第一篇 Part 1

KVM 技术详解与实践

Chapter 1 第 1 章

企业虚拟化选型与 KVM 介绍

虚拟化技术从 2008 年开始越来越热，经过一个大爆发的阶段，目前已经是企业 IT 环境的必备技术，在许多企业里面，虚拟机的数量已经远远大于物理机。同许多技术一样，虚拟化也分为开源和闭源技术。KVM 就是一种开源的虚拟化技术，本章将为读者介绍 KVM 技术的演进过程、KVM 的发展历史及 KVM 的应用场景。

1.1 KVM 的前世今生

如图 1-1 所示，根据 OpenStack 平台上 2013 年 10 月虚拟化引擎选择的调查统计数据，KVM 已经在 OpenStack 平台占到 71% 的份额。

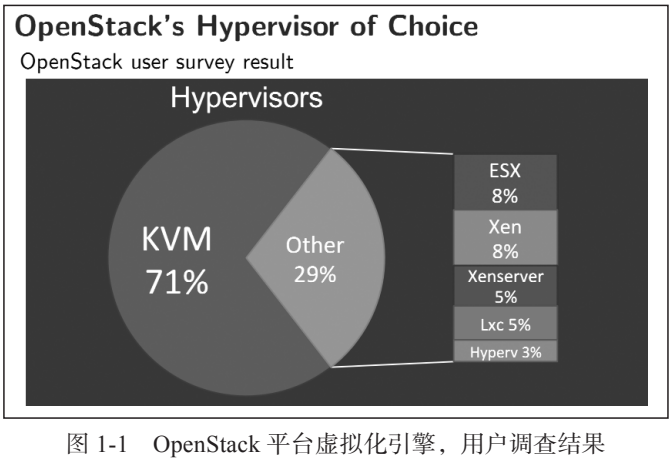


图 1-1　OpenStack 平台虚拟化引擎，用户调查结果

KVM 是一种年轻的虚拟化技术，在出生的时候就吸取了其他虚拟化技术的优点，所以 KVM 的架构简单，没有历史兼容性的包袱，性能表现优异。

本节介绍一下 KVM 的演进过程。

1. 虚拟化技术的演变过程

虚拟化技术的演变过程可以分为软件模拟、虚拟化层翻译、容器虚拟化三个大的阶段。

其中，虚拟化层翻译又可以分为：

- ❑ 软件捕获翻译，即软件全虚拟化。
- ❑ 改造虚拟机系统内核加虚拟化层翻译，即半虚拟化。
- ❑ 硬件支持的虚拟化层翻译，即硬件支持的全虚拟化。

下面分别介绍一下这几种技术方式。

（1）软件模拟的技术方式

软件模拟是通过软件完全模拟 CPU、芯片组、磁盘、网卡等计算机硬件，如图 1-2 所示。

图 1-2　QEMU 的虚拟机架构

因为是软件模拟，所以理论上可以模拟任何硬件，甚至是不存在的硬件。但是因为这种方式全部是软件模拟硬件，所以非常低效，一般只用于研究测试的场景。采用这种技术的典型产品有 Bochs、QEMU 等。

（2）虚拟化层翻译

先介绍一下 X86 平台的指令集权限划分。如图 1-3 所示，X86 平台指令集划分为 4 个特权模式：Ring 0、Ring 1、Ring 2、Ring 3。操作系统一般使用 Ring 0 级别，应用程序使用 Ring 3 级别，驱动程序使用 Ring 1 和 Ring 2 级别。X86 平台在虚拟化方面的一个难点就是**如何将虚拟机越级的指令使用进行隔离**。

VMware 公司找到了最早的解决方法，图 1-4 所示为对虚拟机指令的使用进行捕获和翻译的示意图。

图 1-3 X86 CPU 指令级别

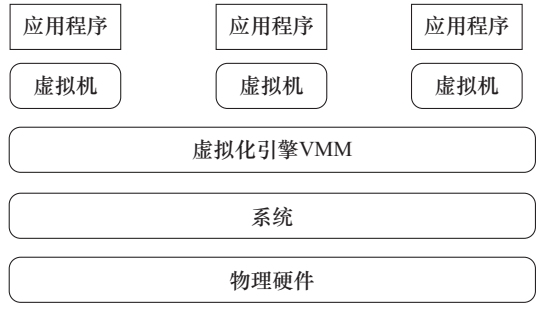


图 1-4 软件全虚拟化方案架构

通过虚拟化引擎，捕获虚拟机的指令，并进行处理，这也是为什么在虚拟机上虽然使用的是物理机一样的指令，但是虚拟机不能对硬件进行操作的原因，比如重启虚拟机不会引起宿主机的重启。这种解决方案也叫软件全虚拟化方案。

（3）改造虚拟机操作系统的方式

通过虚拟化引擎进行指令捕获和翻译的方式虽然可行，但是在虚拟化层要做大量的工作，Xen 项目提供了另外一种思路，就是对虚拟机的操作系统内核进行改造，使虚拟机自己对特殊的指令进行更改，然后和虚拟化层一起配合工作，这也是 Xen 早期一直要使用一个特殊内核的原因，并且不支持 Windows 系统虚拟化。改造的虚拟机虽然使用上有限制，配置比较麻烦，但是这种方式效率非常高，这种方式也被称为半虚拟化方案。

（4）对 CPU 指令进行改造

2005 年，Intel 推出了硬件的方案，对 CPU 指令进行改造，即 VT-x，如图 1-5 所示。VT-x 增加了两种操作模式：VMX root operation 和 VMX non-root operation。VMM 运行在 VMX root operation 模式，虚拟机运行在 VMX non-root operation 模式。这两种操作模式都支持 Ring 0 ~ Ring 3 这 4 个特权级。

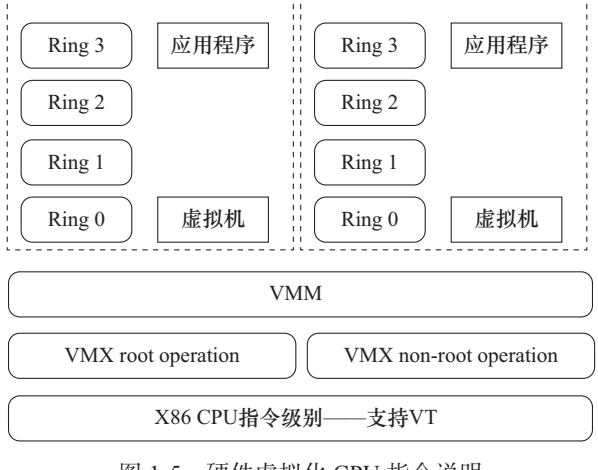


图 1-5 硬件虚拟化 CPU 指令说明

这种方案因为是基于硬件的，所以效率非常高，这种方案也称为硬件支持的全虚拟化方案，如图 1-6 所示。现在的一个发展趋势是不仅 CPU 指令有硬件解决方案，I/O 通信也有硬件解决方案，称为 VT-d；网络通信也有硬件解决方案，称为 VT-c。

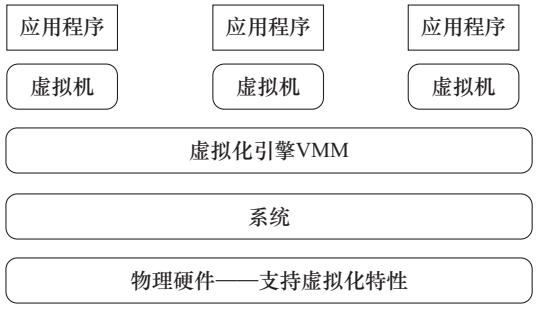


图 1-6　硬件全虚拟化方案架构

当前的虚拟化引擎，都是使用硬件支持的虚拟化解决方案。并且最新的操作系统一般都支持一些半虚拟化的特性，所以宿主机和虚拟机使用比较新的版本，性能也会好一些。

（5）容器虚拟化

容器虚拟化的原理是基于 CGroups、Namespace 等技术将进程隔离，每个进程就像一台单独的虚拟机一样，有自己被隔离出来的资源，也有自己的根目录、独立的进程编号、被隔离的内存空间。基于容器的虚拟化可以实现在单一内核上运行多个实例，因此是一个更高效率的虚拟化方式。目前最热的容器虚拟化技术就是 Docker。Docker 的优势是可以将一个开发环境进行打包，很方便地在另外一个系统上运行起来，并且有版本的概念，可以在前一个版本的基础上累加。但是 Docker 在生产环境的使用还需要一个过程，主要是磁盘、网络性能上还受到很多限制。

2. KVM 的历史

KVM（Kernel-based Virtual Machine）最初是由以色列的公司 Qumranet 开发的。KVM 在 2007 年 2 月被正式合并到 Linux 2.6.20 核心中，成为内核源代码的一部分。2008 年 9 月 4 日，RedHat 公司收购了 Qumranet，开始在 RHEL 中用 KVM 替换 Xen，第一个包含 KVM 的版本是 RHEL 5.4。从 RHEL 6 开始，KVM 成为默认的虚拟化引擎。KVM 必须在具备 Intel VT 或 AMD-V 功能的 X86 平台上运行。它也被移植到 S/390、PowerPC 与 IA-64 平台上。在 Linux 内核 3.9 版中，加入了对 ARM 架构的支持。

KVM 包含一个为处理器提供底层虚拟化、可加载的核心模块 kvm.ko（kvm-intel.ko 或 kvm-amd.ko），使用 QEMU（QEMU-KVM）作为虚拟机上层控制工具。KVM 不需要改变

Linux 或 Windows 系统就能运行。

3. KVM 的架构

KVM 的架构非常简单，如图 1-7 所示，KVM 就是内核的一个模块，用户空间通过 QEMU 模拟硬件提供给虚拟机使用，一台虚拟机就是一个普通的 Linux 进程，通过对这个进程的管理，就可以完成对虚拟机的管理。实际上德国有家公司开发了一个管理平台 Proxmox VE，就是通过对 KVM 进程的管理来实现对虚拟机的管理的。

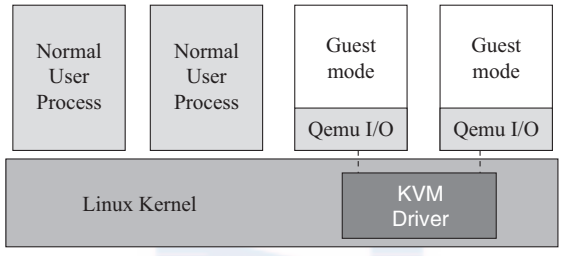


图 1-7 KVM 的架构

因为对进程的管理非常麻烦，RedHat 发布了一个开源项目 Libvirt。Libvirt 有 API，也有一套命令行工具，可以完成对虚拟机的管理，大多数的管理平台都是通过 Libvirt 来完成对 KVM 虚拟机的管理的，比如 OpenStack、CloudStack、OpenNebula 等。

4. QEMU 与 KVM

QEMU 是一个开源项目，实际就是一台硬件模拟器，可以模拟许多硬件，包括 X86 架构处理器、AMD64 架构处理器、MIPS R4000、ARM v6/v7（Cortex-A8，A9，A15）、SPARC sun3 与 PowerPC 架构，还支持其他架构，可以从 QEMU 主页获取完整的列表。

QEMU 可以在其他平台上运行 Linux 的程序，可以存储及还原虚拟机运行状态，还可以虚拟多种设备，包括网卡、多 CPU、IDE 设备、软驱、显卡、声卡、多种并口设备、多种串口设备、多种 USB 设备、PC 喇叭、PS/2 键盘鼠标（默认）和 USB 键盘鼠标、蓝牙设备。

QEMU 还内建 DHCP、DNS、SMB、TFTP 服务器。

比较有意思的是，有人还将 QEMU 编译成 Windows 版本，在 Windows 平台上也可以运行 QEMU。

QEMU 的好处是因为是纯软件模拟，所以可以在支持的平台模拟支持的设备，比如还有人利用 QEMU 在安卓上安装一个 Windows XP 虚拟机出来。

QEMU 的缺点是因为是纯软件模拟，所有非常慢。QEMU 1.0 的时候有一个 QEMU 和 KVM 结合的分支。KVM 只是一个内核的模块，没有用户空间的管理工具，KVM 的虚拟机可以借助 QEMU 的管理工具来管理。QEMU 也可以借助 KVM 来加速，提升虚拟机的性能。QEMU-KVM 的分支版本发布了 3 个正式的版本 1.1、1.2、1.3，随后和 QEMU 的主版本合并，也就是说现在的 QEMU 版本默认支持 KVM，QEMU 和 KVM 已经紧密地结合起来了。

