

本站文章大部分为作者原创，非商业用途转载无需作者授权，但务必在文章标题下面注明作者 刘世民（Sammy Liu）以及可点击的本博客地址<http://www.cnblogs.com/sammyliu/>，谢谢合作

[http://www.cnblogs.com/sammyliu/](#)，谢谢合作

0

世民谈云计算

（声明：本站文章皆基于公开来源信息，仅代表作者个人观点，与作者所在公司无关）

昵称：SammyLiu  
园龄：2年6个月  
荣誉：推荐博客  
粉丝：470  
关注：30  
+加关注

常用链接

我的随笔  
我的评论  
我的参与  
最新评论  
我的标签

我的标签

GRE(1)  
Neutron(1)  
Open vSwitch(1)  
OpenStack(1)

随笔分类 (25)

Ceilometer(3)  
Ceph(13)  
Cinder(6)  
Docker(8)  
Glance  
Heat(2)  
K8S  
Keystone(1)  
KVM(10)  
MessageQueue(4)  
MySQL(1)  
Neutron(17)  
Nova(10)  
OpenStack(33)  
Sahara  
Storage(1)  
Swift(3)  
Trove  
Ubuntu(3)  
VMware(3)  
安装和配置(1)  
版本(4)  
备份(1)  
大数据(5)  
翻译(4)  
高可用（HA）(6)  
基础知识(19)  
监控(1)  
容器(4)  
容器编排  
使用案例(4)  
网络(8)  
问题定位(3)  
行业(14)  
性能(4)  
虚拟化(7)  
原理(22)  
云Cloud(29)

随笔档案 (121)

2017年5月 (1)  
2017年3月 (1)  
2017年1月 (1)  
2016年10月 (7)  
2016年9月 (5)  
2016年8月 (4)  
2016年7月 (1)  
2016年6月 (5)  
2016年5月 (1)  
2016年4月 (1)  
2016年3月 (9)  
2016年2月 (4)  
2016年1月 (2)

博客园 首页 新随笔 订阅 管理

随笔-121 评论-504 文章-45

KVM 介绍（2）：CPU 和内存虚拟化

学习 KVM 的系列文章：

- (1) 介绍和安装
- (2) CPU 和 内存虚拟化
- (3) I/O QEMU 全虚拟化和准虚拟化（Para-virtualization）
- (4) I/O PCI/PCIe设备直接分配和 SR-IOV
- (5) libvirt 介绍
- (6) Nova 通过 libvirt 管理 QEMU/KVM 虚拟机
- (7) 快照（snapshot）
- (8) 迁移（migration）

1. 为什么需要 CPU 虚拟化

x86 操作系统是设计在直接运行在裸硬件设备上的，因此它们自动认为它们完全占有计算机硬件。x86 架构提供四个特权级别给操作系统和应用程序来访问硬件。Ring 是指 CPU 的运行级别，Ring 0 是最高级别，Ring 1 次之，Ring 2 更次之……就 Linux+x86 来说，

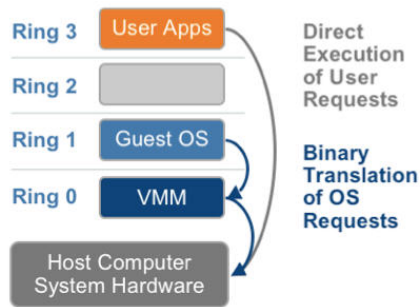
- 操作系统（内核）需要直接访问硬件和内存，因此它的代码需要运行在最高运行级别 Ring 0 上，这样它可以使用特权指令，控制中断、修改页表、访问设备等等。
- 应用程序的代码运行在最低运行级别上 ring 3 上，不能做受控操作。如果要做的，比如要访问磁盘，写文件，那就要通过执行系统调用（函数），执行系统调用的时候，CPU 的运行级别会发生从 ring 3 到 ring 0 的切换，并跳转到系统调用对应的内核代码位置执行，这样内核就为你完成了设备访问，完成之后再从 ring 0 返回 ring 3。这个过程也称作用户态和内核态的切换。



那么，虚拟化在这里就遇到了一个难题，因为宿主操作系统是工作在 ring 0 的，客户操作系统就不能也在 ring 0 了，但是它不知道这一点，以前执行什么指令，现在还是执行什么指令，但是没有执行权限是会出错的。所以这时候虚拟机管理程序（VMM）需要避免这件事情发生。虚拟机怎么通过 VMM 实现 Guest CPU 对硬件的访问，根据其原理不同有三种实现技术：

- 全虚拟化
- 半虚拟化
- 硬件辅助的虚拟化

1.1 基于二进制翻译的全虚拟化（Full Virtualization with Binary Translation）



客户操作系统运行在 Ring 1，它在执行特权指令时，会触发异常（CPU 的机制，没权限的指令会触发异常），然后 VMM 捕获这个异常，在异常里面做翻译，模拟，最后返回到客户操作系统内，客户操作系统认为自己的特权指令工作正常，继续运行。但是这个性能损耗，就非常的大，简单的一条指令，执行完，了事，现在却要经过复杂的异常处理过程。

异常“捕获（trap）、翻译（handle）、模拟（emulate）”过程。

2015年12月 (7)

2015年11月 (7)

2015年10月 (4)

2015年9月 (4)

2015年8月 (5)

2015年7月 (9)

2015年6月 (10)

2015年5月 (3)

2015年4月 (11)

2015年3月 (2)

2015年2月 (6)

2015年1月 (5)

2014年12月 (6)

文章分类 (21)

Ceph(1)

GlusterFS

Web 服务器(2)

操作系统(1)

存储

大数据(2)

分布式系统

服务器(1)

网络(11)

虚拟化(3)

云

文章档案 (42)

2016年10月 (2)

2016年9月 (1)

2016年6月 (1)

2016年5月 (3)

2015年12月 (4)

2015年10月 (5)

2015年9月 (2)

2015年6月 (1)

2015年4月 (23)

积分与排名

积分 - 286831

排名 - 535

最新评论

1. Re:Neutron 理解 (1): Neutron 所实现的虚拟化网络 [How Netruon Virtualizes Network]

eth1 - 公共网络 (untagged), 管理网络 (tag=102), 存储网络 (tag=103) 不好意思, 大家共用同一个eth1端口的时候, 请问这里交换机端口是配置为tagged还是untagged.....

--xianke9

2. Re:理解Docker (5) : Docker 网络

1.12版本上网络的表现如何?

--幽灵狼

3. Re:理解Docker (5) : Docker 网络

我想请问一下运行docker quickstart terminal时一直卡在"waiting for an IP"应该如何解决呢? 希望楼主能解答一下。

--silentbell

4. Re:理解Docker (6) : 若干企业生产环境中的容器网络方案写得好好! 加油。

--itbj00

5. Re:理解Docker (5) : Docker 网络

非常好, 写得很详细。加油!

