



Red Hat Enterprise Linux 7

虚拟化调试和优化指南

优化您的虚拟环境

作者：Dayle Parker 作者：Scott Radvan
翻译、校对：陈西子 – Xizi (Megan) Chen
校对、编辑：傅同杰 – Tongjie (Tony) Fu
校对、责任编辑：鄭中 – Chester Cheng

优化您的虚拟环境

作者：Dayle Parker
红帽公司·工程部出版中心
dayleparker@redhat.com

作者：Scott Radvan
红帽公司·工程部出版中心
sradvan@redhat.com

翻译、校对：陈西子 – Xizi (Megan) Chen
澳大利亚昆士兰大学·笔译暨口译研究生院
ellach@126.com

校对、编辑：傅同杰 – Tongjie (Tony) Fu
红帽公司·全球服务部
tfu@redhat.com

校对、责任编辑：鄭中 – Chester Cheng
红帽公司·全球服务部 & 澳大利亚昆士兰大学·笔译暨口译研究生院
ccheng@redhat.com, chester.cheng@uq.edu.au

法律通告

Copyright © 2013-2015 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

《Red Hat Enterprise Linux 虚拟化调试和优化指南》包括 KVM 和虚拟化性能。此指南为您的系统和客户端虚拟机提供了提示和建议，使您可以充分利用 KVM 性能特点。

目录

第 1 章 简介 2

 1.1. KVM 概述 2

 1.2. KVM 性能构架概述 2

 1.3. 虚拟化性能特性和改进 2

第 2 章 性能监控工具 5

 2.1. 简介 5

 2.2. perf kvm 5

 2.3. 虚拟性能监控单位 7

第 3 章 virt-manager 8

 3.1. 简介 8

 3.2. 操作系统细节和设备 8

 3.3. CPU 性能选项 9

 3.4. 虚拟磁盘性能选项 13

第 4 章 tuned 15

 4.1. 简介 15

 4.2. tuned 和 tuned-adm 15

第 5 章 联网 17

 5.1. 简介 17

 5.2. 联网调试须知 17

 5.3. Virtio 和 vhost_net 17

 5.4. 设备分配和 SR-IOV 18

 5.5. 网络调试技巧 18

第 6 章 块 I/O 20

 6.1. 简介 20

 6.2. 块 I/O 调试 20

 6.3. 缓存 21

 6.4. I/O 模式 21

 6.5. 块 I/O 调试技术 22

第 7 章 内存 24

 7.1. 简介 24

 7.2. 内存调试须知 24

 7.3. 虚拟机内存调试 24

第 8 章 NUMA 29

 8.1. 简介 29

 8.2. NUMA 内存分配策略 29

 8.3. 自动化 NUMA 平衡 29

 8.4. libvirt NUMA 调试 30

 8.5. NUMA-Aware 内核同页合并 37

附录 A. 修订历史 39

第 1 章 简介

1.1. KVM 概述

以下示意图代表 KVM 构架：

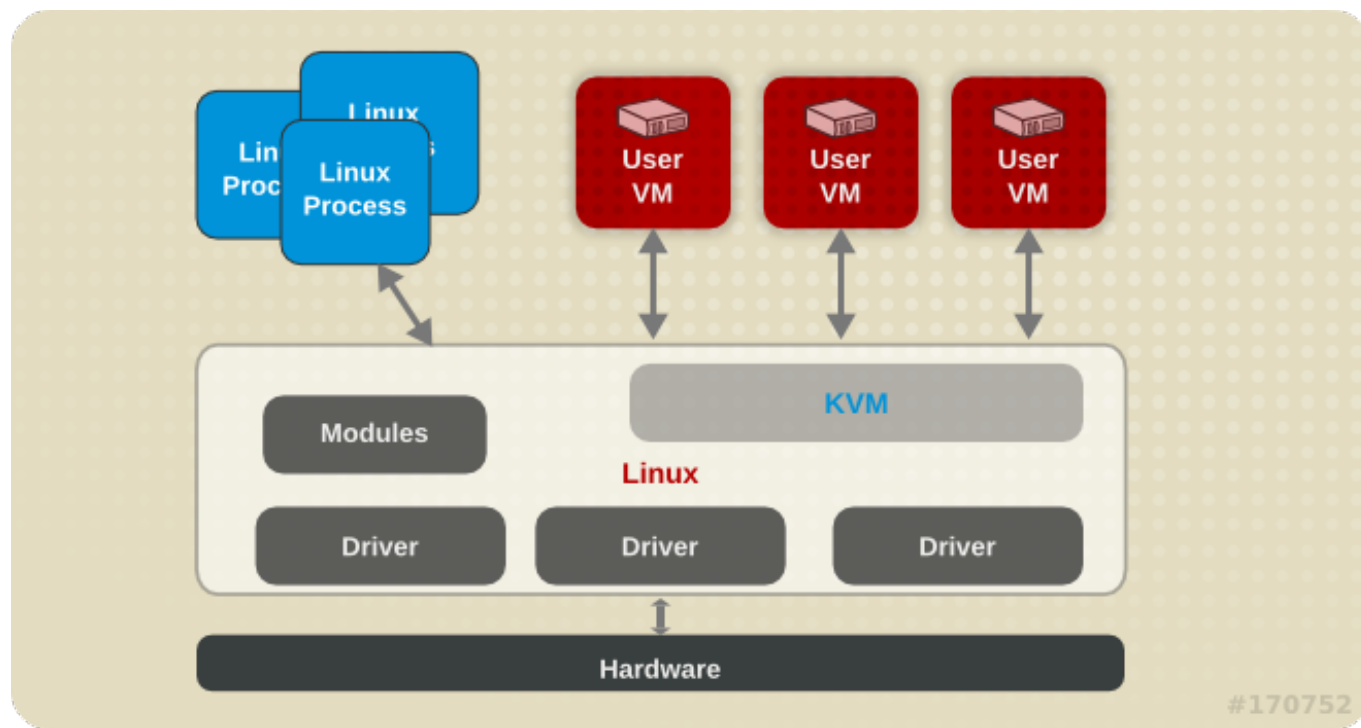


图 1.1. KVM 构架

1.2. KVM 性能构架概述

以下几点提供了 KVM 的简要概述，它适用于系统性能和进程／线程的管理：

- ✧ 使用 KVM 时，客机作为一个 Linux 的进程在主机上运行。
- ✧ 虚拟 CPU (vCPU) 作为正常线程执行，由 Linux 调度器执行。
- ✧ 客机会继承诸如内核中的 NUMA 和大页面一类的功能。
- ✧ 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传送。

1.3. 虚拟化性能特性和改进

Red Hat Enterprise Linux 7 中虚拟化性能的改进

以下功能改进了 Red Hat Enterprise Linux 7 中的虚拟化性能：

自动化 NUMA 平衡

自动化 NUMA 平衡改进了 NUMA 硬件系统中运行应用的性能，且无需为 Red Hat Enterprise Linux 7 客机进行任何手动调试。自动化 NUMA 平衡把任务（任务可能是线程或进程）移到与它们需要访问的内存更近的地方。

如需获取关于自动化 NUMA 平衡的更多信息，请参照[第 8.3 节“自动化 NUMA 平衡”](#)。

多队列 virtio-net

联网方法使数据包发送/接收处理与客机中可用的虚拟 CPU 数量相协调。

关于多队列 virtio-net 的更多信息请参照[第 5.5.2 节“多队列 virtio-net”](#)。

桥接零复制传输

在客机网络和外部网络间传输大数据包中，零复制传输模式（Zero Copy Transmit）对主机 CPU 负荷的减少可以达到 15%，且对吞吐量没有影响。Red Hat Enterprise Linux 7 虚拟机中全面支持桥接零复制传输（Bridge Zero Copy Transmit），但在默认情况下被禁用。

关于零复制传输的更多信息请参照[第 5.5.1 节“桥接零复制传输”](#)。

APIC 虚拟化

更新的 Intel 处理器提供高级可编程中断控制器的硬件虚拟化（APICv，Advanced Programmable Interrupt Controller）。APIC 虚拟化将通过允许客机直接访问 APIC 改善虚拟化 x86_64 客机性能，大幅减少中断等待时间和高级可编程中断控制器造成的虚拟机退出数量。更新的 Intel 处理器中默认使用此功能，并可改善 I/O 性能。

EOI 加速

对在那些较旧的、没有虚拟 APIC 功能的芯片组上的高带宽 I/O 进行 EOI 加速处理。

多队列 virtio-scsi

改进的存储性能和 virtio-scsi 驱动中多队列支持提供的可扩展性。这个命令使每个虚拟 CPU 都可以使用独立的队列和中断，从而不会影响到其他虚拟 CPU。

关于多队列 virtio-scsi 的更多信息请参照[第 6.5.2 节“多队列 virtio-scsi”](#)。

半虚拟化 ticketlocks

半虚拟化 ticketlocks（pvticketlocks）将改善包括过度订阅 CPU 在内的 Red Hat Enterprise Linux 主机中运行的 Red Hat Enterprise Linux 7 客户虚拟机的性能。

半虚拟化页面错误

半虚拟化页面错误在尝试访问主机置换页面时将被加入到客机。这一功能改善了主机内存过载和客机内存被置换时 KVM 的客机性能。

半虚拟化时间 vsyscall 优化

`gettimeofday` 和 `clock_gettime` 系统调用将通过 `vsyscall` 机制在用户空间执行。在此之前，调用这类系统触发需要系统切换到 kernel 模式，之后退回到用户空间。这一操作将极大改善部分应用的性能。

Red Hat Enterprise Linux 中的虚拟化性能特性

✎ CPU/Kernel

- NUMA——非一致性内存访问。关于 NUMA 的详细信息请参照[第 8 章 NUMA](#)。
- CFS——完全公平调度程序。一种新的、使用类（class）的调度程序。

