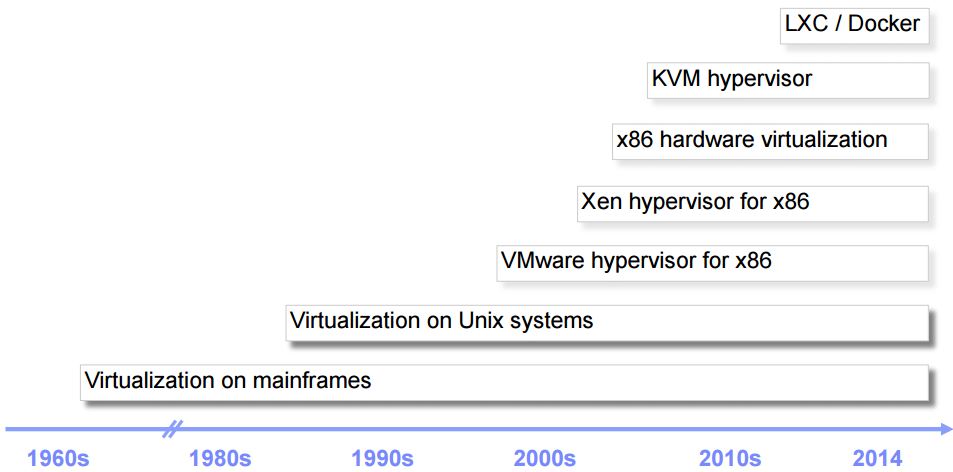
# kvm学习之路

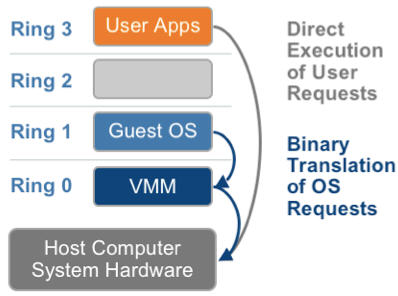
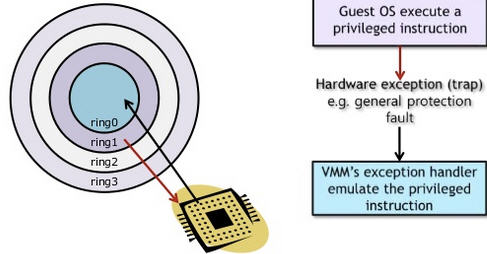
## 虚拟化简介



X86操作系统是设计在直接运行在裸硬件设备上的，因此它自动认为完全占有计算机硬件，X86平台的指令集权限划分为特权模式： Ring0, Ring1, Ring2, Ring3。 操作系统使用Ring0级别；应用程序使用Ring3级别。驱动程序使用Ring1,Ring2级别。应用程序不能做受控操作，如果需要做，比如要访问硬盘，写文件，那就要通过执行系统调用，执行系统调用的时候CPU的运行级别发生3到0的切换，并跳转到系统调用对应的内核代码位置执行，内核就为你完成设备访问，完成之后再从0返回3，这个过程也称为用户态的内核态的切换。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。

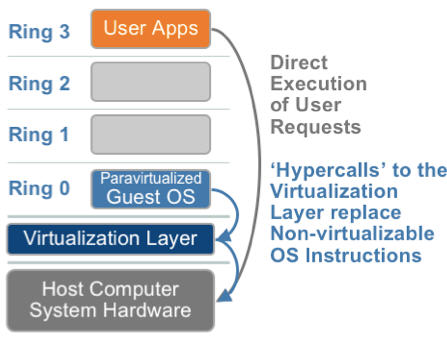
### 软件虚拟化、基于二进制翻译的全虚拟化（full Virtualization with Binary Translation）：

客户操作系统运行在Ring1,它运行特权指令时，会触发异常（CPU机制，没有权限的指令会触发异常）然后VMM （Virtual Machine Manager）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特权指令工作正常，继续运行。但这个性能损耗大。 典型厂商 VMware