--itbj00

阅读排行榜

1. Neutron 理解 (1): Neutron 所实现的虚拟化网络 [How Netruon Virtualizes Network](22087)

2. 理解 OpenStack 高可用 (HA) (1) : OpenStack 高可用和灾备方案 [OpenStack HA and DR](13707)

3. Neutron 理解 (3): Open vSwitch + GRE/VxLAN 组网 [Netruon Open vSwitch + GRE/VxLAN Virtual Network](13434)

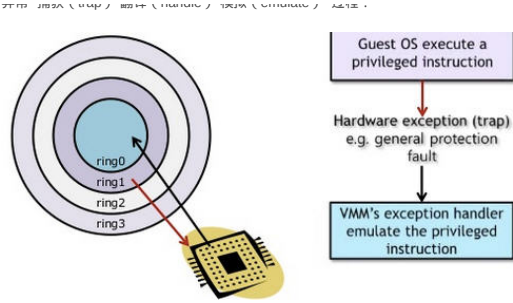
4. 探索 OpenStack 之 (9) : 深入块存储服务Cinder (功能篇) (12921)

5. 理解 OpenStack + Ceph (1) : Ceph + OpenStack 集群部署和配置 (12444)

评论排行榜

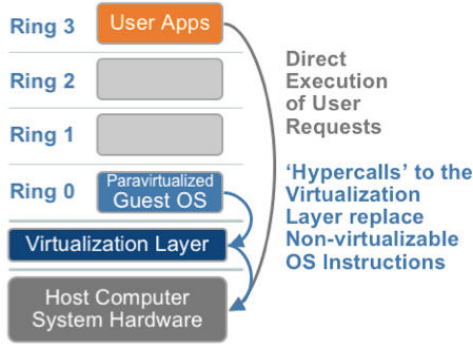
1. Neutron 理解 (1): Neutron 所实现的虚拟化网络 [How Netruon Virtualizes Network](63)

2. Neutron 理解 (14) : Neutron



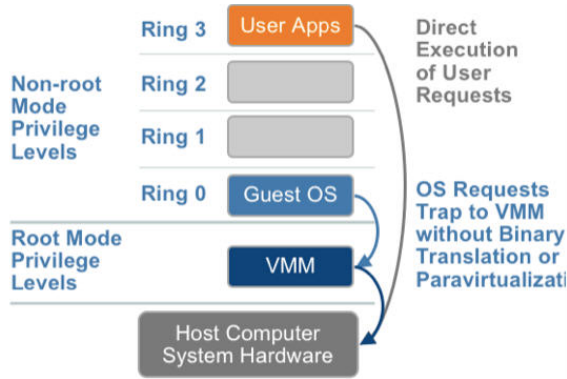
1.2. 超虚拟化 (或者半虚拟化/操作系统辅助虚拟化 Paravirtualization)

半虚拟化的思想就是, 修改操作系统内核, 替换掉不能虚拟化的指令, 通过超级调用 (hypercall) 直接和底层的虚拟化层 hypervisor 来通讯, hypervisor 同时也提供了超级调用接口来满足其他关键内核操作, 比如内存管理、中断和时间保持。这种做法省去了全虚拟化中的捕获和模拟, 大大提高了效率。所以像XEN这种半虚拟化技术, 客户机操作系统都是有一个专门的定制内核版本, 和x86、mips、arm这些内核版本等价。这样以来, 就不会有捕获异常、翻译、模拟的过程了, 性能损耗非常低。这就是XEN这种半虚拟化架构的优势。这也是为什么XEN只支持虚拟化Linux, 无法虚拟化windows原因, 微软不改代码啊。



1.3. 硬件辅助的全虚拟化

2005年后, CPU厂商Intel 和 AMD 开始支持虚拟化了。Intel 引入了 Intel-VT (Virtualization Technology) 技术。这种 CPU, 有 VMX root operation 和 VMX non-root operation 两种模式, 两种模式都支持Ring 0 ~ Ring 3 共 4 个运行级别。这样, VMM 可以运行在 VMX root operation 模式下, 客户 OS 运行在 VMX non-root operation 模式下。



而且两种操作模式可以互相转换。运行在 VMX root operation 模式下的 VMM 通过显式调用 VMLAUNCH 或 VMRESUME 指令切换到 VMX non-root operation 模式, 硬件自动加载 Guest OS 的上下文, 于是 Guest OS 获得运行, 这种转换称为 VM entry。Guest OS 运行过程中遇到需要 VMM 处理的事件, 例如外部中断或缺页异常, 或者主动调用 VMCALL 指令调用 VMM 的服务的时候 (与系统调用类似), 硬件自动挂起 Guest OS, 切换到 VMX root operation 模式, 恢复 VMM 的运行, 这种转换称为 VM exit。VMX root operation 模式下软件的行为与在没有 VT-x 技术的处理器上的行为基本一致; 而VMX non-root operation 模式则有很大不同, 最主要的区别是此时运行某些指令或遇到某些事件时, 发生 VM exit。

也就说, 硬件这层就做了些区分, 这样全虚拟化下, 那些靠“捕获异常-翻译-模拟”的实现就不需要了。而且CPU厂商, 支持虚拟化的力度越来越大, 靠硬件辅助的全虚拟化技术的性能逐渐逼近半虚拟化, 再加上全虚拟化不需要修改客户操作系统这一优势, 全虚拟化技术应该是未来的发展趋势。

	利用二进制翻译的全虚拟化	硬件辅助虚拟化	操作系统协助/半虚拟化
实现技术	BT和直接执行	遇到特权指令转到root模式执行	Hypercall
客户操作系统修改/兼容性	无需修改客户操作系统, 最佳兼容性	无需修改客户操作系统, 最佳兼容性	客户操作系统需要修改来支持hypercall, 因此它不能运行在物理硬件本身或其他的hypervisor上, 兼容性差, 不支持Windows

ML2 + Linux bridge + VxLAN 组网 (54)

3. Neutron 理解 (8): Neutron 是如何实现虚拟机防火墙的 [How Neutron Implements Security Group](34)

4. Neutron 理解 (3): Open vSwitch + GRE/VxLAN 组网 [Netruon Open vSwitch + GRE/VxLAN Virtual Network](25)

5. Neutron 理解 (5) : Neutron 是如何向 Nova 虚拟机分配固定IP地址的 ( How Neutron Allocates Fixed IPs to Nova Instance ) (21)

推荐排行榜

1. Neutron 理解 (1): Neutron 所实现的虚拟化网络 [How Netruon Virtualizes Network](9)

2. 我所了解的 京东、携程、eBay、小米的 OpenStack 云(6)

3. 理解 OpenStack 高可用 ( HA ) ( 1 ) : OpenStack 高可用和灾备方案 [OpenStack HA and DR](6)

4. Neutron 理解 (2): 使用 Open vSwitch + VLAN 组网 [Netruon Open vSwitch + VLAN Virtual Network](6)

5. 理解 OpenStack 高可用 ( HA ) ( 2 ) : Neutron L3 Agent HA 之 虚拟路由冗余协议 ( VRRP ) (5)