- RCU——读取复制更新。更好地处理共享线程数据。

- 多达 160 种虚拟 CPU (vCPU) 。

» 内存

- 大页面和其他内存密集型环境下的优化。详细信息请参照[第 7 章 内存](#)。

» 联网

- vhost-net——一种基于内核的 VirtIO 快速解决方案。

- SR-IOV——使联网性能级别接近本机。

» 块 I/O

- AIO——支持线程和其他 I/O 操作重叠。

- MSI - PCI 总线设备中断生成。

- 磁盘 I/O 节流——客机磁盘 I/O 的控制要求避免过度使用主机资源。详细信息请参照[第 6.5.1 节 “磁盘 I/O 节流”](#)。



注意

关于虚拟化支持、限制和功能的更多详细信息，请参照《*Red Hat Enterprise Linux 7 虚拟化入门指南*》和以下网址：

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

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

第 2 章 性能监控工具

2.1. 简介

本章节描述用于监控客户虚拟机环境的工具。

2.2. perf kvm

您可以使用带有 **kvm** 选项的 **perf** 命令，从主机收集客机运行系统的统计数据。

在 Red Hat Enterprise Linux 中，*perf* 软件包提供 **perf** 命令。运行 **rpm -q perf** 检查 *perf* 软件包是否已安装。如果没有安装且您想安装该软件包来收集分析客机运行系统统计数据，请以 root 用户运行以下命令：

```
# yum install perf
```

为了在主机中使用 **perf kvm**，您需要从客户端获取 **/proc/modules** 和 **/proc/kallsyms** 文件。有两种可实现的方法。参考以下程序，用[过程 2.1, “从客户端向主机复制 proc 文件”](#) 将文件转换逐级并在文件中运行报告。或者参照[过程 2.2, “备选：使用 sshfs 直接访问文件”](#) 直接安装客户端来获取文件。

过程 2.1. 从客户端向主机复制 proc 文件



重要

如果您直接复制必需文件（例如通过 **scp** 命令），您只会复制零长度的文件。此处描述了首先将客户端中的文件复制到临时位置（通过 **cat** 命令），然后通过 **perf kvm** 命令将它们复制到主机使用的过程。

1. 登录客户端并保存文件

登录客户端并将 **/proc/modules** 和 **/proc/kallsyms** 保存到临时位置，文件名为 **/tmp**：

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

2. 将临时文件复制到主机

在您退出客户端之后，运行以下 **scp** 命令，将已保存的文件复制到主机。如果 TCP 端口和主机名称不符，应替换主机名称：

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

现在您在主机上拥有了两个客户端的文件（**guest-kallsyms** 和 **guest-modules**），准备就绪通过 **perf kvm** 命令进行使用。

3.

通过 **perf kvm** 对事件进行记录和报告

使用前一步骤中获取的文件，并纪录和报告客户端或（与）主机中的事件现在是可行的。

运行以下示例命令：

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



注意

如果 **--host** 和 **--guest** 均在命令中使用，输出将被储存在 **perf.data.kvm** 中。如果只有 **--host** 被使用，文件将被命名为 **perf.data.host**。同样地，如果仅有 **--guest** 被使用，文件将被命名为 **perf.data.guest**。

请按 Ctrl-C 停止纪录。

4.

报告事件

以下示例命令使用纪录过程中获取的文件，并将输出重新定向到一个新文件：**analyze**。

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

查看并分析 **analyze** 文件内容，检测纪录的事件：

```
# 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

[输出删节.....]
```

过程 2.2. 备选：使用 **sshfs** 直接访问文件



重要

这一步仅被作为一个提供的示例。您需要依据环境替换属性值。

```
# Get the PID of the qemu process for the guest:
PID=`ps -eo pid,cmd | grep "qemu.*-name GuestMachine" \
| grep -v grep | awk '{print $1}'`

# Create mount point and mount guest
mkdir -p /tmp/guestmount/$PID
sshfs -o allow_other,direct_io GuestMachine:/ /tmp/guestmount/$PID

# Begin recording
perf kvm --host --guest --guestmount=/tmp/guestmount \
record -a -o perf.data

# Ctrl-C interrupts recording. Run report:
perf kvm --host --guest --guestmount=/tmp/guestmount report \
-i perf.data

# Unmount sshfs to the guest once finished:
fusermount -u /tmp/guestmount
```

2.3. 虚拟性能监控单位

虚拟性能监控单位 (vPMU , virtual performance monitoring unit) 显示客户虚拟机功能的数据。

虚拟性能监控单位允许用户识别客户虚拟机中可能出现的性能问题来源。虚拟性能监控单位基于 Intel 性能监控单位 (Performance Monitoring Unit) 并只能以 Intel 机器为基础。

该特性只能支持运行 Red Hat Enterprise Linux 6 或 Red Hat Enterprise Linux 7 的客户虚拟机，并且会被默认禁用。

要验证系统是否支持虚拟 PMU 时，请运行以下命令检查主机 CPU 中的 **arch_perfmon** 标识：

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

在运行虚拟 PMU 时，以 **host-passthrough** 在客户虚拟机中指定 **cpu mode**：

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

在虚拟 PMU 激活之后，通过运行客户虚拟机的 **perf** 命令显示虚拟机的性能数据。

第 3 章 virt-manager

3.1. 简介

本章节包括了虚拟机管理器 **virt-manager** 中调试可用选项的性能，是一项用来管理客户虚拟机的桌面工具。

3.2. 操作系统细节和设备

3.2.1. 客户虚拟机细节详述

virt-manager 工具会根据新的客户虚拟机的操作系统类型和版本，提供不同的配置文件。创建客户虚拟机时，您应该尽可能地提供详细信息；这样便可以通过启用适用于特定客户虚拟机类型的功能来提高性能。

