，它们只有在系统中有自动 NUMA 平衡活跃时才激活并分配：

- 定期进程内存的 NUMA 取消映射
- NUMA 提示错误
- migrate-on-Fault(MoF)- 将内存移动到程序运行的位置
- task_numa_placement - 移动运行程序更接近其内存

9.2.1. 配置自动 NUMA Balancing

在 Red Hat Enterprise Linux 7 中默认启用自动 NUMA 平衡，并在使用 NUMA 属性引导时自动激活。

满足以下任一条件时启用自动 NUMA 平衡：

- # numactl --hardware 显示多个节点

- # cat /proc/sys/kernel/numa_balancing shows 1

应用程序的手动 NUMA 调优将覆盖自动 NUMA 平衡，禁用定期不映射内存、NUMA 错误、迁移和自动 NUMA 放置。

在某些情况下，首选使用系统范围的手动 NUMA 调整。

要禁用自动 NUMA 平衡，请使用以下命令：

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

要启用自动 NUMA 平衡，请使用以下命令：

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

9.3. LIBVIRT NUMA 调整

通常，在 NUMA 系统上获得最佳性能是通过将客户机大小限制为单一 NUMA 节点上的资源量来实现的。 避免在 NUMA 节点间不必要地分割资源。

使用 numastat 工具查看进程和操作系统的每 NUMA 节点内存统计信息。

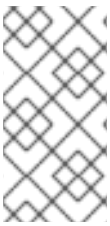
在以下示例中，numastat 工具显示了四个虚拟机，并在 NUMA 节点之间达到低效内存对齐：

```
# 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
```

运行 **numad** 以自动匹配客户机的 **CPU** 和内存资源。

然后再次运行 **numastat -c qemu-kvm** 来查看正在运行的 **numad** 的结果。以下输出显示了资源已对齐：

```
# 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
```



注意

使用 **-c** 运行 **numastat** 提供了紧凑输出；添加 **-m** 选项会在各个节点基础上添加系统范围的内存信息。详情请查看 **numastat man page**。

9.3.1. 每个主机 NUMA 节点监控内存

您可以使用 **nodestats.py** 脚本报告主机上的每个 **NUMA** 节点的总内存和可用内存。此脚本还报告每个正在运行的域都严格绑定到某些主机节点的内存量。例如：

```
# /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
```

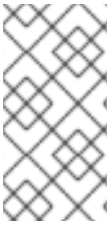
> Node 1: 1024 MiB nodes 1
> Node 2: 1024 MiB nodes 2
> Node 3: 1024 MiB nodes 3

这个示例显示了四个主机 NUMA 节点，每个节点总共包含 4GB 的 RAM(MemTotal)。接近所有的内存都会在每个域中消耗(MemFree)。正在运行的四个域（虚拟机）：域 'rhel7-0' 具有 1.5GB 内存，它不固定到任何特定主机 NUMA 节点上。但是，域 'rhel7-2' 具有 4GB 内存和 4 个 NUMA 节点，这些节点固定为 1:1 到主机节点。

要打印主机 NUMA 节点统计信息，请为您的环境创建一个 nodestats.py 脚本。示例脚本可在 /usr/share/doc/libvirt-python-版本/examples/nodestats.py 中找到 libvirt-python 软件包文件。可以使用 rpm -ql libvirt-python 命令显示到脚本的特定路径。

9.3.2. NUMA vCPU 固定

vCPU 固定提供与在裸机系统上任务固定的类似优势。由于 vCPU 作为主机操作系统上的用户空间任务运行，固定可提高缓存效率。其中一个例子是，所有 vCPU 线程都在同一个物理插槽中运行，因此共享 L3 缓存域。

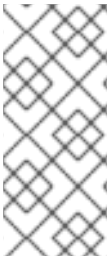


注意

在 Red Hat Enterprise Linux 7.0 到 7.2 中，只能获得活跃的 vCPU。但是，在 Red Hat Enterprise Linux 7.3 中，还提供固定不活跃的 vCPU。

将 vCPU 固定与 numatune 相结合可以避免 NUMA 未命中。NUMA 丢失的性能影响非常重要，通常从 10% 性能命中启动或更高。应将 vCPU 固定和 numatune 配置在一起。

如果虚拟机正在执行存储或网络 I/O 任务，则将所有 vCPU 和内存固定到物理连接到 I/O 适配器的同一物理套接字是有益的。



注意

lstopo 工具可用于视觉化 NUMA 拓扑。它还有助于验证 vCPU 是否已绑定到同一物理套接字上的核心。有关 lstopo 的更多信息，请参见以下知识库文章https://access.redhat.com/site/solutions/62879。

重要

固定导致存在比物理内核更多的 vCPU 数量 增加了复杂性。

以下示例 XML 配置中有一个域进程固定到物理 CPU 0-7。vCPU 线程固定为自己的 cpuset。例如，vCPU0 固定到物理 CPU 0，vCPU1 固定到物理 CPU 1，以此类推：

```
<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 和 vcpupin 标签之间有一个直接的关系。如果没有指定 vcpupin 选项，则该值将自动确定并继承自父 vcpu 标签选项。以下配置显示了缺少 vcpu 5 的 <vcpupin>。因此，vCPU5 会固定到物理 CPU 0-7，如父标签 <vcpu> 中指定的：