### 1.2 半虚拟化/操作系统辅助虚拟化（Paravitualization）

半虚拟化的思想就是，修改操作系统的内核，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同时提供了超级调用接口，来满足其它关键内核操作，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，客户机操作系统都有一个专门的定制内核版本，这也是为什么XEN只能虚拟linux而不能虚拟windows的原因，微软不改代码



### 1.3 硬件辅助的全虚拟化

2005年后 ，CPU厂商Intel, 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,两种模式都支持Ring0－3共4个级别，这样VMM可以运行在VMX root operation下，客户OS运行在VMX non-root operation模式下。

这两种操作模式可以互相转换，运行在VMX root operation模式下的VMM通过显示调用VMLAUNC 或VMRESUME 指令切换到VMX non-root operation模式下，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，例如外部中断或者缺页异常，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。

硬件辅助的虚拟化不需要捕获异常，性能逐渐逼近半虚拟化，再加上不需要修改客户操作系统，已经是未来发展的趋势。 KVM 但是基于硬件辅助的全虚拟化。

## 硬件虚拟化及技术

### 2.1 CPU虚拟化

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的嵌套下去。

多CPU服务器架构

1. SMP （Symmetric Multi-Processor） 多处理理架构，所有的CPU共享全部资源，如总线，内存，IO系统等，这个系统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。SMP服务器主要问题就是它的扩展能力有限，实验证明，SMP服务器CPU利用最好的情况是2－4个CPU (一致内存访问 UMA)
2. MPP （Massive Parallel Processing）一种分布式存储模式，由多个SMP服务器通过一定的节点互联网进行连接，协同工作，完成相同的任务，每个节点都有自己的存储器（内存，存储），是一种完全无共享（Share Nothing）结构，因而扩展能力好，节点之间信息交互是通过节点互联网实现的，这个过程一般称为数据重分配（Data Redistribustion）
3. NUMA (Non-Uniform Memory Access): 非一致存储访问结构，具有多个CPU模块，每个CPU模块由多个CPU组成，并且具有独立的本地内存，IO槽。它由多个SMP服务器通过一定的节点互联网进行连接，协同工作，完成相同的任务。其节点之间可以通过模块（Crossbar Swith）进行连接和信息交互，因此每个CPU可以访问整个系统的内存。由于 访问远地内存的延时远远超过本地内存，因为当CPU数据增加时，系统性能无法线性增加。

查看NUMA信息与配置

numactl --hardware

numastat –c qemu-kvm

使用virsh查看numatune命令，

[root@cobbler ~]# virsh

Welcome to virsh, the virtualization interactive terminal.

Type: 'help' for help with commands

'quit' to quit

virsh # list -all

error: unexpected data '-all'

virsh # list --all

Id Name State

----------------------------------------------------

- CentOs shut off

- CentOs-7-x86\_64 shut off

virsh # numatune CentOs

numa\_mode : strict

numa\_nodeset : 0－1

在线设置cpu数量

virsh setvcpus Centos7 5 --live

在虚拟机里面将第5个CPU激活

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

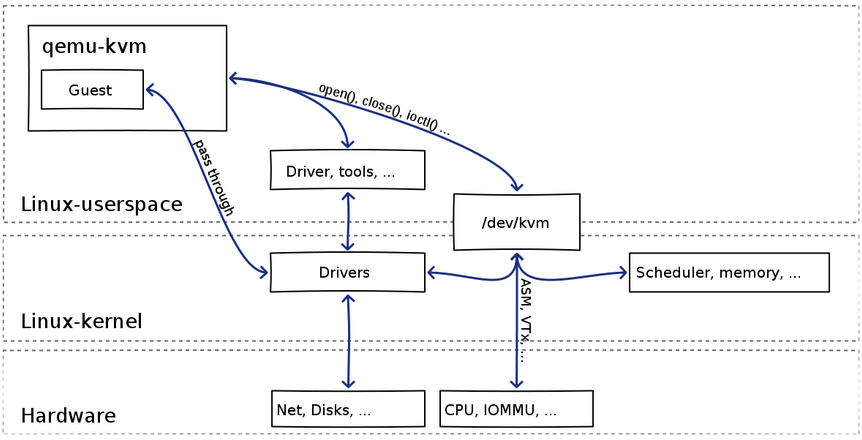
只能热增加，不能减少。

CPU模式配置：

1. custom模式，可以自己配置CPU型号
2. host-model 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。
3. host-passthrough 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就是物理CPU的型号。