请参照以下 **virt-manager** 工具的截屏示例。在建立新的客户虚拟机时，请指定想使用的**操作系统类型和版本**：

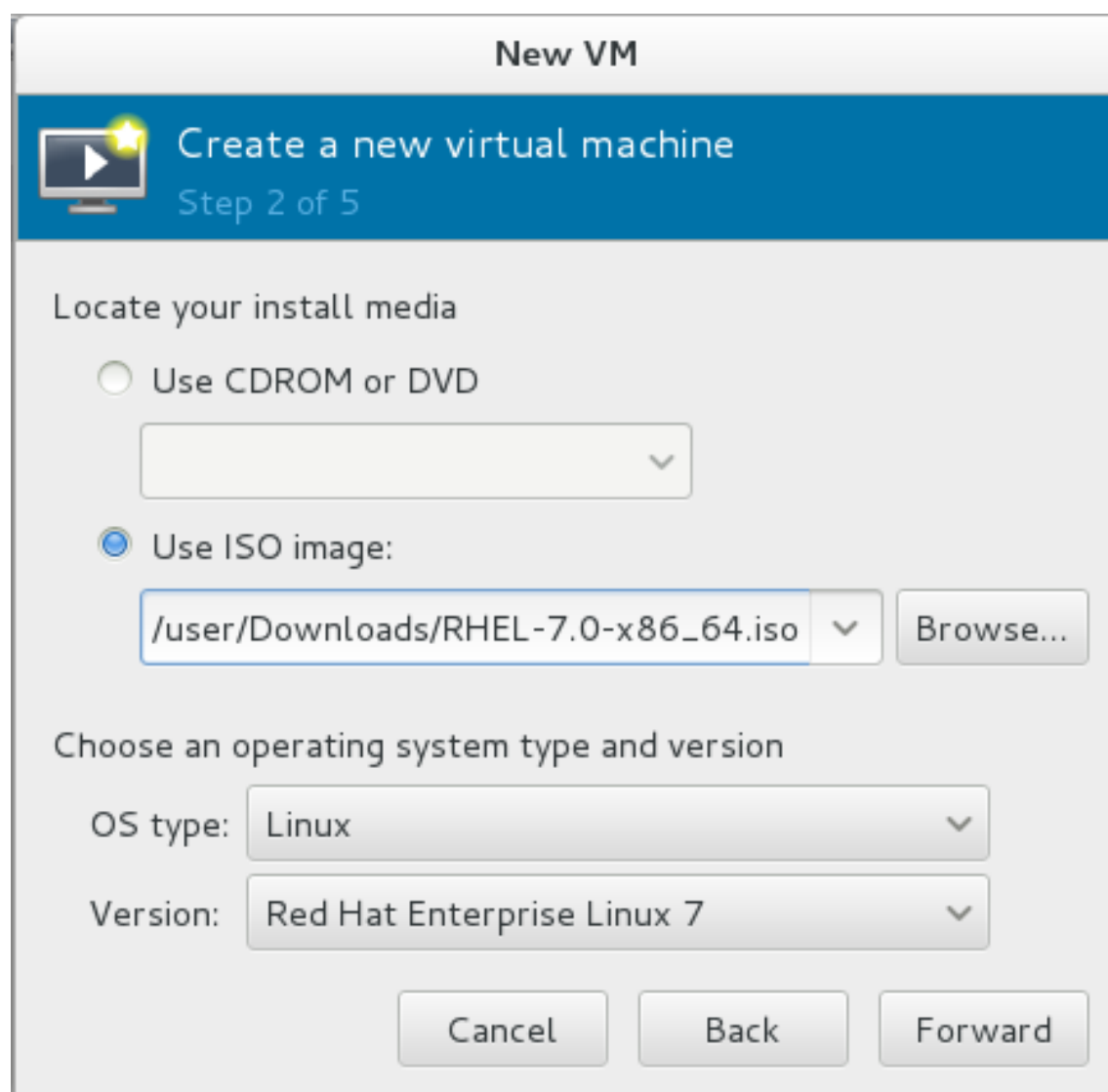


图 3.1. 提供操作系统类型和版本

3.2.2. 删除不使用的设备

删除不使用或不必要的设备可以提高性能。例如，被指派作为网络服务器的客户虚拟机不可能需要音频或附加的平板设备。

请参照以下 **virt-manager** 工具的截屏示例。点击 **移除** 按钮删除不必要的设备：

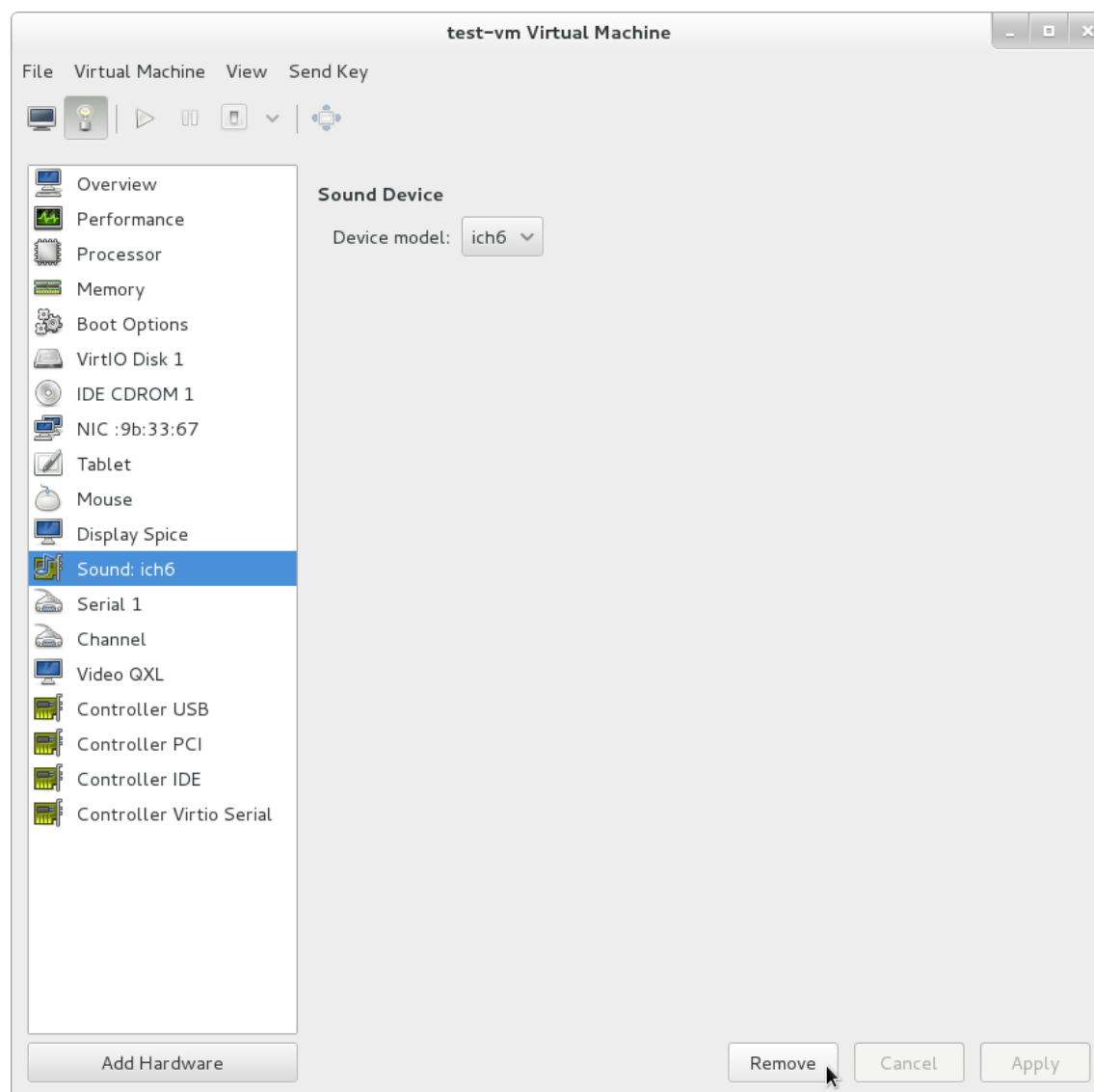


图 3.2. 移除不使用的设备

3.3. CPU 性能选项

客户虚拟机有若干 CPU 相关选项可用。正确配置后，这些选项会对性能产生极大影响。以下图片显示了客户虚拟机可用的 CPU 选项。本章节还会对这些选项的影响进行说明和解释。

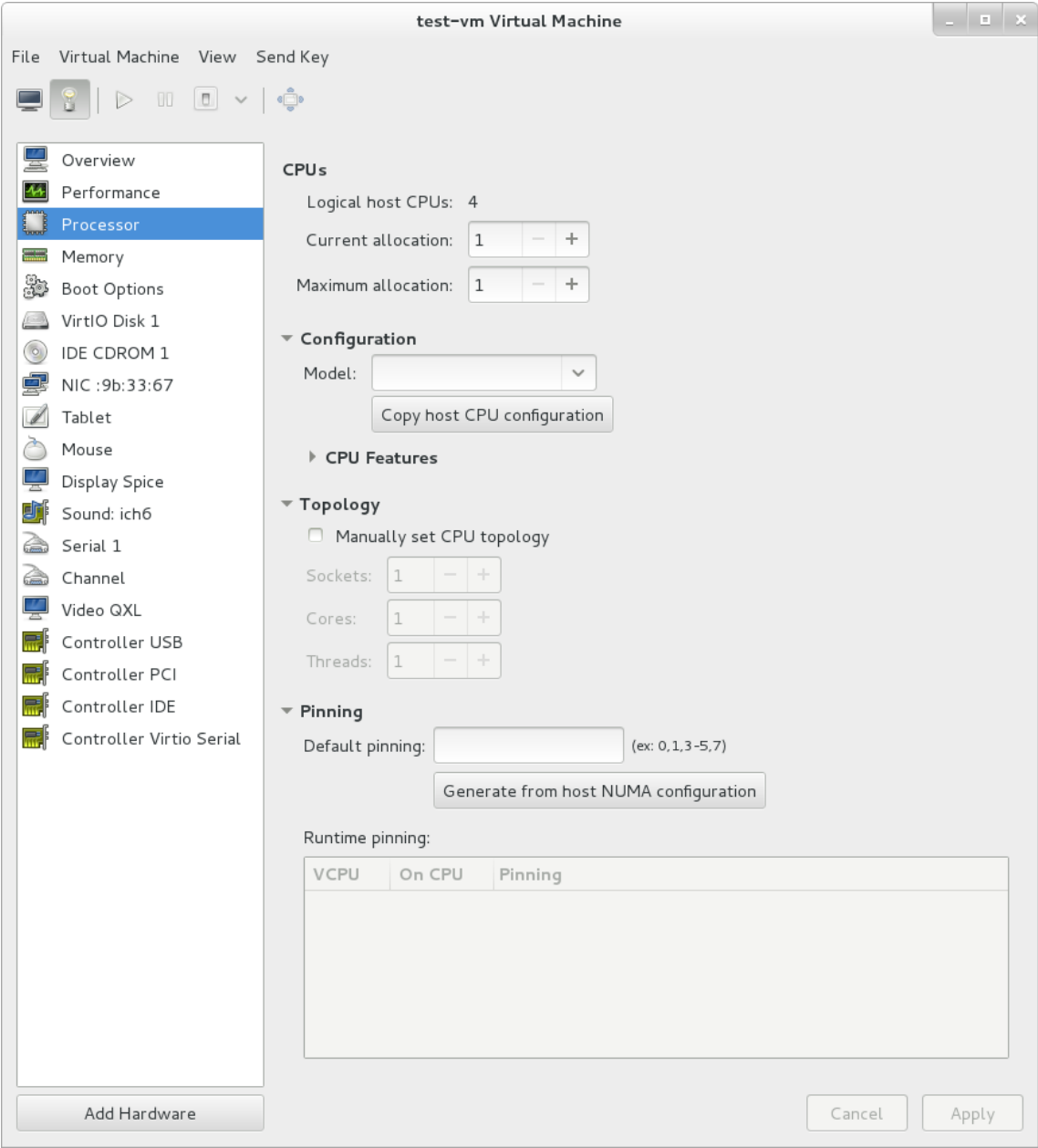


图 3.3. CPU 性能选项

3.3.1. 选项：可用的 CPU

使用该选项来调节客户虚拟机的可用虚拟 CPU（vCPU）的容量。如果您分配的超过了主机可用的数量（overcommitting），会显示警告，如下图中所示：



图 3.4. 过度使用 CPU

如果系统中所有的客机 vCPU 容量大于主机 CPU 总数，CPU 就会被过度使用。如果 vCPU 的总数大于主机 CPU 数量，您可能会使 CPU 被一个或多个客机过度使用。



重要

与内存过度使用相似，CPU 过度使用时，比如在客机负荷过重或无法预测时，可能会给性能造成负面影响。更多有关过度使用的细节请参照《Red Hat Enterprise Linux 虚拟化管理手册 · KVM 过度使用》。

3.3.2. 选项：CPU 配置

根据所需的 CPU 型号，使用以上选项选择 CPU 的配置类型。展开列表查看可用的选项，或点击复制主机 CPU 配置按钮来检测和应用物理主机 CPU 型号或配置。一旦选择了 CPU 配置，CPU 性能列表中就会显示并分别启用/禁用可用的 CPU 特性/指令。以下示意图显示了这些选项：