KVM 的最后一个自己的版本是 KVM 83，随后和内核版本一起发布，和内核版本号保持一致，所以要使用 KVM 的最新版本，就要使用最新的内核。

5. Libvirt 与 KVM

Libvirt 是一套开源的虚拟化的管理工具，主要由 3 部分组成：

- ❑ 一套 API 的 lib 库，支持主流的编程语言，包括 C、Python、Ruby 等。
- ❑ Libvirtd 服务。
- ❑ 命令行工具 virsh。

Libvirt 的设计目标是通过相同的方式管理不同的虚拟化引擎，比如 KVM、Xen、HyperV、VMware ESX 等。但是目前实际上多数场景使用 Libvirt 的是 KVM，而 Xen、HyperV、VMware ESX 都有各自的管理工具。

Libvirt 可以实现对虚拟机的管理，比如虚拟机的创建、启动、关闭、暂停、恢复、迁移、销毁，以及虚拟机网卡、硬盘、CPU、内存等多种设备的热添加。

Libvirt 还支持远程的宿主机管理，只要在宿主机上启动 Libvirtd 服务并做好配置，就可以通过 Libvirt 进行虚拟机的配置。通道可以是以下方式：

- ❑ SSH。
- ❑ TCP。
- ❑ 基于 TCP 的 TLS。

Libvirt 将虚拟机的管理分为以下几个方面：

第一，存储池资源管理，支持本地文件系统目录、裸设备、lvm、nfs、iscsi 等方式。在虚拟机磁盘格式上支持 qcow2、vmdk、raw 等格式。

第二，网络资源管理，支持 Linux 桥、VLAN、多网卡绑定管理，比较新的版本还支持 Open vSwitch。Libvirt 还支持 nat 和路由方式的网络，Libvirt 可以通过防火墙让虚拟机通过宿主机建立网络通道，和外部的网络进行通信。

1.2　KVM 与常用企业级虚拟化产品的 PK

1. 常用企业级虚拟化产品的比较

目前常见的企业级的虚拟化产品有 4 款：分别是 VMware、HyperV、Xen、KVM。

（1）VMware

VMware 是最早的 X86 平台上的虚拟化引擎，1999 年就发布了第一款产品，经过十几年的发展和市场检验，产品成熟、稳定，兼容性也不错。VMware 的产品线非常全面，不仅有虚拟化的解决方案，在 IaaS、SaaS、PaaS 层都有自己的产品。并且 VMware 在网络、存储方面都有相关的解决方案，VMware 和网络存储厂商在协议层面也有一些私有协议，许多主流的厂商都支持 VMware 一些专用的协议，和 VMware 一起形成了一个生态链。

VMware 目前被 EMC 控股，虚拟化产品线主要有针对个人使用的 VMware Workstation，针对苹果用户的 VMware Fusion，针对企业级用户的 VMware ESXi 服务器。管理工具主要是 VMware vSphere 套件。

VMware 的产品基本上都是非开源产品，并且大部分都是收费产品，一般在传统关键行业使用比较多一些，在中小型企业、互联网行业使用得比较少一点。

（2）HyperV

HyperV 是微软的虚拟化产品，最近几年发展非常迅速，在 Windows Server 2012 R2 中的 HyperV 支持许多非常新的虚拟化特性。HyperV 必须使用 64 版的 Windows 产品，HyperV 也支持 Linux 系统的虚拟机。

HyperV 也是一款非开源的收费产品，HyperV 的集群管理工具 SCVMM 配置非常复杂，需要配置 Windows 域、Windows Server 集群，然后才能管理多台宿主机。因为 HyperV 的成本相对比较低，所以最近几年市场占有率也在提升，主要是一些使用 Windows 系统的企业使用比较多。

（3）Xen

Xen 是最早的开源虚拟化引擎，由剑桥大学开发，半虚拟化的概念也是 Xen 最早提出的。Xen 后来被思杰收购，推出了一套叫作 XenServer 的管理工具，XenServer 于 2013 年年底宣布免费。Xen 因为推出的时间比较长，兼容性、稳定性都不错，目前使用 Xen 的主要是一些在 Xen 上面技术积累较多的企业。

（4）KVM

KVM 比较年轻，所以出生的时候就吸取了其他虚拟化技术的优点，一开始就支持硬件虚拟化技术，没有历史兼容包袱。所以 KVM 推出来的时候，性能就非常优异。目前，KVM 是 OpenStack 平台上首选的虚拟化引擎。国内新一代的公有云全部采用 KVM 作为底层的虚拟化引擎。KVM 已经成为开源解决方案的主流选择。

2. KVM 优势

KVM 的优势主要体现在以下几点。

（1）开源

KVM 是一个开源项目，这就决定了 KVM 一直是开放的姿态，许多虚拟化的新技术都是首先在 KVM 上应用，再到其他虚拟化引擎上推广。

虚拟化一般网络和存储都是难点。网络方面，SRIOV 技术就是最先在 KVM 上先有应用，然后再推广到其他虚拟化引擎上。再比如 SDN、Open vSwtich 这些比较新的技术，都是先在 KVM 上得到应用。

磁盘方面，基于 SSD 的分层技术，都是最早在 KVM 上得到应用。

KVM 背靠 Linux 这棵大树，和 Linux 系统紧密结合，在 Linux 上的新技术都可以马上应用到 KVM 上。围绕 KVM 的是一个开源的生态链，从底层的 Linux 系统，到中间层的

Libvirt 管理工具，到云管理平台 OpenStack，莫不是如此。

（2）性能

KVM 吸引许多人使用的一个动因就是性能，在同样的硬件条件下，能提供更好的虚拟机性能，主要是因为 KVM 架构简单，代码只有 2 万行，一开始就支持硬件虚拟化，这些技术特点保证了 KVM 的性能。

（3）免费

KVM 因为是开源项目，绝大部分 KVM 的解决方案都是免费方案，随着 KVM 的发展，KVM 虚拟机越来越稳定，兼容性也越来越好，因而也就得到越来越多的应用。

（4）广泛免费的技术支持

免费并不意味着 KVM 没有技术支持。在 KVM 的开源社区，数量巨大的 KVM 技术支持者都可以提供 KVM 技术支持。另外，如果需要商业级支持，也可以购买红帽公司的服务。

1.3　判断企业是否适合使用 KVM 的标准

1. 业务类型

目前在互联网行业，KVM 虚拟化技术是使用最广泛的，因为互联网行业有对新技术追求的冲动。另外中小企业也推荐使用 KVM 技术，因为使用 KVM 不需要支付额外的费用。

还有哪些企业、个人适合使用 KVM 呢？可以这样总结：KVM 天生就是为 Linux 而生的，凡是可以使用 Linux 的地方，就可以使用 KVM，KVM 天生和 Linux 在一起。

2. 企业对成本的关注度

随着虚拟化这几年的快速发展，在企业的 IT 环境中，虚拟化已经是一个标配的技术。虚拟化带来的好处是巨大的，虚拟化真正实现了资源池化，通过虚拟化可以将服务器资源进行切割，做到资源随取随用，有效节省成本，提高资源利用率。

KVM 虚拟化技术经过几年的发展，已经非常成熟，使用 KVM 技术更不需要在虚拟化方面支付额外的费用，可以进一步节省企业的 IT 成本。

3. 企业对快速部署的关注度

虚拟化还有一个重要的功能就是快速部署。在宿主机层面看，虚拟机就是一个镜像文件，要得到另外一台虚拟机，只需要将镜像文件复制一份就可以了，通常只有几分钟。而按照传统方式部署一台物理机，最起码都要一个小时。通过磁盘差量的技术，甚至可以做到秒级生成虚拟机。

通过虚拟化技术，还可以实现在虚拟化层做高可用和在线迁移。虚拟化层的高可用是系统层面的高可用，比基于应用层的高可用配置要简单很多。虚拟机的在线迁移，更是虚拟化技术的独有手段。通过在线迁移，物理机的维护、系统的维护、网络的维护，都可以做到不中断服务，进一步提高了业务应用的可用性。

因为所有的虚拟机都是相同的虚拟硬件，实现了硬件层级的标准化，降低了自动化的难度，很容易搭建私有或者公有的云平台，所以通过虚拟机技术，很容易实现以下功能：

- 资源使用自动化和自助服务。
- 资源池可随时扩展。
- 资源使用标准化。
- 资源使用保持了很好的兼容性。

1.4 本章小结

本章介绍了虚拟化及 KVM 技术的演进过程，介绍了 KVM 技术的优势，可以简单地概括：凡是使用 Linux 系统的场景，如果上虚拟机项目，就天然适合使用 KVM 虚拟化技术。

下一章将介绍如何开始自己的第一台虚拟机，在安装第一台虚拟机的时候有哪些地方应该注意。

第 2 章 Chapter 2

开始自己的第一台虚拟机

通过第 1 章对 KVM 的发展历史、架构及应用场景的介绍，读者对 KVM 技术有了初步的了解和认识。本章将介绍如何创建一台 KVM 虚拟机、操作步骤及要点。对于熟悉或者使用过 KVM 虚拟化的读者，本章内容可以快速阅读或者跳过。

2.1 服务器 BIOS 设置

KVM 的使用必须有硬件虚拟化支持，所以需要打开 CPU 的硬件虚拟化特性。对于大多数服务器，如 Dell、HP、IBM、浪潮、联想、华为等，在开机启动的第一个画面都会有一些提示的按键，图 2-1 所示是 Dell R610 开机启动的第一个画面。

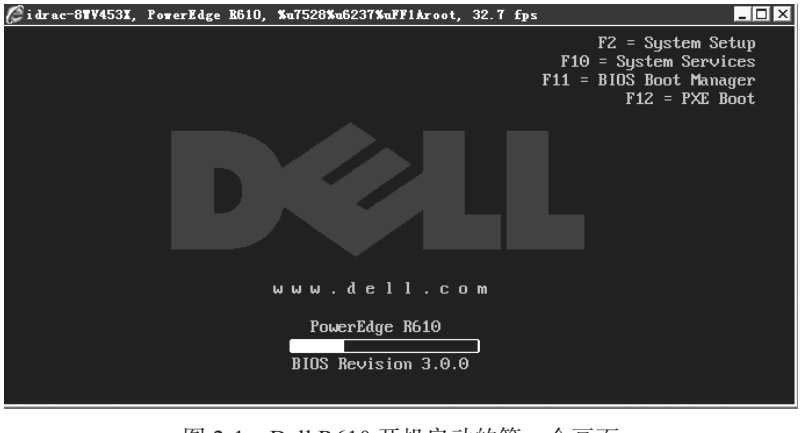


图 2-1 Dell R610 开机启动的第一个画面

该界面提供了如下几个选项：

- 按 F2 键进入 System Setup，进去之后可以进行一些 BIOS 相关的配置。
- 按 F10 键进入 System Services，进去之后可以配置一些系统服务，主要用于服务器的配置和检测。
- 按 F11 键进入 BIOS Boot Manager，进入之后可以选择启动介质，比如从硬盘启动、虚拟 DVD 设备、USB 设备，等等。
- 按 F12 键进入 PXE，进入之后启动网络引导。

提示 不同的厂商服务器，功能键的定义会有一些差别，可以根据提示来操作。

在安装 KVM 虚拟化之前，需要先确认 CPU 虚拟化支持是否开启。在如图 2-1 所示的服务器开机界面中按 F2 键，然后进入 BIOS 配置，选择 Processor Setting，按 Enter 键进入子菜单，其中一项 Virtualization Technology 必须配置为 Enabled（选择菜单项，按方向键右键可更改配置），如图 2-2 所示。