CPU Netsted

nested就是在虚拟机上运行虚拟机，理论上KVM嵌套多层，VMWare只能两层。



可见：

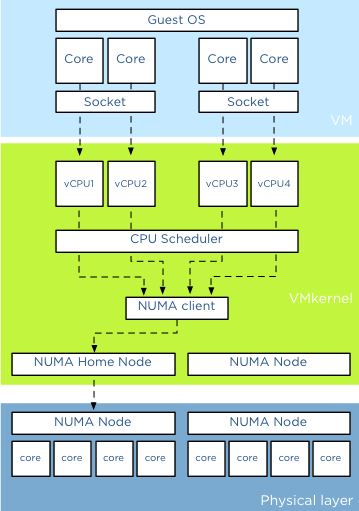
（1）qemu-kvm 通过对 /dev/kvm 的 一系列 ICOTL 命令控制虚机，比如

（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 虚机包括虚拟内存、虚拟CPU和虚机 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。

（4）KVM户机系统的内存是 qumu-kvm 进程的地址空间的一部分。

（5）KVM 虚机的 vCPU 作为 线程运行在 qemu-kvm 进程的上下文中



### 2.2 内存虚拟化

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。

KSM技术

宿主几的内存压缩主要采用KSM（Kernel SamPage Merging）技术，centos6,7默认是打开的，主要的两个服务是KSM服务， ksmtuned服务。

在生产环境中一般要慎用KSM技术，因为打开KSM就为了内存起用，这在业务压力比较大，虚拟机内在变化的时候，将导致两结果：

1. 会消耗一定的计算机资源用于内存扫描，加重CPU的消耗
2. 因为内存起用，当内存不够的时候，会使用swap导致性能下降。

### 2.3 网络虚拟化

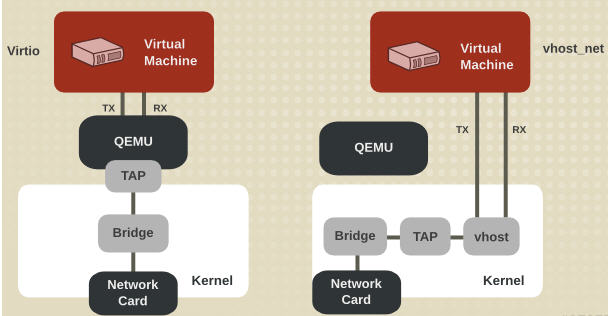
一个完整的数据包从虚拟机到物理机的路径是： 虚拟机 》》 QEMU 虚拟网卡 》》 虚拟化层 》》 内核网桥 》》 物理网卡，优化方案就是让虚拟机访问物理网卡的层数减少，以达到和物理机一样的网络性能。

解决全虚拟化网上效率低下的一个办法就是使用半虚拟化驱动程序Virtio,Virtio可以让虚拟机直接和虚拟化层通信，从而大大提高虚拟机的性能。

#### 2.3.1 vhost-net

运行一台虚拟机是由用户空间的QEMU和内核KVM共同完成的，QEMU负责模拟各种设备提供给虚拟机，KVM负责完成CPU的内存的虚拟。virtio 在宿主机中的后端处理程序（backend）一般是由用户空间的QEMU提供的，然而如果对于网络 I/O 请求的后端处理能够在在内核空间来完成，则效率会更高，会提高网络吞吐量和减少网络延迟。在比较新的内核中有一个叫做 “vhost-net” 的驱动模块，它是作为一个内核级别的后端处理程序，将virtio-net的后端处理任务放到内核空间中执行，减少内核空间到用户空间的切换，从而提高效率。