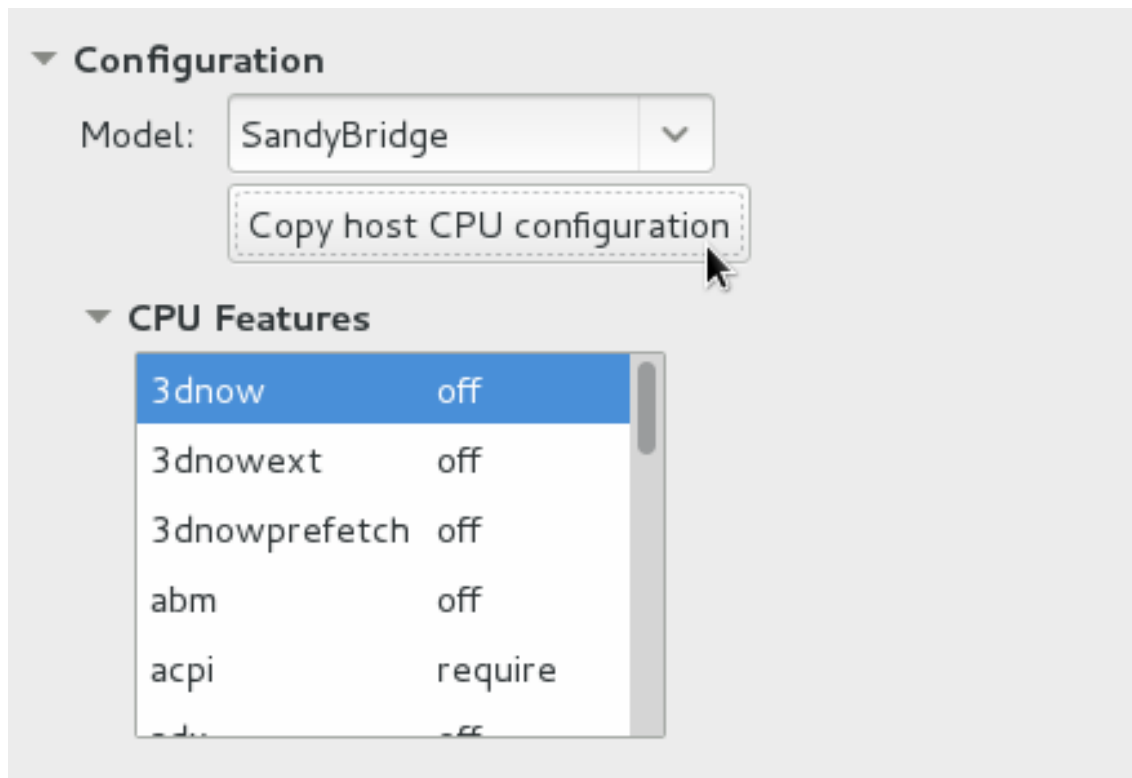


图 3.5. CPU 配置选项



注意

建议在手动配置上复制主机 CPU 配置。

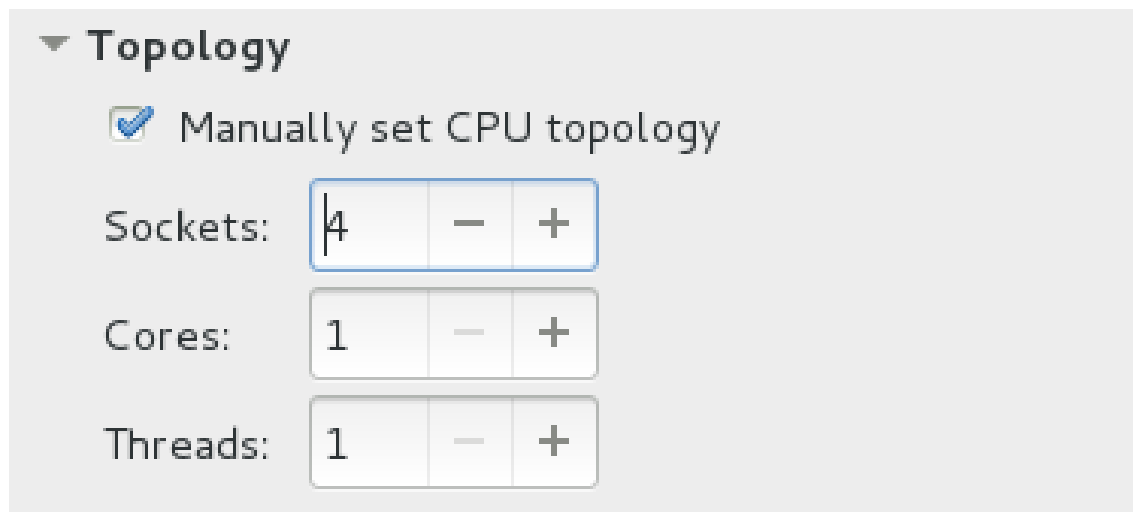


注意

作为替代，在主机上运行 `virsh capabilities` 命令，查看系统虚拟化功能，包括 CPU 类型和 NUMA 功能。

3.3.3. 选项：CPU 拓扑

使用该选项将特定 CPU 拓扑（接口、内核、线程）应用于客户虚拟机中的虚拟 CPU。选项示例请参照以下示意图：



▼ Topology

☒ Manually set CPU topology

Sockets: 4 - +

Cores: 1 - +

Threads: 1 - +

图 3.6. CPU 拓扑选项



注意

尽管您的环境可能会指示其他要求，选择任何所需的接口，但只有一个单一接口和单一线程时性能最佳。

3.3.4. 选项：CPU 钉选 (pinning)

遵循系统特定 NUMA 拓扑可以获得性能的大幅提升。使用该选项自动生成主机可用的钉选配置。

▼

Pinning

Default pinning: (ex: 0,1,3-5,7)

Generate from host NUMA configuration

Runtime pinning:

VCPU	On CPU	Pinning
------	--------	---------

CancelApply

图 3.7. CPU 钉选



敬告

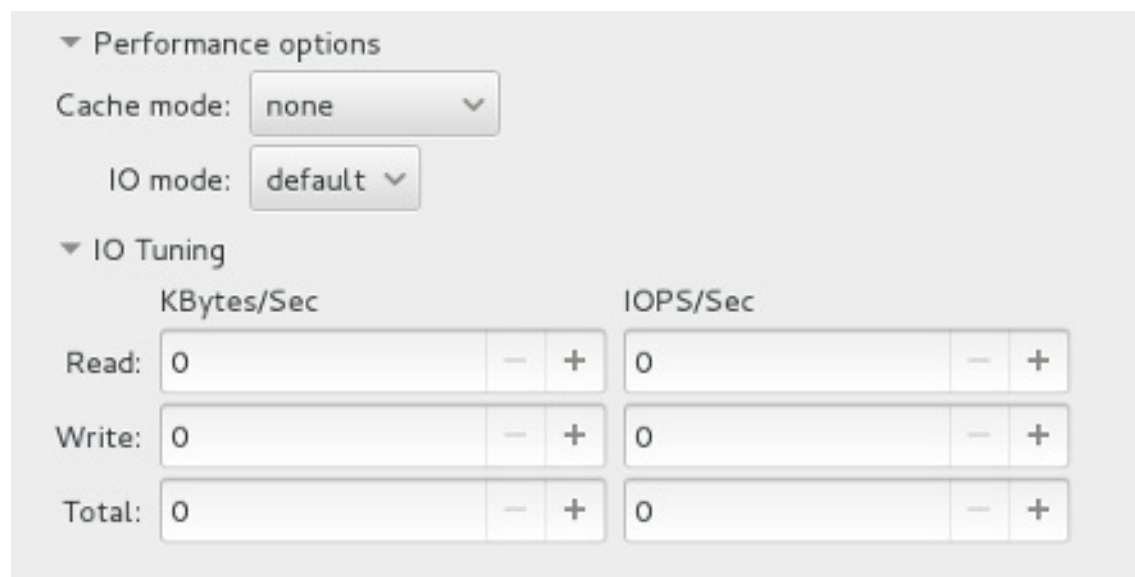
如果客机中的虚拟 CPU 多于单一 NUMA 节点，请勿使用此选项。

使用“钉选”选项会将客机虚拟 CPU 线程限制在单一 NUMA 节点；但线程能够在该 NUMA 节点中移动。如果需要更严密的绑定能力，使用 `lscpu` 命令输出，使用 `virsh cpupin` 在虚拟 CPU 绑定建立 1:1 物理 CPU。NUMA 和 CPU 钉选的更多信息请参照第 8 章 [NUMA](#)。

3.4. 虚拟磁盘性能选项

影响性能的安装过程中，您的客户虚拟机可使用若干虚拟磁盘相关选项。以下图片显示了客机中可用的虚拟磁盘选项。

在 **virt-manager** 的**虚拟磁盘**部分可以选择缓存模式、IO 模式和 IO 调试。以上参数可在**性能选项**下的字段中设置。具体如下图：



The screenshot shows the 'Performance options' section of the virt-manager interface. Under 'Performance options', 'Cache mode' is set to 'none' and 'IO mode' is set to 'default'. Below this is the 'IO Tuning' section, which contains two columns of controls: 'KBytes/Sec' and 'IOPS/Sec'. Each column has three rows: 'Read', 'Write', and 'Total'. Each row consists of a numeric input field (all set to 0) and two buttons, '-' and '+', for adjusting the values.

	KBytes/Sec			IOPS/Sec		
Read:	0	-	+	0	-	+
Write:	0	-	+	0	-	+
Total:	0	-	+	0	-	+

图 3.8. 虚拟磁盘性能选项



重要

在设置 **virt-manager** 中虚拟磁盘性能选项时，虚拟机被重启后才能使设置生效。

编辑客机 XML 配置中设置的设定和指令的描述请参见[第 6.3 节“缓存”](#)和[第 6.4 节“I/O 模式”](#)。

第 4 章 tuned

4.1. 简介

本章节的内容涵盖了在虚拟环境中使用 **tuned** 守护进程调整系统设置。

4.2. tuned 和 tuned-adm

tuned 是一项配置文件调整机制，使 Red Hat Enterprise Linux 适应特定负载特性，例如 CPU 密集型任务要求，或存储/网络吞吐量的响应能力。

伴随它的工具 **ktune** 结合了 **tuned-adm** 工具，提供大量预先配置的调整分析，以便在大量具体使用案例中提高性能并降低能耗。编辑这些配置或者创建新配置可生成为系统定制的性能解决方案。

提供作为 **tuned-adm** 中一部分的虚拟化相关文件包括：

virtual-guest

基于 **throughput-performance** 文件，**virtual-guest** 同样会降低虚拟内存的 swappiness。

在创建 Red Hat Enterprise Linux 7 客户虚拟机时，**virtual-guest** 文件将被自动选择。建议虚拟机使用该文件。

本文件在 Red Hat Enterprise Linux 6.3 和之后可用，但在安装虚拟机时须手动选择。

virtual-host

基于 **throughput-performance** 文件，**virtual-host** 也会降低虚拟内存的 swappiness，并启用更积极的脏页（dirty page）回写。建议虚拟化主机使用本文件，包括 KVM 和 Red Hat Enterprise Virtualization 主机。

在 Red Hat Enterprise Linux 7 安装的默认情况下，将安装 **tuned** 软件包，并启用 **tuned** 服务。

要列出所有可用配置文件并识别目前激活的配置文件，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
```

也可创建自定义 **tuned** 文件对一组调整参数进行封装。创建自定义 **tuned** 文件的使用说明，请参考 **tuned.conf** 手册页。

要只显示当前激活的配置文件请运行：

```
tuned-adm active
```

要切换到某个可用的配置文件请运行：

```
tuned-adm profile profile_name
```

例如，切换到 **virtual-host** 文件，请运行：

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



重要

设置 Red Hat Enterprise Linux 7.1 中的 **tuned** 配置文件和以上命令后，需重启 **tuned** 服务，且系统必须被重启使其持续应用修改。更多信息请查看《Red Hat Enterprise Linux 7 性能调试指南》。

在某些情况下，更优的选择是禁用 **tuned** 并使用手动设定的参数。禁用一切调试请运行：