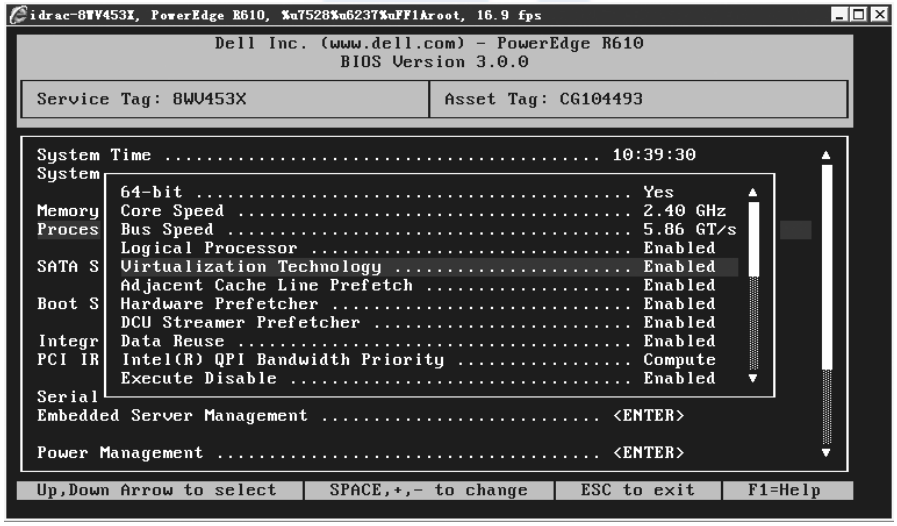


图 2-2 开启 CPU 虚拟化支持

提示 不同厂商的服务器，CPU Virtualization Technology 开关的位置会有些差别，一般都在处理器配置菜单之下。近年来服务器上的 CPU 一般都支持 CPU 虚拟化，在系统中可以执行 egrep '(vmx|svm)' /prov/CPUinfo 命令查看，如果有输出内容，说明 CPU 是支持虚拟化的。

2.2　宿主机 CentOS 6.5、CentOS 7 系统安装与配置技巧

通过 2.1 节的配置，已经打开了服务器 CPU 对虚拟化的支持，下面开始安装宿主机的操作系统。CentOS 6.x 是目前使用比较多的宿主机操作系统版本，当前最新版本是 CentOS 6.6。CentOS 7 从系统跨度来说，是 CentOS 的一个大版本升级，其中主要包括内核版本的跨越。CentOS 6.x 系列使用的是 2.6.32.x 的内核，CentOS 7.x 则使用 3.10.x 的内核。

1. CentOS 6.5 宿主机系统安装及配置

在生产环境中，安装 CentOS 6.5 系统的宿主机，笔者采用 pxe 方式来完成批量的宿主机部署。pxe 配置过程的资料很多，本书就不做详细介绍了，下面分享一个宿主机安装 kickstart 文件的内容，是笔者实际在生产环境中使用的。

```
# Kickstart file automatically generated by anaconda
# 系统安装自动生成的 kickstart 文件
install
url --url=http://10.10.10.1/system/CentOS6564
# 指定安装镜像的目录，可以将不同的系统放在不同的目录，这样可以完成多个系统安装
lang en_US.UTF-8
# 宿主机建议使用英文
keyboard us
network --onboot yes --device eth0 --mtu=1500 --bootproto dhcp
network --onboot yes --device eth1 --noipv4 --noipv6
# 设置网络，笔者装机的时候，一般第一个网卡使用 dhcp 分配 IP，第二个网卡关闭。读者可以根据自己的
  实际情况进行修改
rootpw  cEmXc2pkKets    # 配置系统密码
text
reboot
# 文本方式安装，安装后重启
firewall --disabled
authconfig --useshadow  --passalgo=sha512
# 用户密码加密 seLinux --disabled  # 关闭 SELinux
##################################################
# Installation logging level
logging --level=info
# Do not configure the X Window System
skipx
##################################################
timezone Asia/Shanghai  # 配置时区
bootloader --location=mbr --driveorder=sda --append="crashkernel=auto rhgb quiet"
clearpart --all --initlabel  # 清除硬盘数据，并创建相应分区
part /boot --fstype ext3 --size=256
part pv.3 --size=40960
part pv.4 --size=100 --grow
volgroup datavg --pesize=32768 pv.4
volgroup KVMvg --pesize=32768 pv.3
logvol /datapool --fstype ext4 --name=datapool --vgname=datavg --size=10240
--grow  # 此处创建了一个目录，用于存放虚拟机，笔者习惯的目录是 /datapool
```

```
logvol swap --fstype swap --name=swap --vgname=KVMvg --size=8192
logvol / --fstype ext4 --name=root --vgname=KVMvg --size=30720
%packages
# 安装虚拟化需要的包，主要有：
@virtualization
@Base
@Core
@additional-devel
@base
@large-systems
@storage-client-iscsi
@system-management-snmp
@virtualization
@virtualization-client
@virtualization-platform
@virtualization-tools
#########################################################
%end
```

使用上面的 kickstart 配置文件安装完宿主机系统，可以通过 rpm -q 命令检查一下具体安装了哪些包。

```
[root@localhost ~]# rpm -qa|grep -E 'qemu|libvirt|virt'
libvirt-client-0.10.2-29.el6.x86_64
#Libvirt 的客户端，最重要的功能之一就是就在宿主机关机时可以通知虚拟机也关机，
# 使虚拟机系统正常关机，而不是被强制关机，造成数据丢失
gpxe-roms-qemu-0.9.7-6.10.el6.noarch # 虚拟机 iPXE 的启动固件，支持虚拟机从网络启动
libvirt-python-0.10.2-29.el6.x86_64   #libvirt 为 Python 提供的 API
python-virtinst-0.600.0-18.el6.noarch  # 一套 Python 的虚拟机安装工具
qemu-KVM-0.12.1.2-2.415.el6.x86_64 #KVM 在用户空间运行的程序
Virt-manager-0.9.0-19.el6.x86_64  # 基于 Libvirt 的图像化虚拟机管理软件
libvirt-0.10.2-29.el6.x86_64   # 用于管理虚拟机，它提供了一套虚拟机操作 API
virt-viewer-0.5.6-8.el6.x86_64  # 显示虚拟机的控制台 console
virt-top-1.0.4-3.15.el6.x86_64  # 类似于 top 命令，查看虚拟机的资源使用情况
virt-what-1.11-1.2.el6.x86_64  # 在虚拟机内部执行，查看虚拟机运行的虚拟化平台
qemu-img-0.12.1.2-2.415.el6.x86_64  # 用于操作虚拟机硬盘镜像的创建、查看和格式转化
```

在 KVM 环境中，以上 rpm 包都是必需的。读者可以检查当前系统里面是否已经安装 rpm，如果没有，可使用“yum install 组件名称”命令直接安装。

2. CentOS 7 宿主机系统安装及配置

CentOS 7 宿主机的安装与 CentOS 6 的系统安装方式类似，系统安装完之后，确认已经安装了如下的 rpm 相关包。Libvirt 还包含了很多工具的库，可以使用 yum install libvirt* 命令安装。

```
[root@KVM-host-CentOS7 ~]# rpm -qa|grep -E 'qemu-img|libvirt-[0-9]|virt-install'
qemu-img-1.5.3-60.el7_0.10.x86_64
virt-install-0.10.0-20.el7.noarch
libvirt-1.1.1-29.el7_0.3.x86_64
```

```
[root@KVM-host-CentOS7 ~]# lsmod |grep KVM  # 查看 KVM 模块是否载入
KVM_intel 138567  6
KVM 441119  1 KVM_intel
```

Libvirt 及 guestfish 相关的工具在平时的运维过程中经常会用到，建议使用 yum install libguest* libvirt* 命令安装。

2.3　第一台虚拟机安装

本节介绍如何开始安装第一台虚拟机。首先需要新建一台虚拟机，然后通过诸如物理机安装的方式为虚拟机安装操作系统。但是一般不会使用 pxe、cobbler 等网络引导方式安装，因为这样安装虚拟机系统太慢了。虚拟化相比于物理机，其中一个优势就是创建快速。所以，一般都会使用 ISO 镜像文件安装第一台虚拟机，然后将这台虚拟机做成虚拟机模板，之后的虚拟机都是由这个模板生成的。后面第 16 章会详细介绍虚拟机模板的创建。本节将介绍两个 KVM 虚拟化中常用的管理工具，用它们来创建虚拟机。

模板的概念在虚拟化中非常重要，实际上绝大多数虚拟机都是通过模板克隆出来的，而不是像物理机那样安装出来的。有的云管理平台还有镜像市场的概念，就像应用商店，可以上传、下载各种模板。

1. Virt-Manager 使用介绍

Virt-Manager 是一个图形化的虚拟机管理工具，它提供了一个简易的虚拟机操作界面。要使用它，需要先安装图形化界面，下面以 CentOS 6.5 系统为例，需要安装如下组件。

```
[root@KVM-host ~]#yum groupinstall -y "Desktop" "Desktop Platform" "Desktop
Platform Development" "Fonts" "General Purpose Desktop" "Graphical Administration
Tools" "Graphics Creation Tools" "Input Methods" "X Window System" "Chinese Support
[zh]" "Internet Browser"
```

一般来说，服务器都是在 IDC 机房中的，为了看到宿主机的图形化界面，还需要安装配置 VNC。

```
[root@KVM-host ~]#yum install -y tigervnc
[root@KVM-host ~]#yum install -y tigervnc-server
 <!-- 安装 VNC 程序软件包 tigervnc 和 tigervnc-server  -->
[root@KVM-host ~]#vim /etc/sysconfig/vncservers
VNCSERVERS="1:root" # 配置宿主机 VNC 虚拟机显示器为 1，端口是 5901
VNCSERVERARGS[2]="-geometry 800x600 -nolisten tcp -localhost"
# 虚拟机显示器 [2] 监听的 IP 是 0.0.0.0，虚拟显示器 [1] 监听 127.0.0.1，屏幕分辨率为 800×600
[root@KVM-host ~]# vncpasswd  # 设置 VNC 密码
Password:
Verify:
```

```
[root@KVM-host ~]# service vncserver restart  #启动 vnc-server
Shutting down VNC server:                                  [  OK  ]
Starting VNC server: 1:root xauth:  creating new authority file /root/.Xauthority
New 'whcq-netinst-121:1 (root)' desktop is KVM-host:1
Creating default startup script /root/.vnc/xstartup
Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/ KVM-host:1.log
                    [  OK  ]
```

配置完成之后，使用 VNC Viewer 登录宿主机的图形界面，在 VNC Server 组合框中输入宿主机的“IP: 端口”，端口是之前配置的，本例中输入 192.168.106.221:5901，如图 2-3 所示。

提示 VNC Viewer 使用中，有时候会出现 VNC Viewer 界面闪退，可以通过修改网络速度解决这个问题。在如图 2-4 所示的位置，取消勾选该复选框，调节滑块到 Best quality。

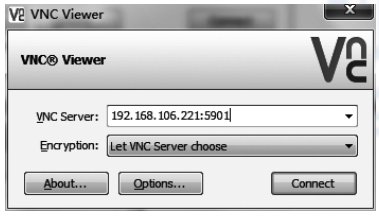


图 2-3 使用 VNC Viewer 登录宿主机图形界面

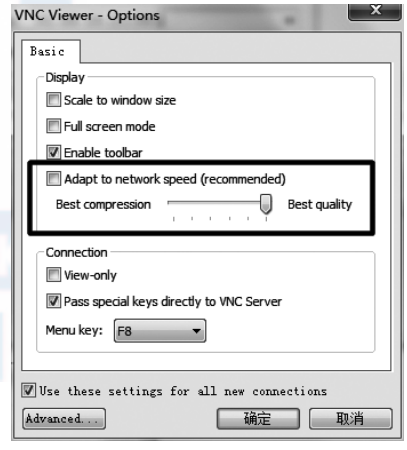


图 2-4 VNC Viewer 配置网络速率

VNC Viewer 配置完成之后，回到图 2-3 所示的界面，单击 Connect 按钮，出现如图 2-5 所示界面，提示输入密码，此处的密码是之前通过 vncpasswd 命令设置的 VNC Server 密码。