性能	差	性能开销；但是，其性能在逐渐逼近半虚拟化。	能接近于物理机。
应用厂商	VMware Workstation/QEMU/Virtual PC	VMware ESXi/Microsoft Hyper-V/Xen 3.0/KVM	Xen

2. KVM CPU 虚拟化

KVM 是基于CPU 辅助的全虚拟化方案，它需要CPU虚拟化特性的支持。

2.1. CPU 物理特性

这个命令查看主机上的CPU 物理情况：

```
[s1@rh65 ~]$ numactl --hardware
available: 2 nodes (0-1) //2颗CPU
node 0 cpus: 0 1 2 3 4 5 12 13 14 15 16 17 //这颗 CPU 有8个内核
node 0 size: 12276 MB
node 0 free: 7060 MB
node 1 cpus: 6 7 8 9 10 11 18 19 20 21 22 23
node 1 size: 8192 MB
node 1 free: 6773 MB
node distances:
node 0 1
0: 10 21
1: 21 10
```

要支持 KVM，Intel CPU 的 vmx 或者 AMD CPU 的 svm 扩展必须生效了：

```
[root@rh65 s1]# egrep "(vmx|svm)" /proc/cpuinfo
flags              : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi
mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good
xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr
pdcm pcid dca sse4_1 sse4_2 popcnt aes lahf_lm arat epb dts tpr_shadow vmni flexpriority ept vpid
```

2.2 多 CPU 服务器架构：SMP，NMP，NUMA

从系统架构来看，目前的商用服务器大体可以分为三类：

- 多处理器结构 (SMP：Symmetric Multi-Processor)：所有的CPU共享全部资源，如总线，内存和I/O系统等，操作系统或管理数据库的副本只有一个，这种系统有一个最大的特点就是共享所有资源。多个CPU之间没有区别，平等地访问内存、外设、一个操作系统。SMP 服务器的主要问题，那就是它的扩展能力非常有限。实验证明，SMP 服务器 CPU 利用率最好的情况是 2 至 4 个 CPU。
- 海量并行处理结构 (MPP：Massive Parallel Processing)：NUMA 服务器的基本特征是具有多个 CPU 模块，每个 CPU 模块由多个 CPU(如 4 个)组成，并且具有独立的本地内存、I/O 槽口等。在一个物理服务器内可以支持上百个 CPU。但 NUMA 技术同样有一定缺陷，由于访问远地内存的延时远远超过本地内存，因此当 CPU 数量增加时，系统性能无法线性增加。
- 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。
- 非一致存储访问结构 (NUMA：Non-Uniform Memory Access)：它由多个 SMP 服务器通过一定的节点互联网络进行连接，协同工作，完成相同的任务，从用户的角度来看是一个服务器系统。其基本特征是由多个 SMP 服务器（每个 SMP 服务器称节点）通过节点互联网络连接而成，每个节点只访问自己的本地资源（内存、存储等），是一种完全无共享 (Share Nothing) 结构。

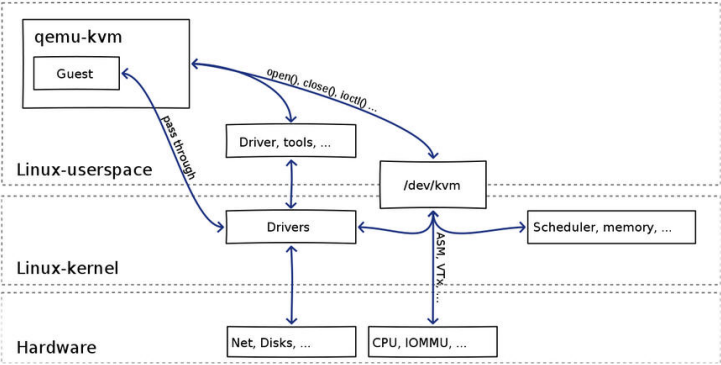
详细描述可以参考 [SMP、NUMA、MPP体系结构介绍](#)。

查看你的服务器的 CPU 架构：

```
[root@rh65 s1]# uname -a
Linux rh65 2.6.32-431.el6.x86_64 #1 SMP Sun Nov 10 22:19:54 EST 2013 x86_64 x86_64 x86_64 GNU/Linux #这服务器是 SMP 架构
```

2.2 KVM CPU 虚拟化

2.2.1 KVM 虚机的创建过程



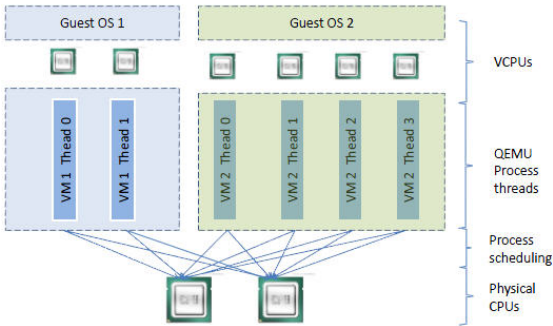
可见：

(1) qemu-kvm 通过对 /dev/kvm 的一系列 IOCTL 命令控制虚机，比如

```
open("/dev/kvm", O_RDWR|O_LARGEFILE) = 3
ioctl(3, KVM_GET_API_VERSION, 0) = 12
ioctl(3, KVM_CHECK_EXTENSION, 0x19) = 0
ioctl(3, KVM_CREATE_VM, 0) = 4
ioctl(3, KVM_CHECK_EXTENSION, 0x4) = 1
ioctl(3, KVM_CHECK_EXTENSION, 0x4) = 1
ioctl(4, KVM_SET_TSS_ADDR, 0xffffb000) = 0
ioctl(3, KVM_CHECK_EXTENSION, 0x25) = 0
ioctl(3, KVM_CHECK_EXTENSION, 0xb) = 1
ioctl(4, KVM_CREATE_PIT, 0xb) = 0
ioctl(3, KVM_CHECK_EXTENSION, 0xf) = 2
ioctl(3, KVM_CHECK_EXTENSION, 0x3) = 1
ioctl(3, KVM_CHECK_EXTENSION, 0) = 1
ioctl(4, KVM_CREATE_IRQCHIP, 0) = 0
ioctl(3, KVM_CHECK_EXTENSION, 0x1a) = 0
```

(2) 一个 KVM 虚机即一个 Linux qemu-kvm 进程，与其他 Linux 进程一样被Linux 进程调度器调度。

- (3) KVM 虚拟机包括虚拟内存、虚拟CPU和虚拟机 I/O设备，其中，内存和 CPU 的虚拟化由 KVM 内核模块负责实现，I/O 设备的虚拟化由 QEMU 负责实现。
  - (3) KVM用户系统的内存是 qemu-kvm 进程的地址空间的一部分。
  - (4) KVM 虚拟机的 vCPU 作为 线程运行在 qemu-kvm 进程的上下文中。
- vCPU、QEMU 进程、Linux 进程调度和物理CPU之间的逻辑关系：