```
tuned-adm off
```



注意

有关 **tuned**、**tuned-adm** 和 **ktune** 的更多信息，请参照《Red Hat Enterprise Linux 7 电源管理指南》，可由 <https://access.redhat.com/documentation/en-US/> 获取。

第 5 章 联网

5.1. 简介

本章节包括虚拟化环境中的联网优化主题。

5.2. 联网调试须知

- » 使用多个网络以避免单一网络过载。例如用专用网络进行管理、备份及/或实时迁移。
- » 通常，在所有组件中可以满足匹配最大传输单元（1,500字节）。如果需要更大的消息，提高最大传输单元值可以减少碎片。如果改变了最大传输单元，路径中所有的设备都应该有一个匹配的最大传输单元值。
- » 使用 **arp_filter** 阻止 ARP Flux，这种不良情况可能会在主机和客机中发生，造成这一现象的原因是机器从一个以上网络界面响应 ARP 请求：运行 **echo 1 > /proc/sys/net/ipv4/conf/all/arp_filter** 或编辑 **/etc/sysctl.conf** 让重启后这设置得以持续。



注意

关于 ARP Flux 的更多信息请参照以下网址：<http://linux-ip.net/html/ether-arp.html#ether-arp-flux>。

5.3. Virtio 和 vhost_net

以下示意图将演示 Virtio 和 vhost_net 构架中，kernel 的作用。

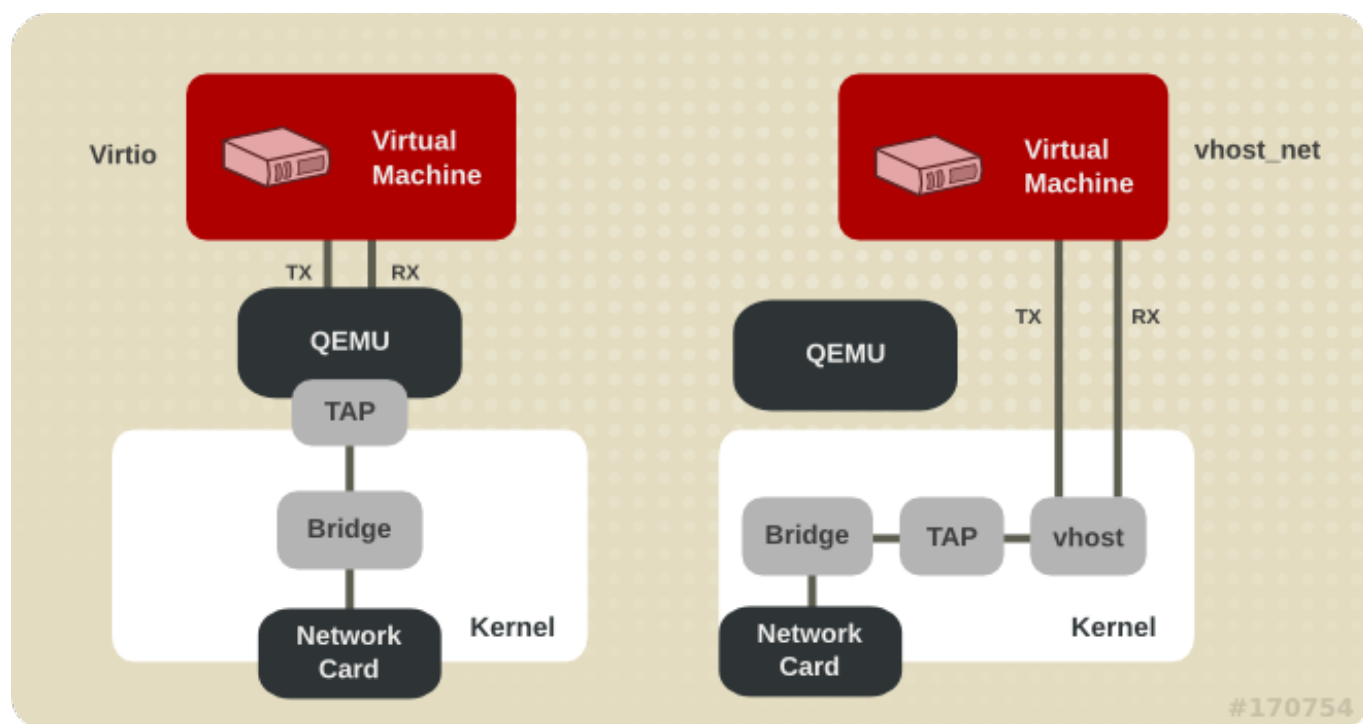


图 5.1. Virtio 和 vhost_net 构架

vhost_net 将部分 Virtio 驱动从用户空间移至 kernel。这将减少复制操作、降低延迟和 CPU 占用量。

5.4. 设备分配和 SR-IOV

以下示意图将演示设备分配和 SR-IOV 构架中 kernel 的作用。

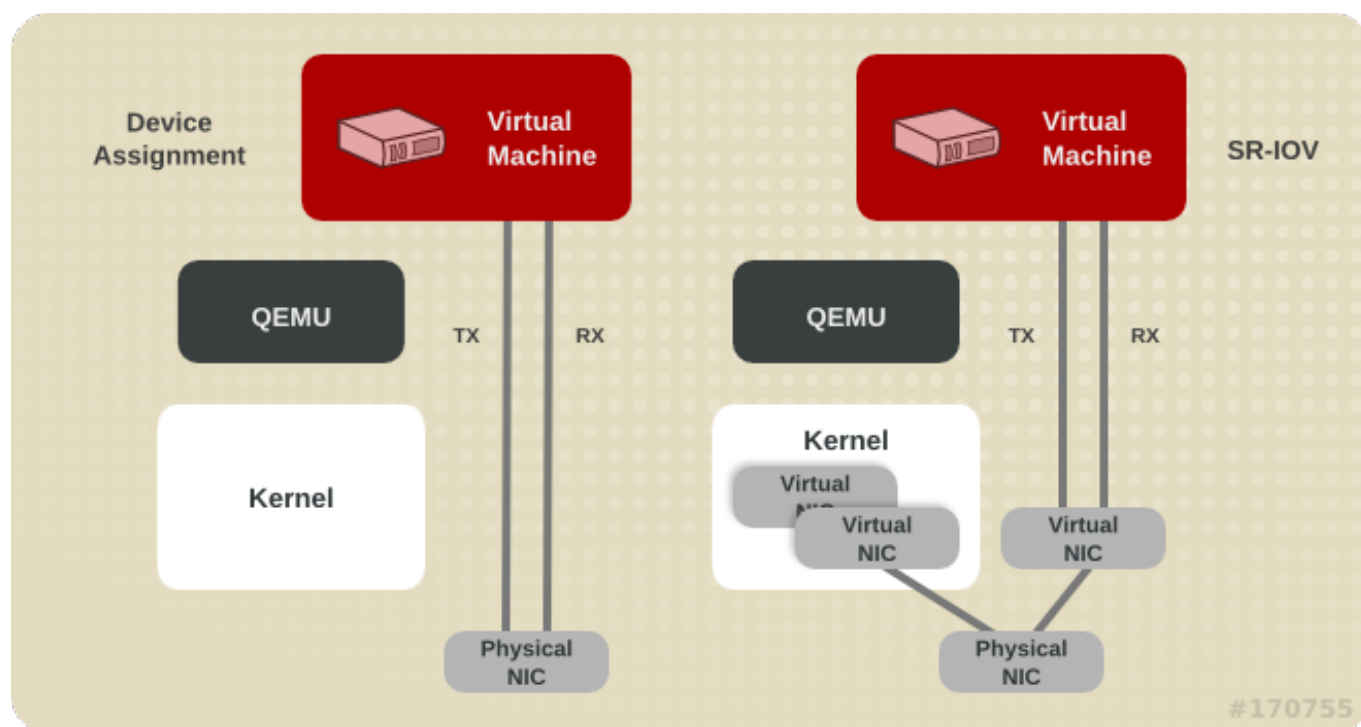


图 5.2. 设备分配和 SR-IOV

设备分配使整个设备在客机中可用。SR-IOV 需要驱动和硬件中的支持，包括 NIC 和系统板，并允许创建多个虚拟设备并进入不同的客机。客机中需要针对供应商的驱动，但 SR-IOV 提供一切网络选项中的最低延迟。

5.5. 网络调试技巧

这一部分描述了虚拟化环境中网络性能调试的技术。

5.5.1. 桥接零复制传输

零复制传输 (Bridge Zero Copy Transmit) 模式对于大尺寸的数据包较为有效。通常在客机网络和外部网络间的大数据包传输中，它对主机 CPU 负荷的减少可达到 15%，对吞吐量没有影响。

它不对客机到客机、客机到主机或小数据包负载造成影响。

Red Hat Enterprise Linux 7 虚拟机完全支持桥接零复制传输，但是被默认禁用。若需要启动零复制传输模式，请将 vhost_net 模块的 **experimental_zcopytx** kernel 模块参数设置到 1。

注意

由于针对限制服务和信息的威胁防御技术泄露攻击，在传输中通常会创建一个额外的数据复制。启动零复制传输将禁用这一威胁防御技术。

如果有性能回归倾向，或如果无需担心主机 CPU 的使用，零复制传输模式可以通过将 `experimental_zcopytx` 设置到 0 被禁用。

5.5.2. 多队列 virtio-net

多队列 virtio-net 提供了随着虚拟 CPU 数量增加而改变网络性能的方法，即允许每一次通过一组以上的 virtqueue 传输数据包。

今天的高端服务器拥有更多处理器，其中所运行客机的虚拟 CPU 数量往往也在增加。在单一队列的 virtio-net 中，客机中协议堆叠的缩放收到限制，因为网络性能不随虚拟 CPU 数量的增加而改变。鉴于 virtio-net 只有一组 TX 和 RX 队列，客机不能并行传输或检索数据包。

多队列支持通过允许并行的数据包处理移除这些瓶颈。

多队列 virtio-net 在这些时候可以提供最佳性能：

- » 流量数据包相对较大。
- » 客机同时在各种连接中活跃，流量从客机、客机到主机或客户端运行到外部系统。
- » 队列数量与虚拟 CPU 相同。因为多队列支持可以优化 RX 中断关联和 TX 队列选择，实现特定队列对于特定虚拟 CPU 的私有化。



注意

多队列 virtio-net 在输入流量中运行良好，但在少数情况下可能会影响输出流量的性能。启用多队列 virtio-net 提高总体吞吐量，同时提高 CPU 可用量。

5.5.2.1. 配置多队列 virtio-net

使用多队列 virtio-net 时，通过向客机 XML 配置（ N 的值为 1 到 8，即 kernel 最多可支持一组多队列 tap 设备中的 8 个队列）添加以下命令：

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

与客机中的 N virtio-net 队列一同运行虚拟机时，执行以下命令（ M 的值为 1 到 N ）时允许多队列支持：

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

第 6 章 块 I/O

6.1. 简介

本章节包括虚拟环境中的优化 I/O 设置。

6.2. 块 I/O 调试

virsh blkiotune 命令允许管理员在客机 XML 配置的 **<blkio>** 元素中，手动设置或显示客户虚拟机的块 I/O 参数。

为虚拟机显示当前的 **<blkio>** 参数：