```
<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

<vcpupin>、<numatune> 和 <emulatorpin> 应该配置为实现最佳、确定的性能。有关 <numatune> 标签的详情请参考 第 9.3.3 节 “域进程”。有关 <emulatorpin> 标签的详情请参考 第 9.3.6 节 “使用 emulatorpin”。

9.3.3. 域进程

如 Red Hat Enterprise Linux 中提供的，libvirt 使用 libnuma 为域进程设置内存绑定策略。这些策略的 nodeset 可以配置为 静态（在域 XML 中指定）或 自动 （通过查询 numad 进行配置）。有关如何在

<numatune> 标签中配置它们的示例，请参阅以下 XML 配置：

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

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

libvirt 使用 sched_setaffinity(2) 为域进程设置 CPU 绑定策略。cpuset 选项可以 是静态的（在域 XML 中指定）或 自动 （通过查询 numad 进行配置）。有关如何在 <vcpu> 标签中配置它们的示例，请参阅以下 XML 配置：

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

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

您用于 <vcpu> 和 <numatune> 的放置模式间有隐式继承规则：

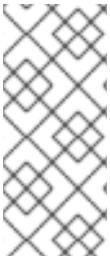
- **<numatune> 的放置模式默认为 <vcpu> 相同的放置模式，如果指定了 <nodeset>，则为 static。**
- **同样，<vcpu> 的放置模式默认为 <numatune> 的同一放置模式，如果指定了 <cpuset>，则为 static。**

这意味着，可以单独指定和定义域的 CPU 调优和内存调优，但也可以将其配置为依赖于其他的放置模式。

还可以使用 numad 配置您的系统，以引导选定的 vCPU 数量，而无需在启动时固定所有 vCPU。

例如：要在有 32 个 vCPU 的系统中只启用 8 个 vCPU，请配置类似如下的 XML：

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



注意

如需有关 vcpu 和 numatune 的更多信息，请参阅以下 URL，以及 http://libvirt.org/formatdomain.html#elementsCPUAllocation http://libvirt.org/formatdomain.html#elementsNUMATuning

9.3.4. 域 vCPU 线程

除了调优域进程外，libvirt 还支持 XML 配置中每个 vcpu 线程的固定策略设置。在 <cputune> 标签内为每个 vcpu 线程设置固定策略：

```
<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>
```

在此标签中，libvirt 使用 cgroup 或 sched_setaffinity(2) 将 vcpu 线程固定到指定的 cpuset。

注意

有关 <cputune> 的详情，请查看以下 URL：http://libvirt.org/formatdomain.html#elementsCPUTuning

另外，如果您需要设置一个 vCPU 多于单个 NUMA 节点的虚拟机，请配置主机，以便客户机能够检测到主机上的 NUMA 拓扑。这允许 1:1 个 CPU、内存和 NUMA 节点的映射。例如，这可以与具有 4 个 vCPU 和 6 GB 内存的客户机应用，以及具有以下 NUMA 设置的主机：

```
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
```

在这种情况下，使用以下域 XML 设置：

```
<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>
```

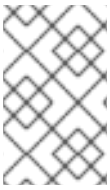
9.3.5. 使用缓存分配技术提高性能

您可以使用内核在特定 CPU 模型中提供的缓存分配技术(CAT)。这可为 vCPU 线程启用主机 CPU 缓存的一部分，从而提高了实时性能。

有关如何在 cachetune 标签中配置 vCPU 缓存分配的示例，请参阅以下 XML 配置：

```
<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>
```

上面的 XML 文件将 vCPU 0 和 1 的线程配置为分配的第一个 L3 缓存(level='3' id='0')中 3 MiB，一次为 L3CODE，一次为 L3DATA。



注意

单个虚拟机可以有多个 <cachetune> 元素。

如需更多信息，请参阅上游 libvirt 文档 中的 cachetune。

9.3.6. 使用 emulatorpin

调优域进程固定策略的另一种方式是在 <emulatorpin> 中使用 <cputune> 标签。

<emulatorpin> 标签指定 仿真程序 （并非包括 vCPU）的仿真程序（不包括 vCPU）的物理

CPU。<emulatorpin> 标签提供了一种精确的关联性到仿真程序线程进程的方法。因此， vhost 线程在物理 CPU 和内存的同一子集中运行，因此可以受益于缓存。 例如：

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

注意

在 Red Hat Enterprise Linux 7 中，默认启用自动 NUMA 平衡。自动 NUMA 平衡减少了手动调优 <emulatorpin> 的需求，因为 vhost-net 模拟器线程会更可靠地遵循 vCPU 任务。有关自动 NUMA 平衡的详情，请参考 第 9.2 节 “自动 NUMA Balancing”。