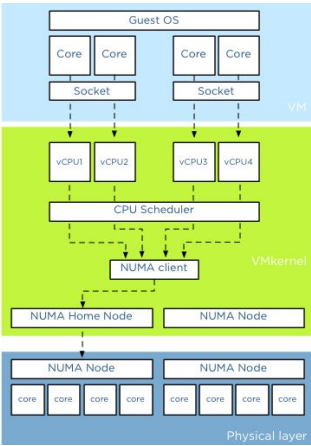


2.2.2 因为 CPU 中的虚拟化功能的支持，并不存在虚拟的 CPU，KVM Guest 代码是运行在物理 CPU 之上

根据上面的 1.3 章节，支持虚拟化的 CPU 中都增加了新的功能。以 Intel VT 技术为例，它增加了两种运行模式：VMX root 模式和 VMX nonroot 模式。通常来讲，主机操作系统和 VMM 运行在 VMX root 模式中，客户机操作系统及其应用运行在 VMX nonroot 模式中。因为两个模式都支持所有的 ring，因此，客户机可以运行在它所需要的 ring 中（OS 运行在 ring 0 中，应用运行在 ring 3 中），VMM 也运行在其需要的 ring 中（对 KVM 来说，QEMU 运行在 ring 3，KVM 运行在 ring 0）。CPU 在两种模式之间的切换称为 VMX 切换。从 root mode 进入 nonroot mode，称为 VM entry；从 nonroot mode 进入 root mode，称为 VM exit。可见，CPU 受控制地在两种模式之间切换，轮流执行 VMM 代码和 Guest OS 代码。

对 KVM 虚拟机来说，运行在 VMX Root Mode 下的 VMM 在需要执行 Guest OS 指令时执行 VMLAUNCH 指令将 CPU 转换到 VMX non-root mode，开始执行客户机代码，即 VM entry 过程；在 Guest OS 需要退出该 mode 时，CPU 自动切换到 VMX Root mode，即 VM exit 过程。可见，KVM 客户机代码是受 VMM 控制直接运行在物理 CPU 上的。QEMU 只是通过 KVM 控制虚拟机的代码被 CPU 执行，但是它们本身并不执行其代码。也就是说，CPU 并没有真正的被虚级化成虚拟的 CPU 给客户机使用。

这篇文章是关于 vSphere 中 CPU 虚拟化的，我觉得它和 KVM CPU 虚拟化存在很大的一致。下图是使用 2 socket 2 core 共 4 个 vCPU 的情形：



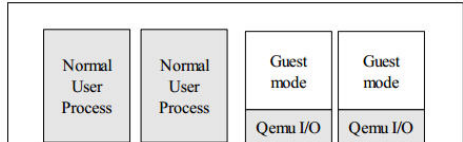
几个概念：socket（颗，CPU 的物理单位），core（核，每个 CPU 中的物理内核），thread（超线程，通常来说，一个 CPU core 只提供一个 thread，这时客户机就只看到一个 CPU；但是，超线程技术实现了 CPU 核的虚拟化，一个核被虚拟化出多个逻辑 CPU，可以同时运行多个线程）。

上图分三层，他们分别是 VM 层，VMkernel 层和物理层。对于物理服务器而言，所有的 CPU 资源都分配给单独的操作系统和上面运行的应用。应用将请求先发送给操作系统，然后操作系统调度物理的 CPU 资源。在虚拟化平台比如 KVM 中，在 VM 层和物理层之间加入了 VMkernel 层，从而允许所有的 VM 共享物理层的资源。VM 上的应用将请求发送给 VM 上的操作系统，然后操作系统调度 Virtual CPU 资源（操作系统认为 Virtual CPU 和物理 CPU 是一样的），然后 VMkernel 层对多个物理 CPU Core 进行资源调度，从而满足 Virtual CPU 的需要。在虚拟化平台中 OS CPU Scheduler 和 Hypervisor CPU Scheduler 都在各自的领域内进行资源调度。

KVM 中，可以指定 socket，core 和 thread 的数目，比如 设置“-smp 5,sockets=5,cores=1,threads=1”，则 vCPU 的数目为 5\*1\*1 = 5。客户机看到的是基于 KVM vCPU 的 CPU 核，而 vCPU 作为 QEMU 线程被 Linux 作为普通的线程/轻量级进程调度到物理的 CPU 核上。至于你是该使用多 socket 和多 core，这篇文章有仔细的分析，其结论是在 VMware ESXi 上，性能没什么区别，只是某些客户机操作系统会限制物理 CPU 的数目，这种情况下，可以使用少 socket 多 core。

2.2.3 客户机系统的代码是如何运行的

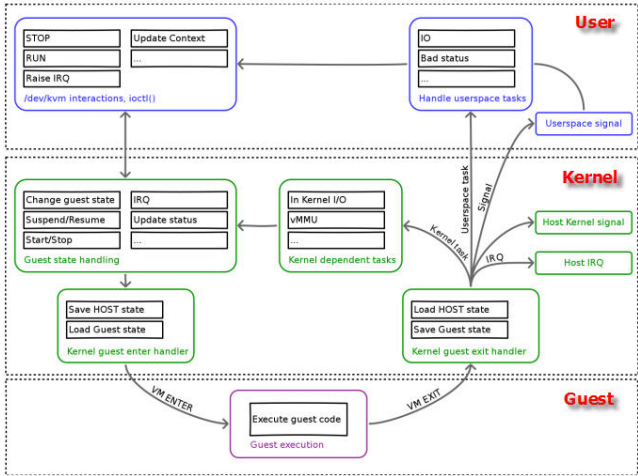
一个普通的 Linux 内核有两种执行模式：内核模式（Kernal）和用户模式（User）。为了支持带有虚拟化功能的 CPU，KVM 向 Linux 内核增加了第三种模式即客户机模式（Guest），该模式对应于 CPU 的 VMX non-root mode。







- KVM 内核模块作为 User mode 和 Guest mode 之间的桥梁：
- User mode 中的 QEMU-KVM 会通过 ICOTL 命令来运行虚拟机
  - KVM 内核模块收到该请求后，它先做一些准备工作，比如将 VCPU 上下文加载到 VMCS（virtual machine control structure）等，然后驱动 CPU 进入 VMX non-root 模式，开始执行客户机代码
- 三种模式的分工为：
- Guest 模式：执行客户机系统非 I/O 代码，并在需要的时候驱动 CPU 退出该模式
  - Kernel 模式：负责将 CPU 切换到 Guest mode 执行 Guest OS 代码，并在 CPU 退出 Guest mode 时回到 Kernel 模式
  - User 模式：代表客户机系统执行 I/O 操作