```
# virsh blkiotune virtual_machine
```

设置虚拟机的 **<blkio>** 参数，请参照以下命令并依据环境替换属性值：

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

参数包括：

weight

I/O 的权重范围在 100 到 1,000 之间。

device-weights

列出一个或多个设备/权值组群的单独字符串，以 **/path/to/device,weight,/path/to/device,weight** 为格式。每一个权值必须在 100-1,000 范围内，或者 0 值从每一个设备列表删除该设备。只修改字符串中列出的设备；任何现有的其它设备的权值保持不变。

config

添加 **--config** 选项，使更改在下次启动时生效。

live

添加 **--live** 选项，在运行的虚拟机中应用这些更改。



注意

--live 选项要求监控程序支持这一行为。并非所有监控程序都允许最大内存限制的实时更改。

current

添加 **--current** 选项，在当前的虚拟机中应用这些更改。



注意

关于 `virsh blkio tune` 命令使用的更多信息，请参照 `# virsh help blkio tune`。

6.3. 缓存

缓存选项可以在客机安装期间用 `virt-manager` 进行配置，或通过编辑客机 XML 配置在现存的客户虚拟机中配置。

表 6.1. 缓存选项

缓存选项	描述
Cache=none	客机中的 I/O 不能在主机上缓存，但可以保留在回写磁盘缓存中。在客机中使用此选项来应对较大的需求。此选项通常是支持迁移的最佳和唯一选项。
Cache=writethrough	客机中的 I/O 在主机上缓存，但在物理媒介中直写。该模式较慢且更易造成缩放问题。最好是在客机数量较小且 I/O 需求较低的情况下使用。推荐的应用对象是无需迁移、不支持回写缓存的客机（如 Red Hat Enterprise Linux 5.5 或更早的版本）。
Cache=writeback	客机中的 I/O 在主机上缓存。
Cache=directsync	与 <i>writethrough</i> 相似，但客机中的 I/O 将忽略主机页面缓存。
Cache=unsafe	主机可能会缓存所有的 I/O 磁盘，客机的同步要求将被忽略。
Cache=default	如果没有指定缓存模式，将会选择系统默认设置。

在 `virt-manager` 中，缓存模式可以在 **Virtual Disk** 下被指定。关于使用 `virt-manager` 以更改缓存模式的信息，请参照[第 3.4 节“虚拟磁盘性能选项”](#)。

在客机 XML 中配置缓存模式，请编辑设置在 `driver` 标签内部的 `cache`，指定一个缓存选项。例如，将缓存设置为 *writeback*：

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

6.4. I/O 模式

I/O 模式选项可以在客机安装期间用 `virt-manager` 进行配置，或通过编辑客机 XML 配置在现存的客户虚拟机中配置。

表 6.2. IO 模式选项

IO 模式选项	描述
IO=native	Red Hat Enterprise Virtualization 环境的默认值。该模式适用于直接 I/O 选项的 kernel 非同步 I/O。
IO=threads	默认为基于主机用户模式的线程。
IO=default	Red Hat Enterprise Linux 7 默认为线程模式。

在 **virt-manager** 中，**虚拟硬盘** 下可以指定 I/O 模式。关于使用 **virt-manager** 以改变 I/O 模式的信息，请参照第 3.4 节“[虚拟磁盘性能选项](#)”。

对客户虚拟机 XML 中的 I/O 模式进行配置时，编辑 **driver** 标签中的 **io** 设置，指定 **native**、**threads** 或 **default**。例如，将 I/O 模式设置为 **threads**：

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

6.5. 块 I/O 调试技术

这一部分描述了更多关于虚拟化环境中块 I/O 性能调试的技术。

6.5.1. 磁盘 I/O 节流

当若干虚拟机同时运行时，它们有可能因过度使用磁盘 I/O 而干扰系统性能。KVM 中的磁盘 I/O 节流可以为虚拟机发往主机的磁盘 I/O 请求设定限制。这样可以避免虚拟机过度使用共享资源，并防止影响其他虚拟机的性能。

磁盘 I/O 节流可以在各种情况下使用，例如当不同客户的客户虚拟机在同一台主机中运行时，或不同的客户虚拟机需要服务质量保证时。磁盘 I/O 节流还可以用于模拟更慢的磁盘。

I/O 节流可以在客机附加的每个块设备中独立应用，并支持吞吐量和 I/O 操作中的限制。请使用 **virsh blkdeviotune** 命令为虚拟机设置 I/O 限制。请参照以下示例：

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

Device 为虚拟机附加的其中一个磁盘设备指定了独特的目标名称 (**<target dev='name' />**) 或来源文件 (**<source file='name' />**)。使用 **virsh domblklist** 命令获取磁盘设备名称列表。

可选的参数包括：

total-bytes-sec

字节每秒的总吞吐量限制。

read-bytes-sec

字节每秒的读取吞吐量限制。

write-bytes-sec

字节每秒的写入吞吐量限制。

total-iops-sec

每秒的 I/O 操作总量限制。

read-iops-sec

每秒的读取 I/O 操作限制。

write-iops-sec

每秒的写入 I/O 操作限制。

例如，如需将 **virtual_machine** 虚拟机中的 **vda** 节流至 I/O 每秒 1000、吞吐量为每秒 50 MB，请运行以下命令：

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

6.5.2. 多队列 virtio-scsi

多队列 virtio-scsi 提供了改进的存储性能和 virtio-scsi 驱动中的可扩展性。该命令使每个虚拟 CPU 可以使用独立队列和中断，从而不会影响到其他虚拟 CPU。

6.5.2.1. 配置多队列 virtio-scsi

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

如需启用客机中的多队列 virtio-scsi 支持，在客机 XML 配置中添加以下命令，其中 *N* 为虚拟 CPU 队列的总数：

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

第 7 章 内存

7.1. 简介

本章节包括虚拟环境的内存优化选项。

7.2. 内存调试须知

在虚拟环境中优化内存性能，需考虑以下几点：

- ✧ 请勿为客机分配超过所需的其它资源。
- ✧ 在可能的情况下，如果 NUMA 节点中有足够的资源，将一台客机分配到一个单一 NUMA 节点。关于 NUMA 使用的更多信息，请参照[第 8 章 NUMA](#)。

7.3. 虚拟机内存调试

7.3.1. 内存监控工具

在裸机环境中使用的监控内存的工具也可以在虚拟机中使用。以下工具可以被用来监控内存的使用情况，以及诊断与内存相关的问题：

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



注意

关于使用以上性能工具的详细信息，请参照《Red Hat Enterprise Linux 7 性能调节指南》和上述命令的手册页。

7.3.2. 使用 virsh 调试内存

客机 XML 配置中可选的 **<memtune>** 元素允许管理员手动配置客户虚拟机内存设置。如果省略 **<memtune>**，内存设置将被默认应用。

通过 **virsh memtune** 命令显示或设置虚拟机 **<memtune>** 元素中的内存参数，依据环境替换属性值：

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

可选的参数包括：

hard_limit

虚拟机可以使用的最大内存，单位是千字节（1,024 字节块）

**警告**

该限制设置过低可导致虚拟机被内核终止。

soft_limit

发生内存争用时，内存限制使用的单位是千字节（1,024 字节块）。

swap_hard_limit

虚拟机加上转化可使用的最大内存，单位是千字节（1,024 字节块）。***swap_hard_limit*** 参数值必须大于 ***hard_limit*** 参数值。

min_guarantee

保证分配给虚拟机可以使用的最小内存，单位是千字节（1,024 字节块）。

**注意**

使用 **virsh memtune** 命令的更多信息请参照 **# virsh help memtune**。

可选的 **<memoryBacking>** 元素可能包含若干影响主页存储备份虚拟内存页面的元素。

设置 **locked** 会阻止主机交换属于客机的内存页面。向客机 XML 添加以下命令，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 **<memtune>** 元素中设置 **hard_limit**，使其达到客机配置的最大内存，以及该进程本身所消耗的内存。

设置 **nosharepages** 阻止主机合并并在客机间使用的相同内存。通过向客机的 XML 添加以下命令，指示虚拟机监控程序禁用与客机的共享页面：

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

7.3.3. 大页面和透明大页面（THP）

x86 CPU 通常会在 4kB 页面中处理内存，但可以使用更大的 2MB 或 1GB 页面，即 *huge page*（大页面）。大页面内存可以支持 KVM 客机部署，通过增加点击转换后备缓冲器（TLB）的 CPU 缓存以改善性能。

kernel 功能将在 Red Hat Enterprise Linux 7 中默认启用，大页面可以大幅提高性能，尤其是对于较大的内存和内存密集型的负载。Red Hat Enterprise Linux 7 可以通过使用大页面增加页面大小，以便有效管理大量内存。

过程 7.1. 为客机启用 1GB 大页面

1. Red Hat Enterprise Linux 7.1 系统支持 2MB 或 1GB 大页面，分配将在启动或运行时进行。页面大小均可以在运行时被释放。例如，在启动时分配 4 个 1GB 的大页面和 1,024 个 2MB 的大页面，请使用以下命令行：

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

此外，大页面还可以在运行时分配。运行时分配允许系统管理员选择从何种 NUMA 模式分配页面。然而由于内存碎片的存在，运行时的页面分配会比启动时分配更容易造成分配失败。以下运行时的分配示例显示了从 **node1** 分配 4 个 1GB 的大页面以及从 **node3** 分配 1,024 个 2MB 的大页面：

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

2. 接下来，将 2MB 和 1GB 的大页面挂载到主机：

```
# 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. 重启 libvirtd，使 1GB 大页面可以在客机上启用：

```
# systemctl restart libvirtd
```

1GB 大页面现在对客机不可用。配置 NUMA 节点特定大页面，请参照[第 8.4.9 节“向多个客机 NUMA 节点指定主机大页面”](#)。

7.3.3.1. 透明大页面配置

透明大页面（THP，transparent huge page）将为性能自动优化系统设置。通过允许所有的空余内存被用作缓存以提高性能。

一旦 `/sys/kernel/mm/transparent_hugepage/enabled` 被设置为 **always**，透明大页面将被默认使用。运行以下命令禁用透明大页面：