9.3.7. 使用 virsh 调整 vCPU 固定

重要

这些仅是示例命令。您需要根据您的环境替换值。

以下示例 virsh 命令将把一个 ID 为 1 的 vcpu 线程 rhel7 固定到物理 CPU 2：

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

您还可以使用 virsh 命令获取当前的 vcpu 固定配置。例如：

```
% virsh vcpupin rhel7
```

9.3.8. 使用 virsh 调整域进程 CPU 固定

重要

这些仅是示例命令。您需要根据您的环境替换值。

模拟器列表 选项将 CPU 关联性设置应用到与每个域进程关联的线程。要进行完整的固定，您必须为每个客户机同时使用 virsh vcpupin （如前面所示）和 virsh 模拟器。例如：

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

9.3.9. 使用 virsh 调整域进程内存策略

可以动态调整域进程内存。请参见以下示例命令：

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

可以在 virsh man page 中找到这些命令。

9.3.10. 客户机 NUMA 拓扑

可以使用客户机虚拟机 XML 中的 <numa> 标签中的 <cpu> 标签来指定客户机 NUMA 拓扑。请查看以下示例，并相应地替换值：

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

每个 <cell> 元素指定 NUMA 单元或 NUMA 节点。cpus 指定作为节点一部分的 CPU 或 CPU 范围，memory 以 kibibytes（1024 字节的块）指定节点内存。从 0 开始，为每个单元或节点分配一个 cellid 或 nodeid。

重要

当修改具有配置有 CPU 插槽、内核和线程拓扑的客户机虚拟机的 NUMA 拓扑时，请确保将属于单一插槽的内核和线程分配到相同的 NUMA 节点。如果为不同的 NUMA 节点分配了来自同一插槽的线程或内核，客户机可能无法引导。

警告

在 Red Hat Enterprise Linux 7 中，不支持将客户机 NUMA 拓扑与 巨页 一起使用，且仅在 Red Hat Virtualization 或 Red Hat OpenStack Platform 等分层产品中可用。

9.3.11. PCI 设备的 NUMA 节点位置

在启动新的虚拟机时，务必要知道主机 NUMA 拓扑和 PCI 设备与 NUMA 节点的关系，以便在请求 PCI 透传时，客户端会固定到正确的 NUMA 节点，以获得最佳性能。

例如：如果客户机固定到 NUMA 节点 0-1，则它的 PCI 设备之一与节点 2 的关系，则节点之间的数据传输会需要一些时间。

在 Red Hat Enterprise Linux 7.1 及更高版本中，libvirt 报告客户机 XML 中 PCI 设备的 NUMA 节点本地状态，启用管理应用程序以做出更好的性能决策。

此信息在 /sys/devices/pci*/*/numa_node 中的 sysfs 文件中可见。验证这些设置的一种方法是使用 lstopo 工具报告 sysfs 数据：

```
# 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"
```

此输出显示：

- NIC em* 和磁盘 sd* 连接到 NUMA 节点 0 和内核 0,2,4,6,8,10,12,14。
- NIC p1* 和 ib* 连接到 NUMA 节点 1 和内核 1,3,5,7,9,11,13,15。

9.4. NUMA-AWARE 内核同页合并(KSM)

内核同页合并(KSM)允许虚拟机共享相同的内存页。KSM 可以检测系统正在使用 NUMA 内存并控制不同 NUMA 节点之间合并页面。

使用 sysfs /sys/kernel/mm/ksm/merge_across_nodes 参数控制不同 NUMA 节点的页面合并。默认

情况下，所有节点都中的页面可以合并在一起。当此参数设置为零时，只有来自同一节点的页面会被合并。

通常，除非您被超过系统内存，通过禁用 KSM 共享来获得更好的运行时性能。

重要

当 KSM 在带有多个客户机虚拟机的 NUMA 主机上合并时，客户机和 CPU 可能会显著增加合并的 KSM 页面的访问延迟。

要指示管理程序禁用客户机的共享页面，请将以下内容添加到客户机的 XML 中：

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

有关使用 <memoryBacking> 元素调整内存设置的详情，请参考 第 8.2.2 节 “使用 virsh 进行内存调优”。

附录 A. 修订历史记录

修订 1.0-35 7.7 Beta 发行版本的版本	Thus May 23 2019	Jiri Herrmann
修订 1.0-34 7.6 GA 发行版本的版本	Tue Oct 25 2018	Jiri Herrmann
修订 1.0-32 7.6 Beta 发行版本的版本	Tue Aug 14 2018	Jiri Herrmann
修订 1.0-31 7.5 GA 发行版本的版本	Wed Apr 4 2018	Jiri Herrmann
修订 1.0-27 7.4 GA 发布的版本	Mon Jul 27 2017	Jiri Herrmann
修订 1.0-24 7.3 GA 发行版本的版本	Mon Oct 17 2016	Jiri Herrmann
修订 1.0-22 重新发布指南以修复几个程序错误	Mon Dec 21 2015	Laura Novich
修订 1.0-19 清理修订记录	Thu Oct 08 2015	Jiri Herrmann