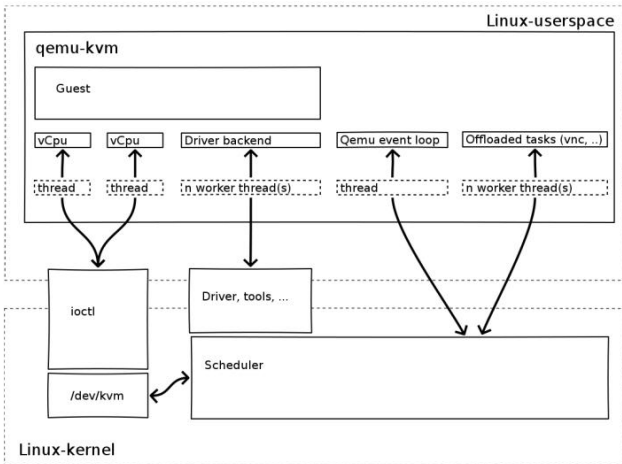
(来源)

- QEMU-KVM 相比原生 QEMU 的改动：
- 原生的 QEMU 通过指令翻译实现 CPU 的完全虚拟化，但是修改后的 QEMU-KVM 会调用 ICOTL 命令来调用 KVM 模块。
  - 原生的 QEMU 是单线程实现，QEMU-KVM 是多线程实现。
- 主机 Linux 将一个虚拟视作一个 QEMU 进程，该进程包括下面几种线程：
- I/O 线程用于管理模拟设备
  - vCPU 线程用于运行 Guest 代码
  - 其它线程，比如处理 event loop，offloaded tasks 等的线程

在我的测试环境中（RedHata Linux 作 Hypervisor）：

smp 设置的值	线程数	线程
4	8	1 个主线程（I/O 线程）、4 个 vCPU 线程、3 个其它线程
6	10	1 个主线程（I/O 线程）、6 个 vCPU 线程、3 个其它线程

这篇文章 谈了这些线程的情况。



(来源)

客户机代码执行（客户机线程）	I/O 线程	非 I/O 线程
虚拟CPU（主机 QEMU 线程）	QEMU I/O 线程	QEMU vCPU 线程
物理 CPU	物理 CPU 的 VMX non-root 模式中	物理 CPU 的 VMX non-root 模式中

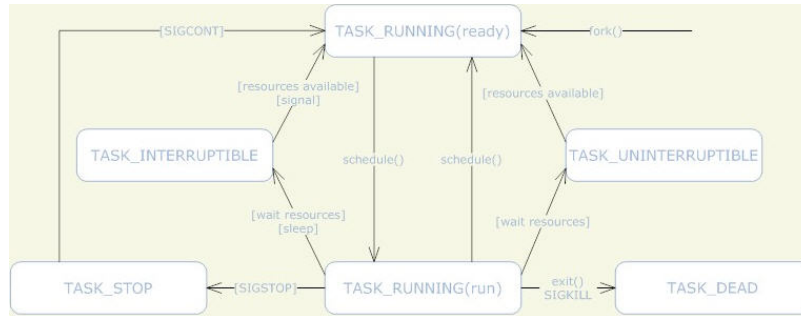
### 2.2.4 从客户机线程到物理 CPU 的两次调度

要将客户机内的线程调度到某个物理 CPU，需要经历两个过程：

1. 客户机线程调度到客户机物理CPU 即 KVM vCPU，该调度由客户机操作系统负责，每个客户机操作系统的实现方式不同。在 KVM 上，vCPU 在客户机系统看起来就像是物理 CPU，因此其调度方法也没有什么不同。
2. vCPU 线程调度到物理 CPU 即主机物理 CPU，该调度由 Hypervisor 即 Linux 负责。

KVM 使用标准的 Linux 进程调度方法来调度 vCPU 进程。Linux 系统中，线程和进程的区别是 进程有独立的内核空间，线程是代码的执行单位，也就是调度的基本单位。Linux 中，线程就是轻量级的进程，也就是共享了部分资源(地址空间、文件句柄、信号量等等)的进程，所以线程也按照进程的调度方式来进行调度。

(1) Linux 进程调度原理可以参考 [这篇文章](#) 和 [这篇文章](#)。通常情况下，在 SMP 系统中，Linux 内核的进程调度器根据自有的调度策略将系统中的一个可运行 (runnable) 进程调度到某个 CPU 上执行。下面是 Linux 进程的状态机：



(2) 处理器亲和性：可以设置 vCPU 在指定的物理 CPU 上运行，具体可以参考[这篇文章](#) 和 [这篇文章](#)。

根据 Linux 进程调度策略，可以看出，在 Linux 主机上运行的 KVM 客户机的总 vCPU 数目最好不要超过物理 CPU 内核数，否则，会出现线程间的 CPU 内核资源竞争，导致有虚拟机因为 vCPU 进程等待而导致速度很慢。

关于这两次调度，业界有很多的研究，比如上海交大的论文 [Schedule Processes, not VCPUs](#) 提出动态地减少 vCPU 的数目即减少第二次调度。

另外，[这篇文章](#) 谈到的是 vSphere CPU 的调度方式，有空的时候可以研究下并和 KVM vCPU 的调度方式进行比较。

### 2.3 客户机 CPU 结构和模型

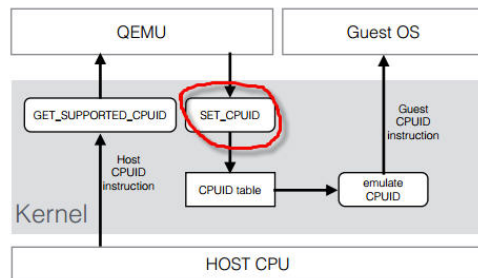
KVM 支持 SMP 和 NUMA 多 CPU 架构的主机和客户机。对 SMP 类型的客户机，使用 "-smp" 参数：

```
-smp <n>[,cores=<ncores>][,threads=<nthreads>][,sockets=<nsockets>][,maxcpus=<maxcpus>]
```

对 NUMA 类型的客户机，使用 "-numa" 参数：

```
-numa <nodes>[,mem=<size>][,cpus=<cpu[-cpu]>][,nodeid=<node>]
```

CPU 模型 (models) 定义了哪些主机的 CPU 功能 (features) 会被暴露给客户机操作系统。为了在具有不同 CPU 功能的主机之间做安全的迁移，qemu-kvm 往往不会将主机 CPU 的所有功能都暴露给客户机。其原理如下：