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

透明大页面支持不能阻止 hugetlbfs 的使用。但在 hugetlbfs 未使用时，KVM 将使用透明大页面来替代常规的 4KB 页面大小。

7.3.3.2. 静态大页面配置

在某些情况下，更优的选择是增加对大页面的控制。在客机中使用静态大页面，使用 **virsh edit** 向客机 XML 配置添加以下命令：

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

以上命令使主机使用大页面向客机分配内存，以替代使用默认的页面大小。

在 Red Hat Enterprise Linux 7.1 中，主机中的大页面可以被分配到客机的 NUMA 节点。在客机 XML 的 **<memoryBacking>** 元素中指定大页面的大小、单位和客机的 NUMA 节点集。关于配置的更多信息和具体示例，请参考[第 8.4.9 节“向多个客机 NUMA 节点指定主机大页面”](#)。

查看当前的大页面值，请运行以下命令：

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

过程 7.2. 大页面设置

以下程序示例显示了大页面设置的命令。

1. 查看当前的大页面值：

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

2. 大页面将以 2MB 为增量进行设置。将大页面的数量设置到 25,000，请运行以下命令：

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



注意

此外，如需进行永久设置，请使用 **# sysctl -w vm.nr_hugepages=N** 命令和显示为大页面数量的 *N*。

3. 大页面挂载：

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

4. 重启 libvirtd，之后再运行以下命令重启虚拟机：

```
# systemctl start libvirtd
```

```
# virsh start virtual_machine
```

5. 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

大页面不仅可以使主机受益，也可以使客机受益。但它们的大页面值总量必须小于主机中的可用值。

第 8 章 NUMA

8.1. 简介

过去，x86 系统中的所有内存都可以通过 CPU 进行同等访问。无论任何 CPU 执行操作，访问时间都相等，这也被称为“统一内存访问”（UMA，Uniform Memory Access）。

最近使用的 x86 处理器已经不再采取这一行为。在非统一内存访问（NUMA，Non-Uniform Memory Access）中，系统内存被划分到 NUMA 节点（node），并且与 socket 相对应，或与特定某一组与本地系统内存子集具有相同访问延迟的 CPU 相对应。

本章节描述了虚拟环境中的内存分配和 NUMA 调试配置。

8.2. NUMA 内存分配策略

以下三种策略定义了系统中节点对内存的分配：

Strict

目标节点中不能分配内存时，分配将被默认操作转进至其他节点。严格的策略意味着，当目标节点中不能分配内存时，分配将会失效。

Interleave

内存页面将被分配至一项节点掩码指定的节点，但将以轮循机制的方式分布。

Preferred

内存将从单一最优内存节点分配。如果内存并不充足，内存可以从其他节点分配。

XML 配置启用所需策略：

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

8.3. 自动化 NUMA 平衡

自动化 NUMA 平衡改进了 NUMA 硬件系统中运行应用的性能。它在 Red Hat Enterprise Linux 7 系统中被默认启用。

通常，应用程式在其程序的线程访问 NUMA 节点上的内存、且此节点位置与线程排程时的位置相同的时候，性能最佳。自动化 NUMA 平衡会把任务（任务可能是线程或进程）移到与它们需要访问的内存更近的地方，同时也会移动内存应用程序数据，使其更靠近参考这一数据的任务。以上均在自动化 NUMA 平衡启用时由内核自动完成。

自动化 NUMA 平衡使用若干算法和数据结构，它们只有在系统中自动化 NUMA 平衡活跃时才能被启用和分配。

- ✧ 进程内存的周期性 NUMA 取消对应
- ✧ NUMA hinting 故障
- ✧ 故障迁移（MoF，Migrate-on-Fault）——将内存移动至需要运行的程序

✧ `task_numa_placement` —— 移动运行的程序，使接近其内存

8.3.1. 配置自动化 NUMA 平衡

自动化 NUMA 平衡在 Red Hat Enterprise Linux 7 中默认启用，并在 NUMA 属性硬件中引导时自动激活。

自动化 NUMA 平衡启用时需满足以下两个条件：

- ✧ `# numactl --hardware` 显示多个节点，以及
- ✧ `# cat /sys/kernel/debug/sched_features` 在标识中显示 **NUMA**

应用程序的手动 NUMA 调试将会重载自动化 NUMA 平衡，并禁用周期性的内存空白、NUMA 错误、迁移和以上应用程序的自动化 NUMA 放置。

在某些情况下，首选系统范围内的手动 NUMA 调试。

要禁用自动化 NUMA 平衡，请使用以下命令：

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

要启用自动化 NUMA 平衡，请使用以下命令：

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

8.4. libvirt NUMA 调试

通常，NUMA 系统通过将客机大小限制在单一 NUMA 节点的资源数量实现最佳性能。应当避免 NUMA 节点中不必要的分散资源。

使用 **numastat** 工具对进程和操作系统的每个 NUMA 节点内存统计进行查看。

在以下示范中，**numastat** 工具显示了 NUMA 节点中的四种非优化内存排列的虚拟机：

```
# 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**，使客机 CPU 和内存资源自动对齐。

接下来再次运行 **numastat -c qemu-kvm**，以查看 **numad** 的运行结果。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

同时运行 **numastat** 和 **-c** 可提供简洁的输出；添加 **-m** 可将每个节点上系统范围内的内存信息添加到输出。更多信息请参见 **numastat** 手册页。

8.4.1. NUMA 虚拟 CPU 钉选

虚拟 CPU 钉选为裸机系统中的任务钉选提供了相似的优点。由于虚拟 CPU 在主机操作系统中作为用户空间任务运行，钉选将提高缓存效率。例如，当所有虚拟 CPU 线程都在同一个物理 socket 中运行时，将会共享 L3 缓存域。

合并虚拟 CPU 钉选和 **numatune** 可以避免 NUMA 缺失。NUMA 缺失会对性能带来显著影响，通常会有至少 10% 或更多的性能损失。虚拟 CPU 钉选和 **numatune** 应该同时配置。

当虚拟机在执行存储或网络 I/O 任务时，最好将所有的虚拟 CPU 和内存固定至同一个连接到 I/O 适配器的物理 socket。

注意

Istopo 工具可以被用作使 NUMA 拓扑可视化。同时还可以用作验证虚拟 CPU 在同一个物理插槽中的内核绑定。更多信息请参照知识库文章，链接如下：**Istopo**：
<https://access.redhat.com/site/solutions/62879>。

重要

在虚拟 CPU 的数量远多于物理内核时，钉选将导致复杂性增加。

以下示例 XML 配置具有固定在物理 CPU 0-7 上的域过程。虚拟线程固定在其自身的 cpuset 上。例如，虚拟 CPU0 被固定在物理 CPU 0 上，虚拟 CPU1 被固定在物理 CPU 1 上等等：

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

虚拟 CPU 与 vcpupin 标签之间有直接的联系。如果 vcpupin 选项不是确定的，那么属性值会被自动指定并会从上一级虚拟 CPU 标签选项中继承。以下配置显示了因为 **vcpu 5** 丢失的 **<vcpupin>**。进而，**vCPU5** 将会被固定在物理 CPU 0-7 上，正如上一级标签 **<vcpu>** 中指定的那样：

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



重要

<vcpupin>、**<numatune>** 以及 **<emulatorpin>** 应该被一同配置，从而实现优化、确定的性能。更多关于 **<numatune>** 标签的信息，请查看[第 8.4.2 节“域进程”](#)。更多关于 **<emulatorpin>** 标签的信息，请参照[第 8.4.4 节“使用 emulatorpin”](#)。

8.4.2. 域进程

如 Red Hat Enterprise Linux 所提供的那样，libvirt 使用 libnuma 来为域进程设定内存绑定政策。这些政策的节点集可以作为 *static*（在域 XML 中指定），或 *auto*（通过询问 numad 进行配置）进行配置。关于如何配置这些 **<numatune>** 标签内部的示例，请参考以下 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)** 来为域进程设定 CPU 绑定政策。cpuset 选项既可以是 *static*（静态；在域 XML 中指定），也可以是 *auto*（自动；通过询问 numad 进行配置）。关于如何配置这些 **<vcpu>** 标签内部的示例，请参考以下 XML 配置：

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

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

在 **<vcpu>** 所使用的放置模式和 **<numatune>** 之间有隐式的继承规则：

- ✱ **<numatune>** 的放置模式默认为与 **<vcpu>** 相同的放置模式，当 **<nodeset>** 被指定时，则被默认为 *static*。
- ✱ 同样，**<vcpu>** 的放置模式默认为与 **<numatune>** 相同的放置模式，当 **<cpuset>** 被指定时，则被默认为 *static*。

这意味着为了域进程而进行的 CPU 调试和内存调试可以被单独指定和定义，但是它们也可以在依赖其他放置模式的情况下进行配置。

也有一种可能，即通过 numad 来配置您的系统，可以启动选定数量的虚拟 CPU，并且在启动时不需要固定到所有的虚拟 CPU。

例如，通过 32 个虚拟 CPU 使仅有的 8 个虚拟 CPU 在一个系统中启动，配置 XML 方式与如下方式类似：

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



注意

更多关于虚拟 CPU 和 numatune 的信息请参照以下网址：

<http://libvirt.org/formatdomain.html#elementsCPUAllocation> 和
<http://libvirt.org/formatdomain.html#elementsNUMATuning>

8.4.3. 域虚拟 CPU 线程

除了调试域进程，libvirt 还允许 XML 配置中每个虚拟 CPU 线程的钉选策略设置。设置 **<cputune>** 标签内部每个虚拟 CPU 线程的钉选策略：