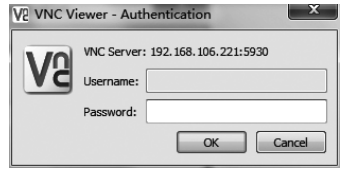


图 2-5 VNC Viewer 登录界面

输入正确密码之后，应该就可以登录成功了，出现如图 2-6 所示界面。

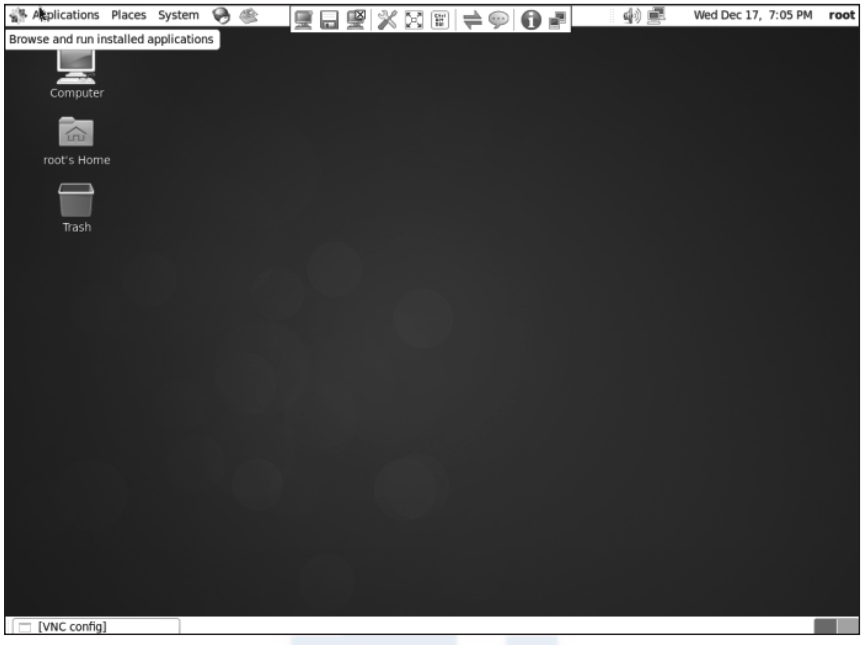


图 2-6　宿主机 VNC 登录之后界面

在菜单中，选择 Applications 菜单中的 System Tools 子菜单，选择打开 Virtual Machine Manager，如图 2-7 所示。其中列出了当前宿主机上所有的虚拟机，并显示了简单的 CPU 利用率信息。在图形界面中可以对虚拟化做常规操作，如创建、删除、编辑、配置及查看远程终端等。

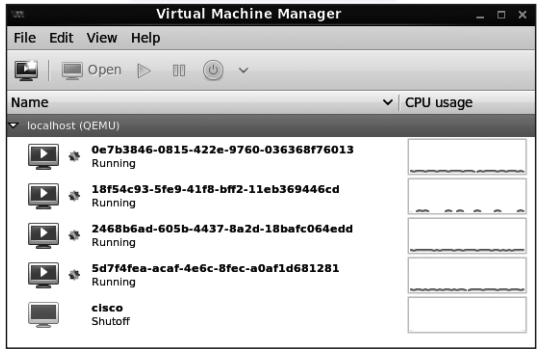


图 2-7　Virtual Machine Manager 界面

2. virt-install 命令使用介绍

virt-install 是一个在命令行创建 KVM 虚拟机的工具，使用 virt-install 配合一些配置参数，最终可以生成一个完整的 .xml 虚拟机配置文件。

```
#virt-install --name=testvm --ram=2048 --vCPUs=4 --os-type=Windows  --hvm
--cdrom=/root/W2003cnent.iso --file=/root/SDG100.img --file-size=10
```

```
--bridge=br0  --vnc --vncport=5920
```

参数说明如下。

- --name：设置虚拟机名称。
- --ram：配置虚拟机内存，单位是 MB。
- --vCPUs：配置 CPU 个数。
- --hvm：配置使用全虚拟化。
- --os-type：指定操作系统类型，如 Linux、Windows。
- --cdrom：使用 cdrom 安装系统，指定 ISO 位置。
- --file：设置虚拟机硬盘文件路径。
- --file-size：配置虚拟机硬盘文件大小，单位是 GB。
- --bridge：配置桥接的网卡。
- --vnc：打开 VNC 支持。
- --vncport：配置 VNC 端口。

执行上述命令之后，virt-install 会创建一台名为 testvm 的虚拟机，并使用 W2003cnent.iso 镜像文件安装系统。此时使用 VNC Viewer，在 VNC Server 中输入宿主机 ip:vncport，便可登录虚拟机的控制台，此时虚拟机开始从 ISO 引导，安装虚拟机系统的步骤和安装普通服务器系统是一样的，这里就不详述了。

3. Windows 虚拟机安装注意事项

第一次安装 Windows 虚拟机的时候，经常会碰到以下几个问题。

（1）qcow2 格式的磁盘如何操作

Virt-Manager 默认创建的磁盘格式是 RAW 格式，如果需要使用 qcow2 格式的磁盘，必须用 qemu-img create 手工先创建一个 qcow2 格式的磁盘镜像。

```
qemu-img create Windows-test.qcow2 -f qcow2 50G
```

然后如图 2-8 所示，在 Virt-Manager 上指定 qcow2 格式。

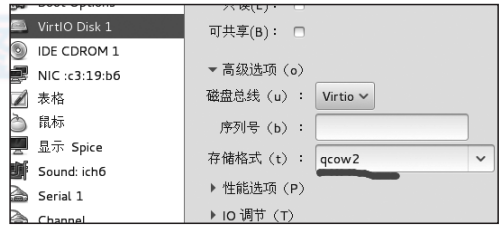


图 2-8 Virt-Manager 需要手工选择 qcow2 方式

提示 在使用 virt-install 命令，磁盘镜像格式为 qcow2 时，在 virt-install 命令中要特别指明磁盘格式，否则会出现镜像复制之后虚拟机系统不能启动的现象，这往往是初学者容易忽视的地方。

（2）光驱自动消失问题

Windows 系统安装的时候，重启后找不到光盘。

KVM 新创建虚拟机，第一次挂载的光驱，重启后自动消失，这是一个功能，专门针对 Linux 系统，但是 Windows 系统安装的时候需要多次重启，所以安装时第一次重启后，会出现如图 2-9 所示的界面，需要再手工挂载一下 Windows 系统 ISO 镜像。

图 2-9　Virt-Manager 需要手工选择再加载光驱一次

也可以修改虚拟机的 xml 配置文件，光驱配置的 xml 文件如下：

```
 <disk type='file' device='cdrom'>
 <driver name='qemu' type='raw' cache='none'/>
 <source file='/home/CentOS-7.0-1406-x86_64-DVD.iso'/>
 <target dev='hdb' bus='ide'/>
 <readonly/>
</disk>
```

（3）鼠标不同步问题

Windows 在 KVM 上会出现鼠标不同步问题，如图 2-10 所示，再添加一个 USB 鼠标设备就可以解决。

图 2-10　Virt-Manager 需要手工添加 USB 指针设备

添加一个 USB 鼠标设备的 xml 文件如下：

```
<input type='tablet' bus='usb'/>
```

如果添加两次 USB 设备，Windows 系统虚拟机系统启动会蓝屏，所以只能添加一次 USB 设备。

4. Linux 虚拟机安装注意事项

Linux 系统的安装除了会碰到 Windows 系统安装的 qcow2 磁盘格式问题、鼠标不同步的问题，Linux 虚拟机的安装还可以使用一种非常有意思的方式，就是可以直接指定内核文件

路径，然后直接加载，如图 2-11 所示。

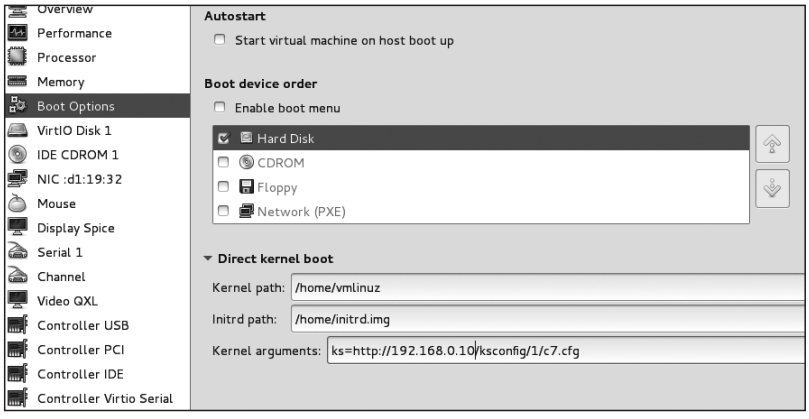


图 2-11 通过指定内核文件安装虚拟机

提示 内核文件需要使用安装光盘中的 images 下的 pxe 内核文件。

系统引导起来之后，会根据 kickstart 文件来下载系统，然后安装，如图 2-12 和图 2-13 所示。

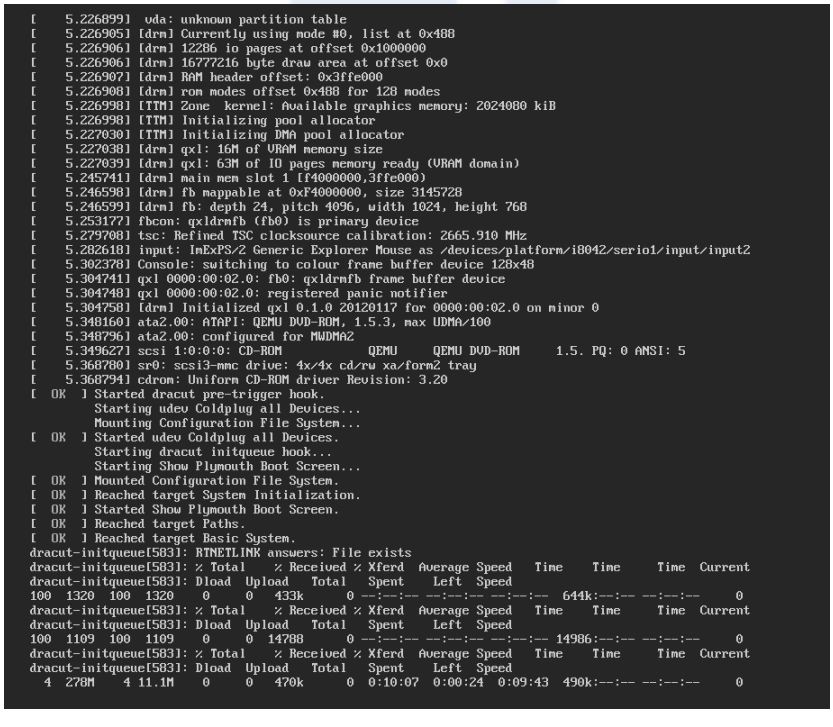
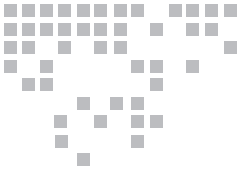


图 2-12 虚拟机启动后根据 kickstart 文件路径下载安装文件

图 2-13　随后根据 kickstart 文件安装系统

2.4　本章小结

本章介绍了如何安装宿主机、如何安装虚拟机及虚拟机安装时的一些注意事项。读者可以正式开始自己的虚拟化环境搭建了，在搭建的时候，还有许多技术点需要了解，下一章将为读者介绍 KVM 虚拟机 CPU、内存方面的技术及应用场景。



Chapter 3 第 3 章

CPU、内存虚拟化技术与应用场景

本章将为读者详细介绍 KVM 虚拟化中的 CPU 和内存技术，以及每种技术的应用场景。

KVM 虚拟机 CPU 的软件调优首先需要对 NUMA 技术有一定的了解，调优的主要手段就是虚拟机对物理机 CPU 逻辑核的手工绑定。

CPU 的 Nested 特性使用也是非常有意思的一个特性，KVM 虚拟机的嵌套在理论上可以无限层地嵌套下去，只要物理机的性能足够。

内存方面的调优手段主要是 KSM，即相同内存页合并、内存气球技术及大页内存的使用。

3.1 NUMA 技术与应用

NUMA 是一种解决多 CPU 共同工作的技术方案，首先回顾一下多 CPU 共同工作技术的架构历史。多 CPU 共同工作主要有 3 种架构，分别是 SMP、MPP、NUMA 架构。SMP、MPP、NUMA 都是为了解决多 CPU 共同工作的问题。

1. SMP 技术

早期的时候，每台服务器都是单 CPU，随着技术的发展，出现了多 CPU 共同工作的需求，最早的多 CPU 技术是 SMP。

如图 3-1 所示，SMP 即多个 CPU 通过一个总线访问存储器，因此 SMP 系统有时也被称为一致内存访问（UMA）结构体系。一致性意指无论在什么时候，处理器只能为内存的每个