你可以运行 `qemu-kvm -cpu ?` 命令来获取主机所支持的 CPU 模型列表。

```

[root@rh65 s1]# kvm -cpu ?
x86   Opteron_G5  AMD Opteron 63xx  class CPU
x86   Opteron_G4  AMD Opteron 62xx  class CPU
x86   Opteron_G3  AMD Opteron 23xx  (Gen 3 Class Opteron)
x86   Opteron_G2  AMD Opteron 22xx  (Gen 2 Class Opteron)
x86   Opteron_G1  AMD Opteron 240   (Gen 1 Class Opteron)
x86   Haswell     Intel Core Processor (Haswell)
x86   SandyBridge Intel Xeon E312xx (Sandy Bridge)
x86   Westmere    Westmere E56xx/L56xx/X56xx (Nehalem-C)
x86   Nehalem     Intel Core i7 9xx (Nehalem Class Core i7)
x86   Penryn      Intel Core 2 Duo P9xxx (Penryn Class Core 2)
x86   Conroe      Intel Celeron 4x0 (Conroe/Merom Class Core 2)
x86   cpu64-rhe15  QEMU Virtual CPU version (cpu64-rhe15)
x86   cpu64-rhe16  QEMU Virtual CPU version (cpu64-rhe16)
x86   n270        Intel(R) Atom(TM) CPU N270 @ 1.60GHz
x86   athlon      QEMU Virtual CPU version 0.12.1
x86   pentium3
x86   pentium2
x86   pentium
x86   486
x86   coreduo     Genuine Intel(R) CPU T2600 @ 2.16GHz
x86   qemu32      QEMU Virtual CPU version 0.12.1
x86   kvm64       Common KVM processor
x86   core2duo    Intel(R) Core(TM)2 Duo CPU T7700 @ 2.40GHz
x86   phenom      AMD Phenom(tm) 9550 Quad-Core Processor
x86   qemu64      QEMU Virtual CPU version 0.12.1

Recognized CPUID flags:
  f_edx: pbe ia64 tm ht ss sse2 sse fxsr mmx acpi ds clflush pn pse36 pat cmov mca pge mtrr sep apic
cx8 mce pae msr tsc pse de vme fpu
  f_ecx: hypervisor rdrand f16c avx osxsave xsave aes tsc-deadline popcnt movbe x2apic sse4.2|sse4_2
sse4.1|sse4_1 dca pcid pdcm xtrp cx16 fma cid sse3 tm2 est smx vmx ds_cpl monitor dtes64
pclmulqdq|pclmuldq pni|sse3
  extf_edx: 3dnow 3dnowext lm|ia64 rdtscp pdpe1gb fxsr_opt|ffxsr fxsr mmx mmxext nx|xd pse36 pat cmov
mca pge mtrr syscall apic cx8 mce pae msr tsc pse de vme fpu
  
```

```
extf_ecx: perfctr_nb perfctr_core topoext_tbm nodeid_msr tce fma4 lwp wdt skinit xop ibs osvw
3dnowprefetch misalignsse sse4a abm cr8legacy extapic svm cmp_legacy lahf_lm
[root@rh65 s1]#
```

每个 Hypervisor 都有自己的策略，来定义默认上哪些CPU功能会被暴露给客户机。至于哪些功能会被暴露给客户机系统，取决于客户机的配置。qemu32 和 qemu64 是基本的客户机 CPU 模型，但是还有其他的模型可以使用。你可以使用 qemu-kvm 命令的 -cpu <model> 参数来指定客户机的 CPU 模型，还可以附加指定的 CPU 特性。“-cpu”会将该指定 CPU 模型的所有功能全部暴露给客户机，即使某些特性在主机的物理CPU上不支持，这时候QEMU/KVM 会模拟这些特性，因此，这时候也许会出现一定的性能下降。

RedHat Linux 6 上使用默认的 cpu64-rhe16 作为客户机 CPU model：

```
address sizes      : 40 bits physical, 40 bits virtual
power management:

processor          : 1
vendor_id          : GenuineIntel
cpu family         : 6
model              : 13
model name         : QEMU Virtual CPU version (cpu64-rhel6)
stepping           : 3
cpu MHz            : 2666.760
cache size         : 4096 KB
cpu                : yes
cpu_exception      : yes
cpuid level        : 4
wp                 : yes
flags              : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pse36
clflush mmx fxsr sse sse2 syscall nx lm unfair_spinlock pni cx16 hypervisor lahf_lm
bogomips           : 5333.52
clflush size       : 64
cache alignment    : 64
address sizes      : 40 bits physical, 40 bits virtual
power management:
```

你可以指定特定的 CPU model 和 feature：

```
qemu-kvm -cpu Nehalem,+aes

model name      : Intel Core i7 9xx (Nehalem Class Core i7)
stepping        : 3
cpu MHz         : 2666.760
cache size      : 4096 KB
cpu             : yes
cpu_exception    : yes
cpuid level     : 4
wp              : yes
flags           : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 syscall nx lm constant_tsc unfair_spinlock pni s
sse3 cx16 sse4_1 sse4_2 x2apic popcnt aes hypervisor lahf_lm
```

你也可以直接使用 -cpu host，这样的话会客户机使用和主机相同的 CPU model。

2.4 客户机 vCPU 数目的分配方法

- 1. 不是客户机的 vCPU 越多，其性能就越好，因为线程切换会耗费大量的时间；应该根据负载需要分配最少的 vCPU。
- 2. 主机上的客户机的 vCPU 总数不应该超过物理 CPU 内核总数。不超过的话，就不存在 CPU 竞争，每个 vCPU 线程在一个物理 CPU 核上被执行；超过的话，会出现部分线程等待 CPU 以及一个 CPU 核上的线程之间的切换，这会有 overhead。
- 3. 将负载分为计算负载和 I/O 负载，对计算负载，需要分配较多的 vCPU，甚至考虑 CPU 亲和性，将指定的物理 CPU 核分给这些客户机。

这篇文章（<http://my.oschina.net/chape/blog/173981>）介绍了一些指导性方法，摘要如下：

我们来假设一个主机有 2 个 socket，每个 socket 有 4 个 core。主频 2.4G MHZ 那么一共可用的资源是 2\*4\*2.4G= 19.2G MHZ。假设主机上运行了三个 VM，VM1和VM2设置为1socket\*1core，VM3设置为1socket\*2core。那么VM1和VM2分别有1个vCPU，而VM3有2个vCPU。假设其他设置为缺省设置。

那么三个VM获得该主机CPU资源分配如下：VM1：25%；VM2：25%；VM3:50%

假设运行在VM3上的应用支持多线程，那么该应用可以充分利用到所非配的CPU资源。2vCPU的设置是合适的。假设运行在VM3上的应用不支持多线程，该应用根本无法同时使用利用2个vCPU。与此同时，VMkernel层的CPU Scheduler必须等待物理层中两个空闲的pCPU，才开始资源调配来满足2个vCPU的需要。在仅有2vCPU的情况下，对该VM的性能不会有太大负面影响。但如果分配4vCPU或者更多，这种资源调度上的负担有可能会对该VM上运行的应用有很大负面影响。

确定 vCPU 数目的步骤。假如我们要创建一个VM，以下几步可以帮助确定合适的vCPU数目

- 1 了解应用并设置初始值
  - 该应用是否是关键应用，是否有Service Level Agreement。一定要对运行在虚拟机上的应用是否支持多线程深入了解。咨询应用的提供商是否支持多线程和SMP（Symmetric-multi-processing）。参考该应用在物理服务器上运行时所需要的CPU个数。如果没有参照信息，可设置1vCPU作为初始值，然后密切观测资源使用情况。
- 2 观测资源使用情况
  - 确定一个时间段，观测该虚拟机的资源使用情况。时间段取决于应用的特点和要求，可以是数天，甚至数周。不仅观测该VM的CPU使用率，而且观测在操作系统内该应用对CPU的占用率。特别要区分CPU使用率平均值和CPU使用率峰值。
  - 假如分配有4个vCPU，如果在该VM上的应用的CPU
    - 使用峰值等于25%，也就是仅仅能最多使用25%的全部CPU资源，说明该应用是单线程的，仅能够使用一个vCPU（4 \* 25% = 1）
    - 平均值小于38%，而峰值小于45%，考虑减少 vCPU 数目
    - 平均值大于75%，而峰值大于90%，考虑增加 vCPU 数目
- 3 更改vCPU数目并观测结果