根据 KVM 官网的这篇文章，vhost-net 能提供更低的延迟（latency）（比 e1000 虚拟网卡低 10%），和更高的吞吐量（throughput）（8倍于普通 virtio，大概 7~8 Gigabits/sec )。



qemu-kvm 命令的 -net tap 有几个选项和 vhost-net 相关的: -net tap,[,vnet\_hdr=on|off][,vhost=on|off][,vhostfd=h][,vhostforce=on|off]

* vnet\_hdr =on|off：设置是否打开TAP设备的“IFF\_VNET\_HDR”标识。“vnet\_hdr=off”表示关闭这个标识；“vnet\_hdr=on”则强制开启这个标识，如果没有这个标识的支持，则会触发错误。IFF\_VNET\_HDR是tun/tap的一个标识，打开它则允许发送或接受大数据包时仅仅做部分的校验和检查。打开这个标识，可以提高virtio\_net驱动的吞吐量。
* vhost=on|off：设置是否开启vhost-net这个内核空间的后端处理驱动，它只对使用MIS-X中断方式的virtio客户机有效。
* vhostforce=on|off：设置是否强制使用 vhost 作为非MSI-X中断方式的Virtio客户机的后端处理程序。
* vhostfs=h：设置为去连接一个已经打开的vhost网络设备

查看vhost\_net 已经被加载

[root@cobbler /]# modprobe vhost-net

[root@cobbler /]# lsmod |grep vhost

vhost\_net 18152 1

vhost 33338 1 vhost\_net

macvtap 22363 1 vhost\_net

tun 27141 4 vhost\_net

启动一个虚拟机，在客户机中使用net定义一个virtio-net 网卡，在主机端使用-netdev启动vhost

kvm –smp 2 –m 1024 –drive file=/data/centos7.qcow2,if=ide,media=disk,format=qcow2 –boot c –name centos7 –net nic,model=virtio –netdev tap,vhost=no,id=guest0

<interface type='bridge'>

<mac address='52:54:00:f3:a7:00'/>

<source bridge='br0'/>

<model type='virtio'/>

<driver name='vhost'/>

<address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>

</interface>

#### 2.3.2 MacVtap

是跳过内核的网桥，使用vhost\_net,必须使用Virtio半虚拟化网上。

#### 2.3.3 PCI Pass-through

这种方式，允许将宿主机的物理PCI设备直接分配给客户机使用，但这需要硬件平台 Intel VT-d 或者AMD IOMMU的支持，几乎所有的PCI和PCI-E设备都支持直接分配，除了显卡以外。

网卡直接分配，需要宿主机的网卡数量有要求，PCI-Passthrough独占1－2块网卡，其它的共享剩余的网卡。

硬盘直接分配：一般SATA，SAS等硬盘的控制器是直接接入到PCI、PCI-E总线的，所以也可以将硬盘的PCI设备直接分配到客户机，需要注意的是，当分配硬盘时，实际上将其控制器作为一个整体分配到客户机中，因此需要在硬件平台上至少有两个或者多个SATA/SAS控制器

准备工作：

1. 在bios中打开intel VT-d
2. 在linux内核中启用PCI Pass-through

添加 intel\_iommu=no 到/boot/grub/grub.conf

使用lspci –nn命令找到待分配的PCI设备，

使用virsh nodedev-list命令找到该设备的PCI编号

将设备从主机解除 virsh nodedev-dettach pci\_000\_lf\_00\_0

然后在虚拟机中增加该设备。kvm –smp 2 –m 2048 –drive file=centos7.qcow2,if=ide,media=disk,format=qcow2 –boot –c –name centos7 –device pci-assign,host=lf:00.0