数据保持或共享唯一一个数值。

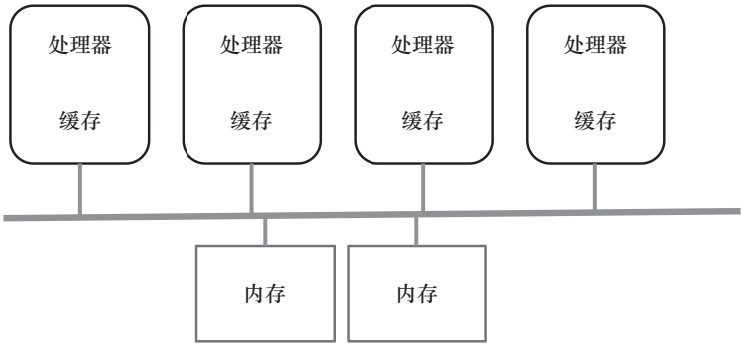


图 3-1　多 CPU 的 SMP 架构

SMP 的缺点是扩展性有限，因为在存储器接口达到饱和的时候，增加处理器并不能获得更高的性能，因此 SMP 方式支持的 CPU 个数有限。

2. MPP 模式

MPP 模式则是一种分布式存储器模式，能够将更多的处理器纳入一个系统的存储器。一个分布式存储器模式具有多个节点，每个节点都有自己的存储器，可以配置为 SMP 模式，也可以配置为非 SMP 模式。单个的节点相互连接起来就形成了一个总系统。MPP 可以近似理解成一个 SMP 的横向扩展集群。MPP 一般要依靠软件实现。

3. NUMA 技术

如图 3-2 所示，NUMA 模式则是每个处理器有自己的存储器，每个处理器也可以访问别的处理器的存储器。

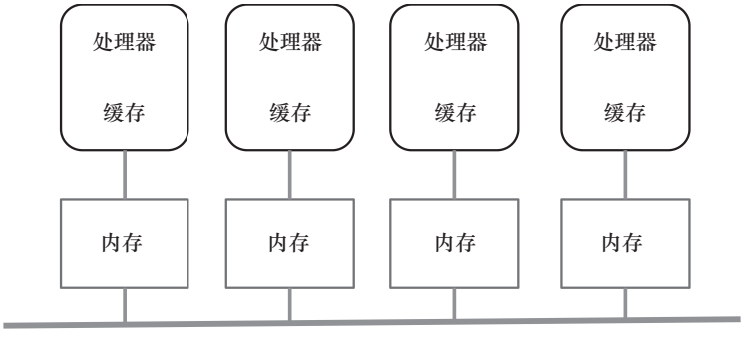


图 3-2　多 CPU 的 NUMA 架构

图 3-3 所示是多核 CPU 的 NUMA 架构。

NUMA-Q 是 IBM 最早将 NUMA 技术应用到 i386 上的商业方案，可以支持更多的 X86 CPU 一起工作。

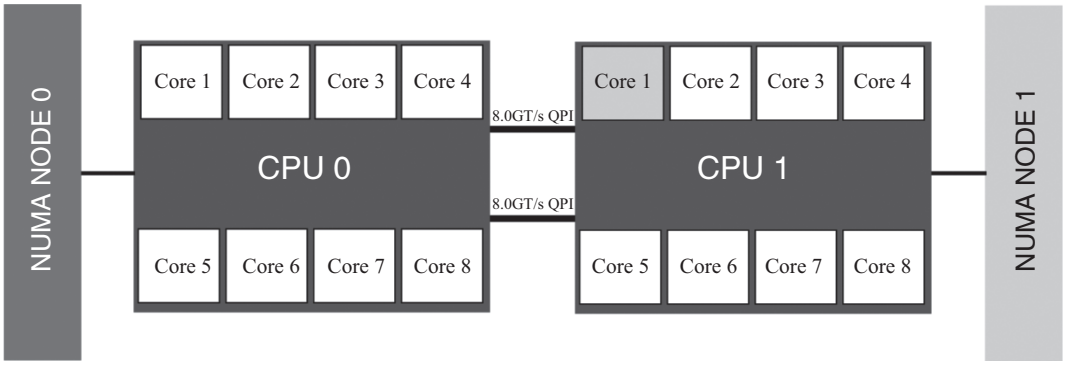


图 3-3 多核 NUMA CPU 架构（来自于 Intel 网站资料）

3.1.1 KVM 虚拟机 NUMA 调优

1. 宿主机的 NUMA 信息查看与配置

因为 NUMA 架构每个处理器都可以访问自己和别的处理器的存储器，访问自己的存储器要比访问别的存储器快很多，速度相差 10 ~ 100 倍，所以 NUMA 调优的目标就是让处理器尽量访问自己的存储器，以提高处理速度。

如图 3-4 所示，通过 numactl --hardware 命令可以看到当前 CPU 硬件的情况，CPU 有两颗，每颗有 8 个核，每个核有 64GB 内存可以使用。

```
[root@localhost ~]# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23
node 0 size: 65501 MB
node 0 free: 689 MB
node 1 cpus: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
node 1 size: 65535 MB
node 1 free: 4829 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

图 3-4 CPU NUMA 情况

使用 numastat 命令可以查看每个节点的内存统计。

```
                  node0           node1
numa_hit      4911255660       490070817 # 使用本节点内存次数
numa_miss       20552767       946830474 # 计划使用本节点内存而被调度到其他节点次数
numa_foreign   946830474        20552767 # 计划使用其他节点内存而使用本地内存次数
interleave_hit    14625           14612 # 交叉分配使用的内存中使用本节点的内存次数
local_node     911244851       490037204 # 在本节点运行的程序使用本节点内存次数
other_node      20563576       946864087 # 在其他节点运行的程序使用本节点内存次数
```

numastat 命令使用 -c 参数可以查看相关进程的 NUMA 内存使用情况。

```
numastat -c qemu-kvm
Per-node process memory usage (in MBs)
PID              Node 0 Node 1 Total
---------------  ------ ------ -----
3135 (qemu-kvm)    2317   1443  3760
3187 (qemu-kvm)    2508   2454  4962
3245 (qemu-kvm)    4424   2287  6711
20830 (qemu-kvm)    942   1124  2066
31717 (qemu-kvm)    852   1114  1967
---------------  ------ ------ -----
Total             11044   8422 19466
```

Linux 系统默认是自动 NUMA 平衡策略。如果要关闭 Linux 系统的自动平衡，可以使用如下命令：

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

如果要开启自动 NUMA 平衡策略，可以使用如下命令：

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

2. 虚拟机 NUMA 信息查看与配置

使用 virsh numatune 命令可以查看或者修改虚拟机的 NUMA 配置。

```
virsh # numatune 4
numa_mode      : strict
numa_nodeset   : 0-1
```

NUMA 工作方式可以是 strict 指定 CPU，或者 auto 使用系统的 numad 服务。

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

可以使用 numatune 命令配置虚拟机的 NUMA。

```
virsh numatune rhel7 --nodeset '0, 2-3'
```

vpcu 的设置如下：

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

<vcpu> 和 <numatune> 需要保持一致，<numatune> 配置的是物理 CPU，<vcpu> 配置的是 CPU 的核，包括超线程产生的核。<numatune> 使用 static 模式，<nodeset> 也必须是。

可以设置一个虚拟机给 32 个虚拟 CPU，但是一开始只能使用 8 个，然后根据系统压力，

热添加 CPU 给虚拟机。

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

也可以给每个虚拟机 CPU 指定具体的物理机 CPU pinning 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>
```

也可以使用 emulatorpin 的方式。emulatorpin 标签可以指定一个特定的物理 CPU 范围，比如一个物理 CPU 内部所有的核，使虚拟机使用的 CPU 和存储器都在一个物理机 CPU 内部，xml 文件如下：

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

可以在线调整，命令方式如下：

```
virsh emulatorpin CentOS7 1-3
```

1 ~ 3 的核都在一个物理 CPU 内部。也可以设置虚拟机对 NUMA 资源的使用。

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

标签项意义如下。

- cell：numa 的 cell 或者 numa 节点。
- cpus：一个物理 CPU 可以使用的 CPU 范围。
- memory：可以使用的内存大小，单位为 KB。

3. 虚拟机 NUMA 和内存 KSM

KSM 技术可以合并相同的内存页，即使是不同的 NUMA 节点，如果需要关闭跨 NUMA 节点的内存合并，设置 /sys/kernel/mm/ksm/merge_across_nodes 参数为 0。或者可以关闭特定虚拟机的 KSM 内存合并，在虚拟机的 xml 配置文件中添加以下内容就可以：

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

3.1.2　CPU 绑定操作方法

CPU 绑定是一项非常神奇的技术，最神奇的地方就是可以在线配置，并且立即生效，可以解决生产环境 CPU 利用率严重不平均的问题。

1. CPU 信息查看

使用 virsh vcpuinfo 命令查看虚拟机 VCPU 和物理 CPU 的对应关系。

```
virsh  vcpuinfo 21
VCPU: 0
CPU: 25
State: running
CPU time: 10393.0s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy

VCPU: 1
CPU: 8
State: running
CPU time: 7221.2s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy

VCPU: 2
CPU: 26
State: running
CPU time: 7321.4s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy

VCPU: 3
CPU: 8
State: running
CPU time: 6808.9s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy
```

可以看到 VCPU 0 被调度到物理机 CPU 25 上，目前是使用状态，使用时间是 10 393.0s。最后一行是 CPU 亲和性的信息：

```
CPU Affinity: --------yyyyyyyy--------yyyyyyyy
```

yyyyyyy 表示可以使用的物理 CPU 内部的逻辑核，可以看到这台虚拟机允许在 8 ~ 15、24 ~ 31 这些物理 CPU 之间调度。为什么不能使用 0 ~ 7、16 ~ 23 这些 CPU 呢？是因为系统的自动 NUMA 平衡服务在发生作用，一个虚拟机默认只能使用同一颗物理 CPU 内部的逻辑核。

使用 emulatorpin 命令可以查看虚拟机可以使用哪些物理逻辑 CPU。

```
virsh # emulatorpin 21
emulator: CPU Affinity
----------------------------------
       *: 0-31
```

可以看到 0 ~ 31 都可以使用，意味着可以强制将 VCPU 调度到任何物理 CPU 核上。

2. 在线绑定虚拟机的 CPU

（1）让虚拟机只能在部分物理 CPU 之间调度

可以强制让虚拟机只能在部分物理 CPU 之间调度。比如使用下面的命令，使编号为 21 的虚拟机 CPU 在 26 ~ 31 这些物理 CPU 之间调度：

```
virsh  emulatorpin 21 26-31 --live
```

通过以下命令查看绑定是否生效：

```
virsh  emulatorpin 21
emulator: CPU Affinity
----------------------------------
       *: 26-31
```

我们可以通过 vcpuinfo 命令和虚拟机的 xml 配置文件验证。

1）查看虚拟机的 VCPU 调度信息。

```
virsh  vcpuinfo 21
VCPU: 0
CPU: 28
State: running
CPU time: 10510.5s
CPU Affinity: ----------------------------yyyyyy

VCPU: 1
CPU: 28
State: running
CPU time: 7289.7s
CPU Affinity: ----------------------------yyyyyy

VCPU: 2
CPU: 27
State: running
CPU time: 7391.3s
CPU Affinity: ----------------------------yyyyyy

VCPU: 3
CPU: 27
State: running
CPU time: 6886.6s
CPU Affinity: ----------------------------yyyyyy
```

2）通过虚拟机的 xml 配置文件查看 VCPU 调度信息。

```
virsh # dumpxml 21
<domain type='kvm' id='21'>
  <name>cacti-230</name>
  <uuid>23a6455c-5cd1-20cd-ecfe-2ba89be72c41</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
```

```
  <vcpu placement='static'>4</vcpu>
  <cputune>
    <emulatorpin cpuset='26-31'/>
  </cputune>
...
```

（2）强制 VCPU 和物理机 CPU 一对一绑定

我们也可以强制 VCPU 和物理机 CPU 一对一地绑定。比如强制 VCPU 0 和物理机 CPU 28 绑定，强制 VCPU 1 和物理机 CPU 29 绑定，强制 VCPU 2 和物理机 CPU 30 绑定，强制 VCPU 3 和物理机 CPU 31 绑定，命令如下：

```
virsh  vcpupin 21 0 28
virsh  vcpupin 21 1 29
virsh  vcpupin 21 2 30
virsh  vcpupin 21 3 31
```

查看 xml 文件，验证 <cputune> 块中的 vcpupin 配置信息。

```
virsh # dumpxml 21
<domain type='kvm' id='21'>
  <name>cacti-230</name>
  <uuid>23a6455c-5cd1-20cd-ecfe-2ba89be72c41</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>4</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='28'/>
    <vcpupin vcpu='1' cpuset='29'/>
    <vcpupin vcpu='2' cpuset='30'/>
    <vcpupin vcpu='3' cpuset='31'/>
    <emulatorpin cpuset='26-31'/>
  </cputune>
...
```