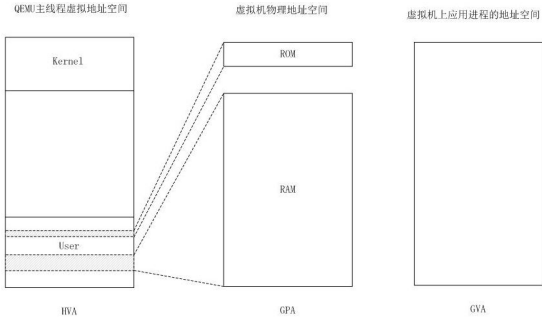
每次的改动尽量少，如果可能需要4vCPU，先设置2vCPU在观测性能是否可以接受。

2. KVM 内存虚拟化

2.1 内存虚拟化的概念

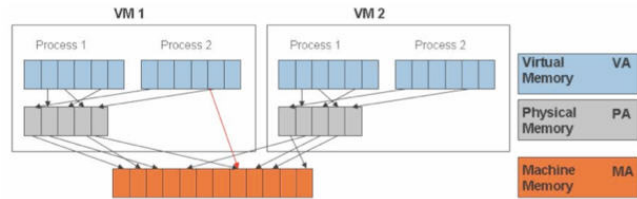
除了 CPU 虚拟化，另一个关键是内存虚拟化，通过内存虚拟化共享物理系统内存，动态分配给虚拟机。虚拟机的内存虚拟化很象现在的操作系统支持的虚拟内存方式，应用程序看到邻近的内存地址空间，这个地址空间无需和下面的物理机器内存直接对应，操作系统保持着虚拟页到物理页的映射。现在所有的 x86 CPU 都包括了一个称为内存管理的模块MMU（Memory Management Unit）和TLB(Translation Lookaside Buffer)，通过MMU和TLB来优化虚拟内存的性能。

KVM 实现客户机内存的方式是，利用mmap系统调用，在QEMU主线程的虚拟地址空间中申明一段连续的大小的空间用于客户机物理内存映射。



( 图片来源 HVA 同下面的 MA，GPA 同下面的 PA，GVA 同下面的 VA )

在有两个虚机的情况下，情形是这样的：



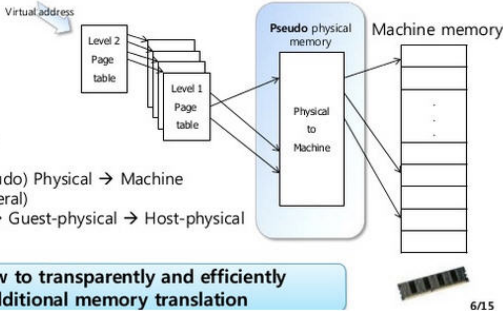
可见，KVM 为了在一台机器上运行多个虚拟机，需要增加一个新的内存虚拟化层，也就是说，必须虚拟 MMU 来支持客户操作系统，来实现 VA -> PA -> MA 的翻译。客户操作系统继续控制虚拟地址到客户内存物理地址的映射（VA -> PA），但是客户操作系统不能直接访问实际机器内存，因此VMM 需要负责映射客户物理内存到实际机器内存（PA -> MA）。

VMM 内存虚拟化的实现方式：

- 软件方式：通过软件实现内存地址的翻译，比如 Shadow page table（影子页表）技术
- 硬件实现：基于 CPU 的辅助虚拟化功能，比如 AMD 的 NPT 和 Intel 的 EPT 技术

影子页表技术：

- Virtual -> Physical -> Machine



Terminology

- Xen  
Virtual -> (Pseudo) Physical -> Machine
- Others (general)  
Guest-virtual -> Guest-physical -> Host-physical

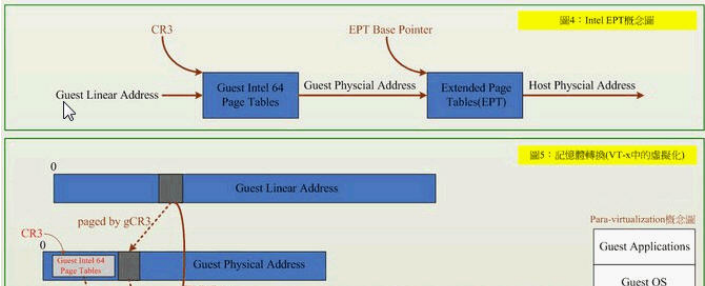
[Issue] How to transparently and efficiently manage additional memory translation

2.2 KVM 内存虚拟化

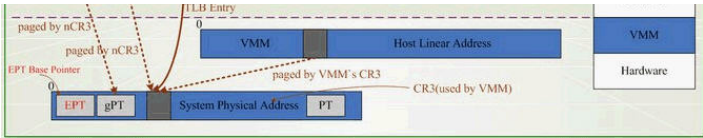
KVM 中，虚机的物理内存即为 qemu-kvm 进程所占用的内存空间。KVM 使用 CPU 辅助的内存虚拟化方式。在 Intel 和 AMD 平台，其内存虚拟化的实现方式分别为：

- AMD 平台上的 NPT（Nested Page Tables）技术
- Intel 平台上的 EPT（Extended Page Tables）技术

EPT 和 NPT采用类似的原理，都是作为 CPU 中新的一层，用来将客户机的物理地址翻译为主机的物理地址。关于 EPT，Intel 官方文档中的技术如下（实在看不懂...）







EPT的好处是，它的两阶段记忆体转换，特点就是将 Guest Physical Address → System Physical Address，VMM不用再保留一份 SPT (Shadow Page Table)，以及以往还得经过 SPT 这个转换过程。除了降低各部虚拟机在切换时所造成的效能损耗外，硬体指令集也比虚拟化软体处理来得可靠与稳定。

2.3 KSM (Kernel SamePage Merging 或者 Kernel Shared Memory)

KSM 在 Linux 2.6.32 版本中被加入到内核中。

2.3.1 原理

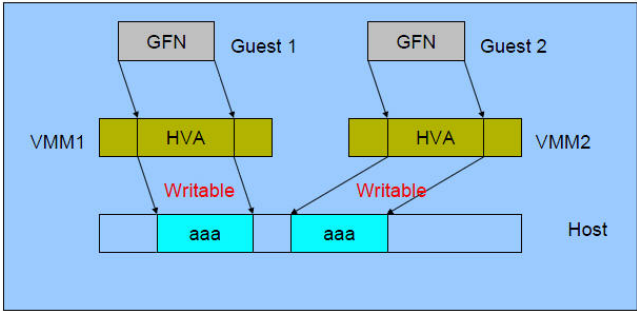
其原理是，KSM 作为内核中的守护进程（称为 ksmd）存在，它定期执行页面扫描，识别副本页面并合并副本，释放这些页面以供它用。因此，在多个进程中，Linux将内核相似的内存页合并成一个内存页。这个特性，被KVM用来减少多个相似的虚拟机的内存占用，提高内存的使用效率。由于内存是共享的，所以多个虚拟机使用的内存减少了。这个特性，对于虚拟机使用相同镜像和操作系统时，效果更加明显。但是，事情总是有代价的，使用这个特性，都要增加内核开销，用时间换空间。所以为了提高效率，可以将这个特性关闭。