```
<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>
```

在这个标签中，libvirt 可以使用 cgroup 或 **sched_setaffinity(2)**，使虚拟 CPU 线程固定在指定的 cpuset 中。

**注意**

更多关于 **<cputune>** 的信息，请参照以下网址：
<http://libvirt.org/formatdomain.html#elementsCPUTuning>

8.4.4. 使用 **emulatorpin**

调试域进程钉选策略的另一个方法是使用 **<cputune>** 中的 **<emulatorpin>** 标签。

<emulatorpin> 标签指定了 *emulator*（一个域的子集，不包括虚拟 CPU）将被固定的主机物理 CPU。**<emulatorpin>** 标签提供了一个将精确关联设定成仿真线程进程的方法。因此，vhost 线程在同一个物理 CPU 和内存子集中运行，从而可以从缓存位置获益。例如：

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

**注意**

在 Red Hat Enterprise Linux 7 中，默认启用自动化 NUMA 平衡。随着 vhost-net 仿真线程能够更加可靠地跟随虚拟 CPU 任务，自动化 NUMA 平衡减少了手动调试 **<emulatorpin>** 的需要。关于自动化 NUMA 平衡的更多信息，请参照[第 8.3 节“自动化 NUMA 平衡”](#)。

8.4.5. 用 **virsh** 调试 vcpu CPU 钉选

**重要**

这些只是示例命令。您需要依据环境替换属性值。

以下示例 **virsh** 命令会将 ID 为 1 的虚拟 CPU 线程 (*rhel7*) 固定到物理 CPU 2。

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

您也可以通过 **virsh** 命令获得当前虚拟 CPU 的钉选配置。例如：

```
% virsh vcpupin rhel7
```

8.4.6. 用 **virsh** 调试域进程 CPU 钉选

**重要**

这些只是示例命令。您需要依据环境替换属性值。

emulatorpin 选项将 CPU 关联设置应用到与每个域进程关联的线程。为了完全固定，您必须给每个客机同时使用 **virsh vcpupin** (如之前所展示的) 和 **virsh emulatorpin**。例如：

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

8.4.7. 用 virsh 调试域进程内存策略

域进程内存可以动态调试。请参考以下示例指令：

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

关于这些指令的更多示例，请参见 **virsh** 手册页。

8.4.8. 客机 NUMA 拓扑

客机 NUMA 拓扑可以通过使用 **<cpu>** 标签中的 **<numa>** 标签在客机虚拟机的 XML 中进行指定。请参照以下示例，并替换相应的属性值：

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

每个 **<cell>** 元素指定一个 NUMA cell 或者 NUMA 节点。**cpus** 指定了 CPU 或者部分节点的系列 CPU，**memory** 以千位字节 (1,024 字节一块) 指定了节点内存。从 0 开始，**cellid** 或 **nodeid** 以递增的次序被指定到每个 cell 或节点。

8.4.9. 向多个客机 NUMA 节点指定主机大页面

在 Red Hat Enterprise Linux 7.1 中，主机中的大页面可以被分配到多个客机的 NUMA 节点。这一过程可以优化内存性能，因为客机 NUMA 节点可以被移动到主机 NUMA 节点需要的位置上，同时客机可以继续使用主机指定的大页面。

在配置客机 NUMA 节点拓扑后 (详情参考[第 8.4.8 节“客机 NUMA 拓扑”](#))，在客机 XML 的 **<memoryBacking>** 元素中指定大页面的大小和客机 NUMA 节点集。**page size** 和 **unit** 代表主机大页面的大小。**nodeset** 指定了大页面被分配的客机 NUMA 节点 (或若干节点)。

在以下示例中，客机 NUMA 节点 0-5 (不包括 NUMA 节点 4) 将会使用 1 GB 的大页面，客机 NUMA 节点 4 将使用 2 MB 的大页面，无论客机 NUMA 节点在主机的任何位置。为了在客机中使用 1 GB 的大页面，主机必须先启动 1 GB 大页面；启动 1 GB 大页面的方法请参考[第 7.3.3 节“大页面和透明大页面 \(THP\)”](#)。

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

当一些客机 NUMA 节点和单独主机 NUMA 节点进行合并起作用时，将允许对大页面控制的增强，但继续使用不同的大页面尺寸。例如，即使客机 NUMA 节点 4 和 5 被移动到了主机的 NUMA 节点 1 上，它们也将继续使用

用不同大小的大页面。



注意

当使用 **strict** 内存模式时，在 NUMA 节点上不具有足够可用的大页面的情况下，客机将无法启用。关于 `<numatune>` 标签中 **strict** 内存模式选项的配置示例，请参照[第 8.4.2 节“域进程”](#)。

8.4.10. PCI 设备的 NUMA 节点位置

当启动一个新的虚拟机时，了解主机 NUMA 拓扑和 NUMA 节点中的 PCI 设备归属是重要的一点，以便在请求 PCI 传递时，客机可以被固定在正确的 NUMA 节点以优化其内存性能。

例如，如果客机被固定在 NUMA 节点 0-1 上，但是其 PCI 设备中的一个隶属于节点 2，那么节点之间的数据传输将花费一段时间。

在 Red Hat Enterprise Linux 7.1 中，libvirt 在客机 XML 中为 PCI 设备报道了 NUMA 节点位置，使管理应用程序完成更好的性能决策。

该信息在 `/sys/devices/pci*/*/numa_node` 的 **sysfs** 文件中可见。使用 **lstopo** 工具来回报 **sysfs** 数据，可以作为验证这些设置的一种方法。

```
# 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"

```

此结果表明：

- * NIC **em*** 与磁盘 **sd*** 是与 NUMA 节点 0 和 cores 0、2、4、6、8、10、12、14 连接的。
- * NIC **p1*** 与 **ib*** 是与 NUMA 节点 1 和 cores 1、3、5、7、9、11、13、15 连接的。

8.5. NUMA-Aware 内核同页合并

内核同页合并 (KSM, Kernel SamePage Merging) 允许虚拟机共享相同的内存页。KSM 可以探测出正在使用 NUMA 内存的系统并控制不同 NUMA 节点中的页面合并。

使用 `sysfs /sys/kernel/mm/ksm/merge_across_nodes` 参数来控制不同 NUMA 节点中的页面合并。在默认情况下，所有节点的页面都可以进行合并。当该参数被设置为 0 时，只有来自同一个节点的页面可以合并。

通常，除了系统内存过量的情况下，禁用 KSM 共享可以带来更好的运行性能。



重要

当 KSM 通过多个客机虚拟机在 NUMA 主机上的节点进行合并时，远端节点的客机和 CPU 在合并 KSM 页面的访问延迟将显著增加。

通过向客机 XML 添加以下命令，对虚拟机监控程序和客机共享页面的禁用进行指示：

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

更多关于使用 `<memoryBacking>` 元素调试内存设置的信息，请参照[第 7.3.2 节“使用 virsh 调试内存”](#)。

附录 A. 修订历史

修订 1.0-14.1	Mon Mar 14 2015	Chester Cheng
说明：7.1 版翻译、校对完成。 翻译、校对：陈西子。 校对、责任编辑：鄭中。 附注：本简体中文版来自「红帽公司·全球服务部」与「澳大利亚昆士兰大学·笔译暨口译研究生院」之产学合作计划。若有疏漏之处，盼各方先进透过以下网址，给予支持指正： https://bugzilla.redhat.com/ 。		
修订 1.0-14	Fri Feb 27 2015	Dayle Parker
更新的大页面章节基于 BZ#1134744 的 SME 反馈。		
修订 1.0-13	Wed Feb 25 2015	Dayle Parker
更新的大页面章节基于 BZ#1134744 的 SME 反馈。 7.1 GA 更新版本发布。		
修订 1.0-12	Wed Feb 18 2015	Dayle Parker
7.1 GA 版本发布。		
修订 1.0-11	Tue Feb 17 2015	Dayle Parker
为 BZ#1134746 的 PCI 设备区域增加 NUMA 节点局部性。 1GB 大页面文本的细微修改。		
修订 1.0-10	Fri Feb 13 2015	Dayle Parker
关于 BZ#1134660 严格页面规模的补充注释。 关于在大页面章节启动 1GB 大页面和 BZ#1134744 NUMA 大页面区域的补充注释。		
修订 1.0-09	Wed Jan 21 2015	Scott Radvan
修复 BZ#1183951 中 sysctl 的命令句法。		
修订 1.0-08	Tue Jan 13 2015	Scott Radvan
发布指南的更新版本。		
修订 1.0-07	Mon Jan 12 2015	Scott Radvan
现在调试的软件包默认安装并启动服务；删除 yum/systemctl 指令。参见 BZ#1096788		
修订 1.0-06	Mon Dec 8 2014	Scott Radvan
更新并在大字页面中执行新的 sort_order。		
修订 1.0-05	Thurs Dec 4 2014	Dayle Parker
RHEL 7.1 Beta 发布。		
修订 1.0-04	Fri Nov 28 2014	Dayle Parker
编辑 BZ#1134665 大页面区域的参考，分配给多个客户的 NUMA 节点		
修订 1.0-03	Thurs Nov 27 2014	Dayle Parker
编辑 BZ#1134665 内容，分配大页面给多个客户的 NUMA 节点		
修订 1.0-02	Wed Nov 26 2014	Dayle Parker

第一章支持信息更新的网址。

添加 [BZ#1134665](#) 内存章节的段落和示例，分配大页面给多个客户的 NUMA 节点重新部署章节顺序，使监控工具在前端，组合内存调试和 NUMA 调试（相关）。

修订 0.1-53	Mon Nov 17 2014	Dayle Parker
在 BZ#1132234 添加关于调整后功能性的注解		
修订 0.1-52	Tues Sept 9 2014	Dayle Parker
在 BZ#1098010 添加关于 KSM 部分系统内存过量的注解		
修订 0.1-51	Fri Aug 1 2014	Dayle Parker
调整章节中客户门户网站更新。		
修订 0.1-50	Tues June 3 2014	Dayle Parker
7.0 GA 发行版本		
修订 0.1-49	Tues June 3 2014	Dayle Parker
虚拟机管理器章节的屏幕截图更新。 BZ#1064612 第一章支持限制网址更新。 添加关于内存章节 nosharepages 参数的注解，获得 BZ#1064616 的 QE 反馈。		
修订 0.1-48	Fri May 23 2014	Dayle Parker
修改 NUMA 章节中的术语和客户名称，获得 BZ#1064618 的 QE 反馈。 编辑 blkio tune 命令，在 Block I/O 章节中添加 "live" 和 "current" 选项，获得 BZ#1064617 的 QE 反馈。 编辑 tuned-adm 列表命令输出，在调整章节的 tuned.conf 手册页中添加参考，获得 BZ#1096793 和 BZ#1096795 的 QE 反馈。		
修订 0.1-47	Tues May 20 2014	Dayle Parker
〈简介〉章节中的支持限制网址更新。		
修订 0.1-46	Mon May 19 2014	Dayle Parker
从简介章节和序言资料中移除文档集列表。		
修订 0.1-45	Fri May 9 2014	Dayle Parker
为风格更改进行重建。		