验证一下虚拟机重启后，配置是否继续生效。将虚拟机关机再开机，检查一下配置是否一直有效，可以看到配置一直生效。

```
virsh  vcpuinfo 22
VCPU: 0
CPU: 28
State: running
CPU time: 1.8s
CPU Affinity: ----------------------------y---
VCPU: 1
CPU: 29
State: running
CPU time: 0.0s
CPU Affinity: ----------------------------y--
VCPU: 2
CPU: 30
```

```
State: running
CPU time: 0.0s
CPU Affinity: ------------------------------y-
VCPU: 3
CPU: 31
State: running
CPU time: 0.0s
CPU Affinity: -------------------------------y
```

3. CPU 绑定技术的原理

CPU 绑定实际上是 Libvirt 通过 CGroup 来实现的，通过 CGroup 直接去绑定 KVM 虚拟机进程也可以。通过 CGroup 不仅可以做 CPU 绑定，还可以限制虚拟机磁盘、网络的资源控制，本书第 6 章将详细介绍 CGroup 在 KVM 虚拟化中的使用。

4. CPU 绑定技术的应用场景

在 CPU 压力比较大的时候使用，就要考虑使用 CPU 绑定技术。根据笔者的经验，不同的业务场景，使用 CPU 绑定效果不一样，有的业务性能有 20% 以上的提升。

笔者还碰到过一个案例，在一台 32 核的宿主机上，有一台虚拟机压力很大，宿主机的前 3 个 CPU 利用率达到了 90% 以上，中间几个 CPU 利用率为 60% ~ 80%，后面第 27 ~ 31 的 6 个 CPU 利用率只有 40%，当时业务已经出现性能问题，通过在线使用 CPU 绑定技术，将压力大的虚拟机绑定到了后面几个 CPU 上，解决了问题。

所以 CPU 绑定技术适用于以下场景：

- ❑ 系统的 CPU 压力比较大。
- ❑ 多核 CPU 压力不平衡，可以通过 cpu pinning 技术人工进行调配。

NUMA 在虚拟化应用之外的提示：

某些应用，比如数据库，为了保证性能，需要尽量使用更多的内存，如果使用系统默认的 NUMA 自动平衡策略，有可能一个 CPU 的内存消耗光，另外一个 CPU 还有大量的内存可以使用，但是系统却去调度 swap 来使用，造成性能严重降低。使用这些内存消耗型的应用时，可以考虑直接将系统的 NUMA 自动平衡策略关闭。

5. CPU 绑定对性能提升的测试案例

CPU 绑定到底对性能影响有多大，笔者进行了一个简单的测试。

（1）测试环境

硬件：Intel(R) Xeon(R) CPU X5650 @ 2.67GHz 2 颗。

软件：CentOS 7 更新到内核 3.10.0-123.8.1.el7.x86_64。

虚拟机：CentOS 7 更新到内核 3.10.0-123.8.1.el7.x86_64。

虚拟机：CPU 一颗。

测试工具：unixbench 5.1.2（使用方法及分项测试意义在第 15 章有介绍）。

（2）测试结果

1）不做 CPU 绑定 unixbench 的得分：

```
1 CPU in system; running 1 parallel copy of tests
Dhrystone 2 using register variables       28890881.0 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     3880.4 MWIPS (9.0 s, 7 samples)
...
                                                                 ========
System Benchmarks Index Score                                    1444.7
```

2）做了 CPU 绑定 unixbench 的得分：

```
1 CPU in system; running 1 parallel copy of tests
Dhrystone 2 using register variables       29812559.6 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     3928.7 MWIPS (8.9 s, 7 samples)
...
                                                                 ========
System Benchmarks Index Score                                    1464.1
```

（3）简单比较

- unixbench 综合得分（性能提升 1.34%）：绑定——1464.1；不绑定——1444.7。
- CPU 浮点运算（性能提升 1.24%）：绑定——3928.7；不绑定——3880.4。

6. CPU 绑定的一个生产环境使用案例

图 3-5 和图 3-6 所示是一台 32 核的宿主机没有做 CPU 绑定之前的利用率，从第一个核和最后一个核 CPU 利用率的比较中可以看出差距较大。

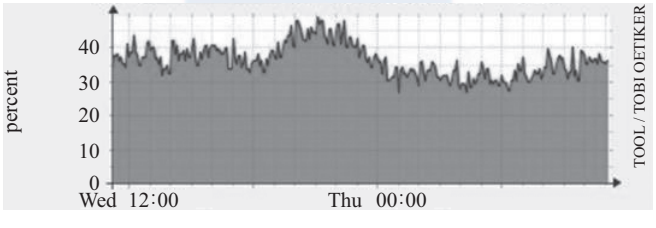


图 3-5　未做 CPU 绑定之前的 CPU 1 利用率

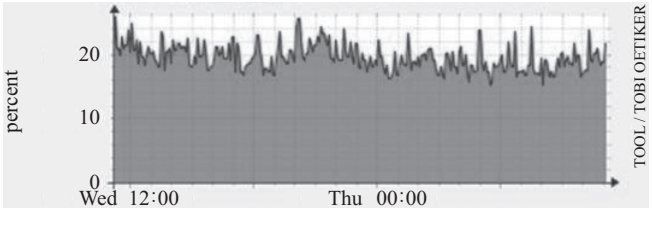


图 3-6　未做 CPU 绑定之前的 CPU 31 利用率

图 3-7 和图 3-8 所示是做了 CPU 绑定之后，第一个核和最后一个核 CPU 利用率差距已

经减少，拉平了整体的 CPU 利用率。

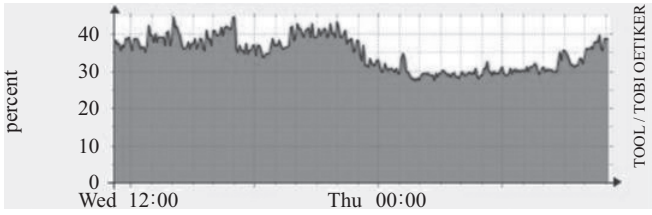


图 3-7 做 CPU 绑定之后 CPU 1 利用率

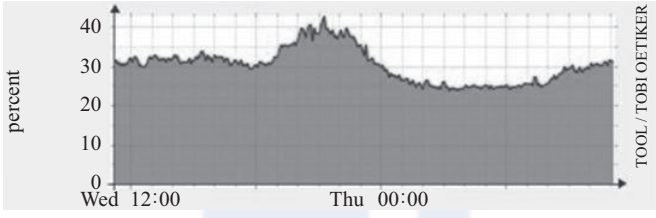


图 3-8 做 CPU 绑定之后 CPU 31 利用率

3.2 CPU 热添加与应用

CPU 热添加最早的时候是大型机上的硬件技术，随后一些高端的 X86 服务器也支持硬件的 CPU 热添加。CPU 热添加也需要操作系统的支持，在虚拟化上 CPU 热添加反而变得容易，因为虚拟化只需要修改软件配置就可以，不需要做硬件配置变更。

CPU 热添加是 CentOS 7 的一个新特性，Linux 系统要求宿主机和虚拟机都是 CentOS 7，Windows 虚拟机系统要求是 Windows Server 2008 数据中心版或者 Windows Server 2012 标准版和数据中心版。

1. Linux 系统的 CPU 热添加

如图 3-9 所示，在给虚拟机分配 CPU 的时候，就要预留足够的 CPU。

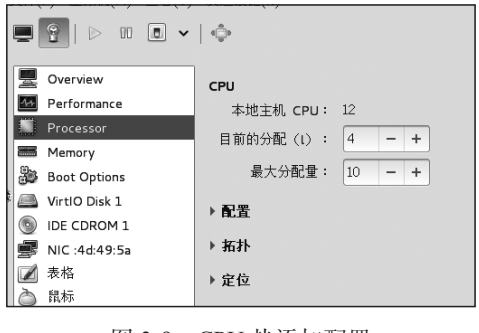


图 3-9 CPU 热添加配置

如图 3-10 所示，在虚拟机中可以看到 4 个 CPU。

```
[root@localhost ~]# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:        123          0          0          0   IO-APIC-edge      timer
  1:         10          0          0          0   IO-APIC-edge      i8042
  6:          3          0          0          0   IO-APIC-edge      floppy
  8:          0          0          0          0   IO-APIC-edge      rtc0
  9:          0          0          0          0   IO-APIC-fasteoi   acpi
 10:       4007          0          0          0   IO-APIC-fasteoi   virtio3
 11:         62         10          0          0   IO-APIC-fasteoi   uhci_hcd:usb1, qxl
 12:        144          0          0          0   IO-APIC-edge      i8042
 14:          0          0          0          0   IO-APIC-edge      ata_piix
 15:        144          0          0         67   IO-APIC-edge      ata_piix
 40:          0          0          0          0   PCI-MSI-edge      virtio2-config
 41:       7342          0        140          0   PCI-MSI-edge      virtio2-requests
 42:          0          0          0          0   PCI-MSI-edge      virtio4-config
 43:        205          0          0          0   PCI-MSI-edge      virtio4-requests
 44:          0          0          0          0   PCI-MSI-edge      virtio0-config
 45:        294          0          0          0   PCI-MSI-edge      virtio0-input.0
 46:          1          0          0          0   PCI-MSI-edge      virtio0-output.0
 47:          0          0          0          0   PCI-MSI-edge      virtio5-config
 48:        205          0          0          0   PCI-MSI-edge      virtio5-requests
 49:          0          0          0          0   PCI-MSI-edge      virtio6-config
 50:        205          0          0          0   PCI-MSI-edge      virtio6-requests
 51:          0          0          0          0   PCI-MSI-edge      virtio7-config
```

图 3-10　CPU 热添加前进行虚拟机 CPU 查看

把 CPU 在线修改成 5 个：

```
virsh setvcpus CentOS7 5 --live
```

在虚拟机里面将第 5 个 CPU 激活：

```
echo 1 >/sys/devices/system/cpu/cpu4/online
```

如图 3-11 所示，可以看到虚拟机的 CPU 已经变成了 5 个：

```
[root@localhost ~]# cat /proc/interrupts |less
           CPU0       CPU1       CPU2       CPU3       CPU4
  0:        123          0          0          0          0   IO-APIC-edge      timer
  1:         10          0          0          0          0   IO-APIC-edge      i8042
  6:          3          0          0          0          0   IO-APIC-edge      floppy
  8:          0          0          0          0          0   IO-APIC-edge      rtc0
  9:          1          0          0          0          0   IO-APIC-fasteoi   acpi
 10:       4007          0          0          0          0   IO-APIC-fasteoi   virtio3
 11:         64         35          0          0          0   IO-APIC-fasteoi   uhci_hcd:usb1, qxl
 12:        144          0          0          0          0   IO-APIC-edge      i8042
 14:          0          0          0          0          0   IO-APIC-edge      ata_piix
 15:        144          0          0        233          0   IO-APIC-edge      ata_piix
 40:          0          0          0          0          0   PCI-MSI-edge      virtio2-config
 41:       7342          0        172          0          6   PCI-MSI-edge      virtio2-requests
 42:          0          0          0          0          0   PCI-MSI-edge      virtio4-config
 43:        205          0          0          0          0   PCI-MSI-edge      virtio4-requests
 44:          0          0          0          0          0   PCI-MSI-edge      virtio0-config
 45:        866          0          0         76          0   PCI-MSI-edge      virtio0-input.0
 46:          1          0          0          0          0   PCI-MSI-edge      virtio0-output.0
 47:          0          0          0          0          0   PCI-MSI-edge      virtio5-config
 48:        205          0          0          0          0   PCI-MSI-edge      virtio5-requests
 49:          0          0          0          0          0   PCI-MSI-edge      virtio6-config
 50:        205          0          0          0          0   PCI-MSI-edge      virtio6-requests
 51:          0          0          0          0          0   PCI-MSI-edge      virtio7-config
 52:        205          0          0          0          0   PCI-MSI-edge      virtio7-requests
 53:          0          0          0          0          0   PCI-MSI-edge      virtio8-config
 54:        205          0          0          0          0   PCI-MSI-edge      virtio8-requests
 55:          0          0          0          0          0   PCI-MSI-edge      virtio1-config
```

图 3-11　热添加 CPU 到 5 个

我们也可以查看 cpuinfo 的信息来确认 CPU 增加了。

```
cat /proc/cpuinfo
...
processor       : 4
vendor_id       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Intel(R) Xeon(R) CPU           X5650  @ 2.67GHz
stepping        : 2
microcode       : 0x1
cpu MHz         : 2665.908
cache size      : 4096 KB
physical id     : 4
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 4
initial apicid  : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                  pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb
                  rdtscp lm constant_tsc arch_perfmon rep_good nopl pni
                  pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic popcnt aes
                  hypervisor lahf_lm tsc_adjust
bogomips        : 5331.81
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

因为一开始预留的是 10 个，所以最多的时候，可以热添加 CPU 到 10 个。