2.3.2 好处

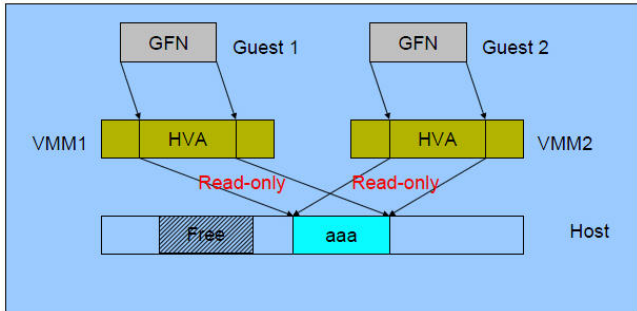
其好处是，在运行类似的客户机操作系统时，通过 KSM，可以节约大量的内存，从而可以实现更多的内存超分，运行更多的虚拟机。

2.3.3 合并过程

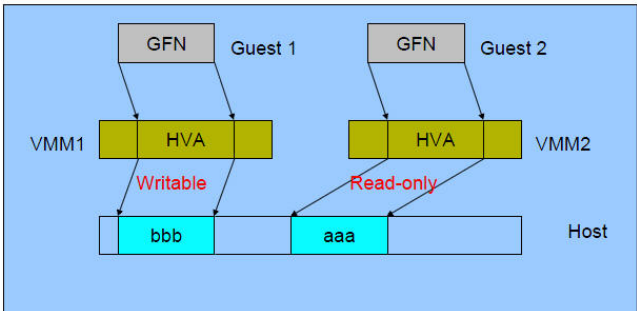
(1) 初始状态：



(2) 合并后：



(3) Guest 1 写内存后：



2.4 KVM Huge Page Backed Memory (巨页内存技术)

这是KVM虚拟机的又一个优化技术。Intel 的 x86 CPU 通常使用4Kb内存页，当是经过配置，也能够使用巨页(huge page): (4MB on x86\_32, 2MB on x86\_64 and x86\_32 PAE)

使用巨页，KVM的虚拟机的页表将使用更少的内存，并且将提高CPU的效率。最高情况下，可以提高20%的效率！

使用方法，需要三部：

```
mkdir /dev/hugepages
mount -t hugetlbfs hugetlbfs /dev/hugepages
#保留一些内存给巨页
sysctl vm.nr_hugepages=2048 (使用 x86_64 系统时，这相当于从物理内存中保留了2048 x 2M = 4GB 的空间来给虚拟机使用)
#给 kvm 传递参数 hugepages
qemu-kvm - qemu-kvm -mem-path /dev/hugepages
也可以在配置文件里加入：
<memoryBacking>
<hugepages/>
</memoryBacking>
```

验证方式，当虚拟机正常启动以后，在物理机里查看：

```
cat /proc/meminfo |grep -i hugepages
```

老外的一篇文档，他使用的是libvirt方式，先让libvirt进程使用hugepages空间，然后再分配给虚拟机。

参考资料：

- <http://www.cnblogs.com/xusongwei/archive/2012/07/30/2615592.html>
  - <https://www.ibm.com/developerworks/cn/linux/l-cn-vf/>
  - <http://www.slideshare.net/HwanjuKim/3cpu-virtualization-and-scheduling>
  - <http://www.cse.iitb.ac.in/~puru/courses/autumn12/cs695/classes/kvm-overview.pdf>
  - <http://www.linux-kvm.com/content/using-ksm-kernel-samepage-merging-kvm>
  - [http://blog.csdn.net/summer\\_liuwei/article/details/6013255](http://blog.csdn.net/summer_liuwei/article/details/6013255)
  - <http://blog.pchome.net/article/458429.html>
  - <http://blog.chinaunix.net/uid-20794164-id-3601787.html>
- 虚拟化技术性能比较和分析，周斌，张莹
- <http://wiki.qemu.org/images/c/c8/Cpu-models-and-libvirt-devconf-2014.pdf>
  - <http://frankdenneman.nl/2011/01/11/beating-a-dead-horse-using-cpu-affinity/>

分类: KVM

好文置顶 关注我 收藏该文





SammyLiu  
关注 - 30  
粉丝 - 470

荣誉：推荐博客  
+加关注

3 0

 推荐  反对

« 上一篇：KVM 介绍（1）：简介及安装  
» 下一篇：KVM 介绍（3）：I/O 全虚拟化和准虚拟化 [KVM I/O QEMU Full-Virtualization Para-virtualization]

posted on 2015-06-02 17:07 SammyLiu 阅读(9522) 评论(4) 编辑 收藏

评论:

- #1楼 2015-06-02 23:05 | 花儿笑弯了腰

不明觉厉

支持(0) 反对(0)
- #2楼 2015-12-28 11:29 | liunian0o0

膜拜

支持(0) 反对(0)
- #3楼 2016-03-16 10:59 | 叫声凯哥


膜拜膜拜，写的真是太棒了！！！！还有我也看不懂Intel的页表处理~~

支持(0) 反对(0)
- #4楼 2017-03-14 21:42 | JollyWing

世民兄，请教一个问题。  
Qemu在为虚拟机进程分配内存的时候，是否有将分配的内存空间清零的操作？  
我最近在做虚拟化安全的工作。  
不知道kvm在内存隔离这方面做得怎么样，请不吝赐教。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

- 【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库
- 【报表】Excel 报表开发18 招式，人人都能做报表
- 【活动】阿里云海外云服务全面降价助力企业全球布局
- 【实用】40+篇云服务器操作及运维基础知识！



最新IT新闻:

- 知乎上线视频功能，以后看教程更方便了
  - 一年只赚2万元：乐视游戏或被出售
  - Unity获得4亿美元投资，现估值为26亿美元
  - 直播对陌陌的意义，就像王者荣耀之于腾讯游戏
  - 死磕支付宝？苏宁金融发布“星辰计划”：扫码支付返888元
- » 更多新闻...



最新知识库文章:

· 程序员的工作、学习与绩效

· 软件开发为什么很难

· 唱吧DevOps的落地；微服务CI/CD的范本技术解读

· 程序员，如何从平庸走向理想？

· 我为什么鼓励工程师写blog

» 更多知识库文章...

Powered by: 博客园 模板提供：沪江博客 Copyright ©2017 SammyLiu