```
virsh # setvcpus ct7 10 --live
```

在虚拟机上执行如下命令：

```
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu4/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu5/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu6/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu7/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu8/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu9/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu9/online
```

查看 cpuinfo，已经是 10 个 CPU 了。

```
cat /proc/cpuinfo  |grep processor |wc -l
10
```

目前还不支持 CPU 热去除，可以在虚拟机里面关闭 CPU，使用如下命令：

```
echo 0 >/sys/devices/system/cpu/cpu4/online
```

但是在宿主机上如果要把 CPU 个数改小，是不支持的。

```
virsh # setvcpus ct7 4 --live
错误：内部错误：无法更改这个域的 vcpu 计数
```

2. Windows 系统的 CPU 热添加

Windows 系统也一样，开始的时候就要预留 CPU，需要的时候直接添加就可以了。如图 3-12 所示，未做 CPU 添加之前，有 4 个 CPU。

图 3-12　Windows 系统热添加 CPU 之前

将 CPU 在线添加到 6 个。

```
virsh # setvcpus window2012 6 --live
```

如图 3-13 所示，添加之后，虚拟机 CPU 已经变成 6 个。Windows 系统 CPU 添加后直接就可以自动刷新出来，不需要做特殊配置。同样 Windows 系统 CPU 也不能在线减少。

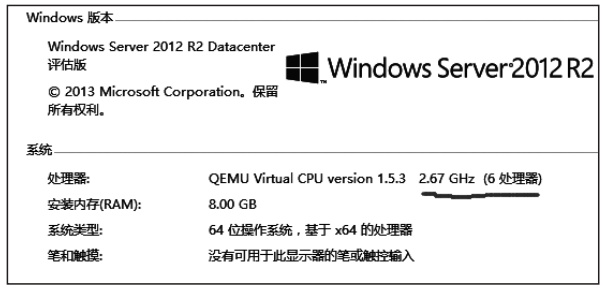


图 3-13　Windows 系统热添加 CPU 之后

3. CPU 热添加技术的应用场景

如果在虚拟机运行的应用非常重要，不能停机而性能严重不足的场景，CPU 热添加的技术是一个很好的解决方案。CPU 热添加的技术是比较新的技术，要求的宿主机、虚拟机操作

系统比较新，还要待在生产环境中逐步实践。

3.3 CPU host-passthrough 技术与应用

1. 关于 KVM 虚拟机的 CPU 型号的定义

为了保证虚拟机在不同宿主机之间迁移时候的兼容性，Libvirt 对 CPU 提炼出标准的几种类型，在 /usr/share/libvirt/cpu_map.xml 中可以查到。cpu_map.xml 不仅是 CPU 型号，还有生产商信息、每种型号的 CPU 特性定义等信息。

```
<cpus>
  <arch name='x86'>
    <!-- vendor definitions -->
    <vendor name='Intel' string='GenuineIntel'/>
    <vendor name='AMD' string='AuthenticAMD'/>

    <!-- standard features, EDX -->
    <feature name='fpu'> <!-- CPUID_FP87 -->
      <cpuid function='0x00000001' edx='0x00000001'/>
    </feature>
    <feature name='vme'> <!-- CPUID_VME -->
      <cpuid function='0x00000001' edx='0x00000002'/>
    </feature>
...
<!-- models -->
    <model name='486'>
      <feature name='fpu'/>
      <feature name='vme'/>
      <feature name='pse'/>
    </model>
...
 <model name='Haswell'>
      <model name='SandyBridge'/>
      <feature name='fma'/>
      <feature name='pcid'/>
      <feature name='movbe'/>
      <feature name='fsgsbase'/>
      <feature name='bmi1'/>
      <feature name='hle'/>
      <feature name='avx2'/>
      <feature name='smep'/>
      <feature name='bmi2'/>
      <feature name='erms'/>
      <feature name='invpcid'/>
      <feature name='rtm'/>
    </model>
...
```

CentOS 6.6 所带的 Libvirt 主要规定了以下几种 CPU 型号：'486'、'pentium'、'pentium2'、'pe

ntium3'、'pentiumpro'、'coreduo'、'pentiumpro'、'n270'、'coreduo'、'core2duo'、'qemu32'、'kvm32'、'cpu64-rhel5'、'cpu64-rhel6'、'kvm64'、'qemu64'、'Conroe'、'Penryn'、'Nehalem''Westmere'、'SandyBridge'、'Haswell'、'athlon'、'phenom'、'Opteron_G1'、'Opteron_G2'、'Opteron_G3'、'Opteron_G4'、'Opteron_G5'、'POWER7'、'POWER7_v2.1'、'POWER7_v2.3'。

2. CPU 模式配置

CPU 配置模式可以有以下几种。

（1）custom 模式

xml 配置文件如下：

```
<cpu mode='custom' match='exact'>
    <model fallback='allow'>kvm64</model>
 ...
    <feature policy='require' name='monitor'/>
  </cpu>
```

（2）host-model 模式

根据物理 CPU 的特性，选择一个最靠近的标准 CPU 型号。如果没有指定 CPU 模式，默认也是使用这种模式，xml 配置文件如下：

```
<cpu mode='host-model' />
```

（3）host-passthrough 模式

直接将物理 CPU 暴露给虚拟机使用，在虚拟机上完全可以看到的就是物理 CPU 的型号，xml 配置文件如下：

```
<cpu mode='host-passthrough'/>
```

使用 host-model 看到的 VCPU 如下：

```
processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Westmere E56xx/L56xx/X56xx (Nehalem-C)
...
```

使用 host-passthrough 看到的 VCPU 如下：

```
processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Intel(R) Xeon(R) CPU           X5650  @ 2.67GHz
```

可以看到，使用 host-model 模式，Libvir 会根据物理 CPU 的型号，从规定的 CPU 中选择一种最接近的 CPU 型号，而使用 host-passthrough 模式直接看到的就物理 CPU 的型号。

3. CPU host-passthrough 技术的应用场景

HOST 技术适用于以下场景：

- 需要将物理 CPU 的一些特性传给虚拟机使用，比如使用虚拟机嵌套的 nested 技术的时候。
- 需要在虚拟机里面看到和物理 CPU 一模一样的 CPU 品牌型号，这个在一些公有云很有意义，用户体验比较好。

使用 CPU host-passthrough 技术需要注意，不同型号 CPU 的宿主机之间虚拟机不能迁移。

3.4 CPU Nested 技术与配置方法

Nested 技术，简单地说，就是在虚拟机上运行虚拟机，即 KVM on KVM。KVM 虚拟机嵌套和 VMWare 原理不同，VMWare 第一层是用的硬件虚拟化技术，第二层就是完全软件模拟出来的，所以 VMWare 只能做两层嵌套。KVM 是将物理 CPU 的特性全部传给虚拟机，所以理论上可以嵌套 N 多层，但是事实上笔者测试的时候，跑了两层就已经非常慢了。

CentOS 7 官方宣称不正式支持 Nested 技术，所以测试的时候建议使用 Fedora 测试，笔者测试使用的是 Fedora 21。

Nested 配置方法如下。

第一步：打开 KVM 内核模块的 Nested 特性。

```
rmmode kvm-intel
modprobe kvm-intel nested=1
```

或者修改 modprobe.d，编辑 /etc/modprobe.d/kvm_mod.conf，添加以下内容：

```
options kvm-intel nested=y
```

然后使用 rmmod kvm-intel 命令删除 kvm-intel 模块，再通过 modprobe kvm-intel 命令加载 kvm-intel 模块，编辑 modprobe.d 配置文件，就不用在加载模块的时候带参数。

检查是否打开 Nested 功能，可以查看 /sys/module/kvm_intel/parameters/nested 的内容，为 Y 表示打开 Nested 特性。

```
cat /sys/module/kvm_intel/parameters/nested
Y
```

第二步：第一层的虚拟机配置文件，要将物理机 CPU 的特性全部传给虚拟机，使用 CPU HOST 技术。

```
<cpu mode='host-passthrough'/>
```

第三步：和宿主机一样，将第一层虚拟机按照宿主机配置，安装相应的组件，然后就可以安装第二层的虚拟机了。

图 3-14 所示是三层 Fedora 21 嵌套的效果（fedora21 on fedaro21 kvm on fedaro21 kvm）。

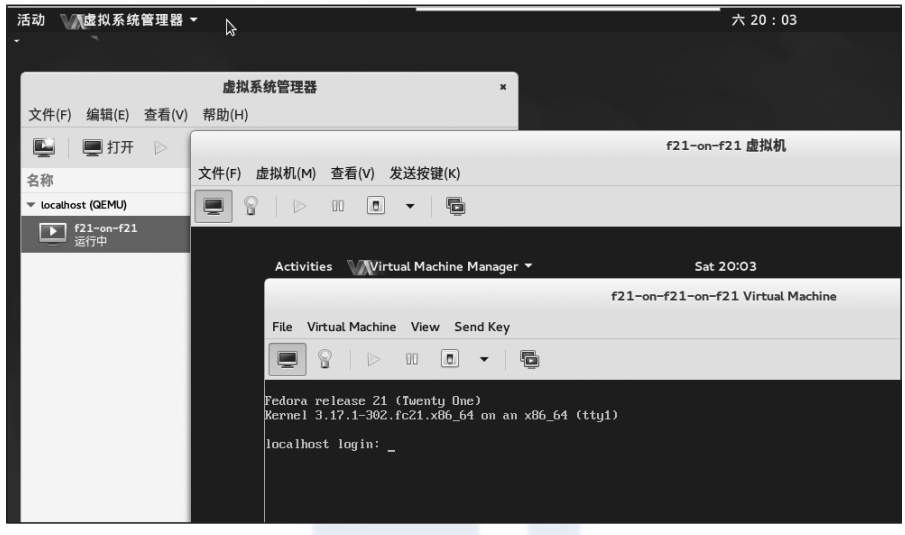


图 3-14　Fedora 21 中嵌套两层虚拟机的情况

3.5　KSM 技术与应用

1. 宿主机内存合并（压缩）

宿主机的内存压缩主要采用 KSM（Kernel SamePage Merging）技术，原理和软件压缩类似，就是将相同的内存分页进行合并。KSM 在 CentOS 6、CentOS 7 上默认是打开的，主要有两个服务：

- ❑ KSM 服务。
- ❑ ksmtuned 服务。

要关闭 KSM，关闭相关的两个服务就可以。

```
service ksm stop
service ksmtuned stop
chkconfig ksm off
chkconfig ksmtuned off
```

如果需要打开或者关闭 KSM，在生产环境中可以直接在线操作，系统会逐步合并或者释放内存页，对业务是没有影响的。所以当宿主机内存不足时，临时打开 KSM 也是一种应急方案。

2. 阻止个别虚拟机进行内存压缩的方法

使用 nosharepages 关键字阻止宿主机将特定的虚拟机内存页合并，xml 配置文件如下：

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

3. 查看 KSM 运行的情况

在 /sys/kernel/mm/ksm/ 中可以看到 KSM 运行的情况。

- pages_shared：有多少共享内存页正在被使用。
- pages_sharing：how 有多少节点被共享并且多少被保存。
- pages_unshared：内存被合并时有多少内存页独特但是反复被检查。
- pages_volatile：多少内存页改变太快被放置。
- full_scans：多少次可以合并区域被扫描。

4. KSM 技术的应用场景

在生产环境中一般要慎用 KSM 技术，因为打开 KSM 就是为了内存超用，这在业务压力比较大，虚拟机内存变化非常频繁的时候，将导致两个结果：

- 一是会消耗一定的计算机资源用于内存扫描，加重 CPU 的消耗。
- 二是因为是内存超用，当内存不够的时候，只能是频繁地使用 swap 交互，导致虚拟机性能严重下降。

但是在测试环境和桌面虚拟化这样的场景中就很适合使用 KSM 技术，因为测试环境一般压力都不太高，能容忍虚拟机性能突然下降，打开 KSM 可以节约大量的内存，多部署一些虚拟机出来。在桌面虚拟化中，往往虚拟机的操作系统都是一样的，打开 KSM，内存压缩的效果就非常明显，只要控制好内存超用的范围就好了。

KSM 应用场景总结如下：

- 生产环境慎用。
- 测试环境推荐使用。
- 桌面虚拟机推荐使用，但是要根据实际情况控制内存超用的比率。

3.6 内存气球技术详解与应用

1. 宿主机内存气球技术配置

KVM 的内存气球技术可以在虚拟机之间按照需要调节内存大小，提高内存的利用率。

使用的时候，虚拟机需要安装 virt balloon 的驱动，内核开启 CONFIG_VIRTIO_BALLOON。CentOS 7 默认已经开启，并且默认已经安装 virt balloon 驱动。如图 3-15 所示，在虚拟机中可以看到有一个名为 Virtio memory balloon 的 PCI 设备。

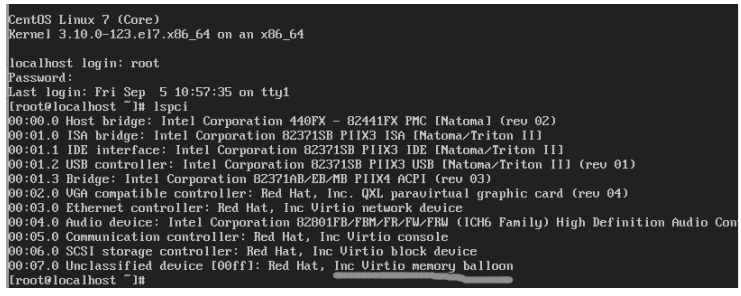


图 3-15　虚拟机的 virt balloon 设备

虚拟机 .xml 配置文件需要增加以下配置：

```
  <memballoon model='virtio'>
      <alias name='balloon0'/>
 </memballoon>
```

balloon 有两种操作。

- ❑ 膨胀：虚拟机的内存被拿掉给宿主机。
- ❑ 压缩：宿主机的内存还给虚拟机。

气球技术最大的优点是内存可以超用；缺点是有可能造成内存不够使用而影响性能。

2. 虚拟机内存气球配置

（1）Linux 系统配置

查看当前内存大小，使用 virsh qemu-monitor-command 命令。

```
virsh  qemu-monitor-command ct7 --hmp --cmd info balloon
balloon: actual=4096
```

限制内存大小为 2GB 并查看。

```
virsh  qemu-monitor-command ct7 --hmp --cmd balloon 2048
virsh  qemu-monitor-command ct7 --hmp --cmd info balloon
balloon: actual=2048
```

图 3-16 所示是在虚拟机中通过 free 命令查看内存限制效果。

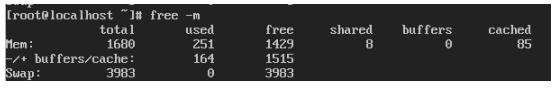


图 3-16　限制内存大小为 2GB 时虚拟机内存情况

限制内存大小为 4GB 并查看。

```
virsh # qemu-monitor-command ct7 --hmp --cmd balloon 4096
virsh # qemu-monitor-command ct7 --hmp --cmd info balloon
balloon: actual=4096
```

图 3-17 所示是在虚拟机中通过 free 命令查看内存限制效果。

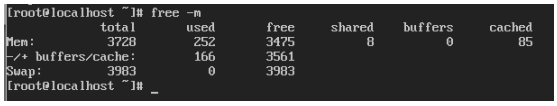


图 3-17 限制内存大小为 4GB 时虚拟机内存情况

限制内存大小为 8GB 并查看。

```
virsh # qemu-monitor-command ct7 --hmp --cmd balloon 8192
virsh # qemu-monitor-command ct7 --hmp --cmd info balloon
balloon: actual=8192
```

图 3-18 所示是在虚拟机中通过 free 命令查看内存限制效果。

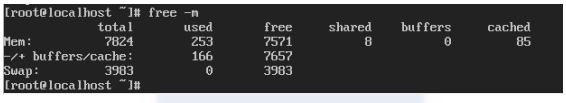


图 3-18 限制内存大小为 8GB 时虚拟机内存情况

要实现虚拟机内存气球的自动平衡，可以根据自己的业务情况，编写脚本来实现。

（2）Windows 系统配置

Windows 系统的使用方法与 Linux 系统类似，但是 Windows 系统能看到整体分配的内存，然后通过内存气球技术来限制系统可以分配的内存。具体步骤如下。

第 1 步：安装 virt balloon 设备驱动，安装完成后，如图 3-19 所示，就可以看到一个 virt balloon 的 PCI 设备。

图 3-19 Windows 系统的 balloon 设备

第 2 步：安装内存气球服务，如图 3-20 所示。

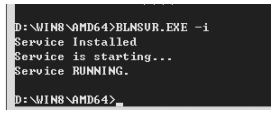


图 3-20 Windows 系统 balloon 服务安装

通过命令在线调整内存气球为 512MB、2GB、4GB、8GB。

```
virsh #  qemu-monitor-command 14 --hmp --cmd balloon 512
virsh #  qemu-monitor-command 14 --hmp --cmd balloon 2048
virsh #  qemu-monitor-command 14 --hmp --cmd balloon 4096
virsh #  qemu-monitor-command 14 --hmp --cmd balloon 8192
```

如图 3-21 所示，可以明显看到内存变化。

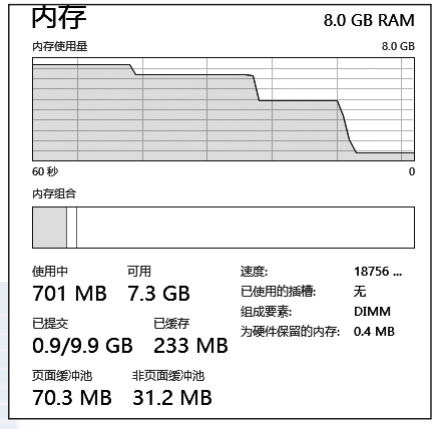


图 3-21　Windows 系统 balloon 内存变化

3. 内存气球技术应用场景

如果有两种或者几种不同业务的虚拟机在同一台宿主机上，可以考虑使用气球技术，在不同的时间段释放或者申请内存，这样可以提高内存利用率。往往要达到效果，需要编写一个脚本，通过时间或者一定的触发条件和虚拟机互动来调整内存。

3.7　内存限制技术与应用

1. 内存限制技术详解

可以将虚拟机的内存限定在一定的范围内，可以通过 virsh 命令行限制内存，也可以编辑 .xml 文件，命令行格式为：

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

可选的参数如下。

- hard_limit：虚拟机可以使用的最大内存，单位为 kibibytes（blocks of 1024 bytes）。
- soft_limit：竞争时的内存，单位为 kibibytes（blocks of 1024 bytes）。
- swap_hard_limit：最大内存加 swap，单位为 kibibytes（blocks of 1024 bytes）。
- min_guarantee：最低保证给虚拟机使用的内存，单位为 kibibytes（blocks of 1024 bytes）。

应用示例：

1）限制虚拟机 c7 最大使用 9GB 内存，写到配置文件中，下次重启虚拟机进程生效。

```
memtune c7 --hard-limit 9437184 --config
```

2）限制虚拟机 c7 竞争时为 7GB 内存。

```
memtune c7 --soft-limit 7340032 --config
```

3）限制虚拟机 c7 最大内加可以使用的宿主机 swap 不超过 10GB 内存。

```
memtune c7 --swap-hard-limit 10488320 --config
```

4）保证虚拟机 c7 最少可以使用 4GB 内存。

```
memtune c7 --min_guarantee 4194304 --config
```

注意，memtune 生效方式有 3 种。

- --config：写到配置文件中，下次重启虚拟机进程生效。
- --live：影响正在运行的虚拟机，虚拟机进程停止后，效果消失，这是默认的方式。
- --current ：影响停止和正在运行的虚拟机，如果虚拟机运行，虚拟机进程停止后，效果消失。

如果虚拟机未运行，写入到 .xml 配置文件中，对应的 .xml 文件如下：

```
<memory unit='KiB'>8388608</memory>
<currentMemory unit='KiB'>4194304</currentMemory>
<memtune>
    <hard_limit unit='KiB'>9437184</hard_limit>
    <soft_limit unit='KiB'>7340032</soft_limit>
    <min_guarantee unit='KiB'>4194304</min_guarantee>
    <swap_hard_limit unit='KiB'>10488320</swap_hard_limit>
</memtune>
```

还可以限制虚拟机对宿主机 swap 的使用，主要是通过 .xml 文件的 <memoryBacking> 标签来实现的。<memoryBacking> 标签包含一些元素影响宿主机对虚拟内存页的支持。

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

locked 阻止宿主机将 swap 内存（交换分页内存）分配给虚拟机。设置 locked 参数，必须在 <memtune> 块中设置 hard_limit。

2. 内存限制技术的应用场景

内存限制技术可以和内存气球技术结合，将内存气球技术限制在一定范围内，避免内存被气球无限压缩。

3.8 巨型页内存技术与应用

1. 巨型页与透明巨型页

X86 默认的内存页大小是 4KB，但是也可以使用 2MB 或者 1GB 的巨型页，系统的巨型页可以传输过虚拟机，KVM 虚拟机可以通过分配巨型页提高性能。在 CentOS 5 上面，需要手工配置巨型页。在 CentOS 6 上面启用一种叫作透明巨型页的技术，默认开启巨型页，并且可以自动调整。

使用巨型页可以提升内存的分配效率，提升系统性能。巨型页可以手工配置，也可以使用透明巨型页技术。

巨型页可以手工配置的坏处：必须手工配置，虚拟机的数量、可用的内存、虚拟机的启

动、关闭、迁移都需要重新配置，并且不能使用 swap。

使用透明巨页内存的好处：

- 可以使用 swap，内存页默认大小是 2MB，需要使用 swap 的时候，内存被分割为 4KB。
- 对用户透明，不需要用户做特殊配置。
- 不需要 root 权限。
- 不需要依赖某种库文件。

2. 透明巨型页内存配置

CentOS 6.x 默认启用透明巨型页内存。查看目前状态：

```
cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

修改配置：

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

参数说明如下：

- never：关闭，不使用透明内存。
- alway ：尽量使用透明内存，扫描内存，有 512 个 4KB 页面可以整合，就整合成一个 2MB 的页面。
- madvise：避免改变内存占用。

使用情况监控：

可以查看 /sys/kernel/mm/transparent_hugepage/khugepaged 下的信息。

- pages_to_scan（默认 4096=16MB）：一个扫描周期被扫描的内存页数。
- scan_sleep_millisecs（默认 10 000=10sec）：多长时间扫描一次。
- alloc_sleep_millisecs（默认 60 000=60sec）：多长时间整理一次碎片。

也可以查看 /proc/meminfo 信息。

```
grep Huge /proc/meminfo
AnonHugePages:    266240 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

使用注意要点：虚拟机和宿主机都需要启动 THP，这样可以获得以下好处。

- 虚拟机的 CPU 可以使用 2MB 的 TLB。
- 相同的算法。
- 相同的代码。

❑ 相同的内核镜像。

3. 巨型页手工配置

（1）虚拟机巨型页手工配置

有的场景需要手工配置虚拟机可以使用的巨型页数量。通过配置虚拟机 .xml 文件，可以指定虚拟机可以使用的巨型页数量。

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

（2）巨型页使用检查

要查看当前巨型页使用情况，使用如下命令：

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

查看当前的巨型页值，命令如下：

```
View the current huge pages value:
# cat /proc/meminfo | grep Huge
AnonHugePages:      2048 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

（3）修改宿主机巨型页数量

巨型页默认大小是 2MB，可以通过如下命令修改可以使用的巨型页数量：

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

或者使用 sysctl 命令，N 是要设置的巨型页数量：

```
sysctl vm.nr_hugepages=N
```

挂载巨型页，命令如下：

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

重启 libvirtd 服务和虚拟机：

```
systemctl start libvirtd
virsh start virtual_machine
```

（4）关闭巨型页

关闭巨型页，命令如下：

```
sysctl vm.nr_hugepages=0
umount hugetlbfs
```

4. 内存巨型页的应用场景

对于生产环境是 CentOS 6、CentOS 7 的宿主机，默认是打开透明巨型页并且是自动调整的，所以不需要我们做过多的设置。但是可以监控巨型页的使用分配，在内存遇到瓶颈的时候，方便查找原因。

3.9　本章小结

本章介绍了 KVM 虚拟化 CPU、内存技术，如果在生产环境中遇到 CPU 压力比较高的情况，CPU 的绑定是一个有效的技术解决手段。一般在生产环境中，内存限制技术建议慎用。目前服务器内存相对比较便宜，建议宿主机的内存尽量配置高一些。

下一章将为读者介绍 KVM 虚拟化网络技术及应用场景。