#### SR-IOV

Single Root IO Virtualization sharing

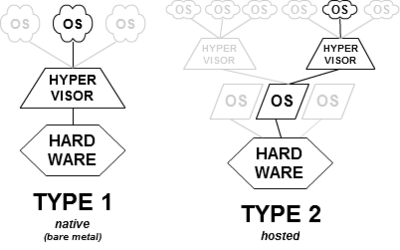
### 磁盘虚拟化

|  |  |
| --- | --- |
| 类型 | 选项 |
| 磁盘设备（软盘、硬盘、CDROM等） | **-drive** option[,option[,option[,...]]]：定义一个硬盘设备；可用子选项有很多。  file=/path/to/somefile：硬件映像文件路径；  if=interface：指定硬盘设备所连接的接口类型，即控制器类型，如ide、scsi、sd、mtd、floppy、pflash及virtio等；  index=index：设定同一种控制器类型中不同设备的索引号，即标识号；  media=media：定义介质类型为硬盘(disk)还是光盘(cdrom)； format=format：指定映像文件的格式，具体格式可参见qemu-img命令；  -**boot** [order=drives][,once=drives][,menu=on|off]：定义启动设备的引导次序，每种设备使用一个字符表示；不同的架构所支持的设备及其表示字符不尽相同，在x86 PC架构上，a、b表示软驱、c表示第一块硬盘，d表示第一个光驱设备，n-p表示网络适配器；默认为硬盘设备(-boot order=dc,once=d) |
| 网络 | **-net nic**[,vlan=n][,macaddr=mac][,model=type][,name=name][,addr=addr][,vectors=v]：创建一个新的网卡设备并连接至vlan n中；PC架构上默认的NIC为e1000，macaddr用于为其指定MAC地址，name用于指定一个在监控时显示的网上设备名称；emu可以模拟多个类型的网卡设备；可以使用“qemu-**kvm** -net nic,model=?”来获取当前平台支持的类型；  **-net tap**[,vlan=n][,name=name][,fd=h][,ifname=name][,script=file][,downscript=dfile]：通过物理机的TAP网络接口连接至vlan n中，使用script=file指定的脚本(默认为/etc/qemu-ifup)来配置当前网络接口，并使用downscript=file指定的脚本(默认为/etc/qemu-ifdown)来撤消接口配置；使用script=no和downscript=no可分别用来禁止执行脚本；  **-net user**[,option][,option][,...]：在用户模式配置网络栈，其不依赖于管理权限；有效选项有：      vlan=n：连接至vlan n，默认n=0；      name=name：指定接口的显示名称，常用于监控模式中；      net=addr[/mask]：设定GuestOS可见的IP网络，掩码可选，默认为10.0.2.0/8；      host=addr：指定GuestOS中看到的物理机的IP地址，默认为指定网络中的第二个，即x.x.x.2；      dhcpstart=addr：指定DHCP服务地址池中16个地址的起始IP，默认为第16个至第31个，即x.x.x.16-x.x.x.31；      dns=addr：指定GuestOS可见的dns服务器地址；默认为GuestOS网络中的第三个地址，即x.x.x.3；      tftp=dir：激活内置的tftp服务器，并使用指定的dir作为tftp服务器的默认根目录；      bootfile=file：BOOTP文件名称，用于实现网络引导GuestOS；如：qemu -hda linux.img -boot n -net user,tftp=/tftpserver/pub,bootfile=/pxelinux.0 |

## KVM 架构

KVM （Kernel-based Virtual Machine）它是一个linux的一个内核模块，该内核模块使得Linux 变成了一个Hypervisor

Hypervisor: 又称 Virtual machine monitor (VMM) 是用来建立与执行虚拟机器的部分电脑软件，固件或者硬件



KVM是基于虚拟化扩展（Intel VT, AMD-V）的X86硬件开源的硬件辅助全虚拟化方案。虚拟机被实现为常规的Linux进程，由标准Linux调用程序进行调试，虚拟的每个虚拟CPU被实现为常规的Linux进程，这使得KVM能够使用Linux内核的已有功能。

### QEMU 和 KVM

QEMU其实际就是一台硬件模拟器，可以模拟许多硬件，包括CPU虚拟化、内存虚拟化、以及网卡，显卡，存储控制器，硬盘等，因为是纯软件模拟，所以其性能低下，KVM 在QEMU的基础上做了修改，运行在用户空间，提供硬件I/O 虚拟化，通过IOCTL /dev/kvm设备和KVM交互。 可以理解为KVM是用来为QEMU硬件虚拟化加速，KVM的虚拟机借助QEMU的管理工具来管理

yum install qemu-kvm qemu-img virt-manager libvirt libvirt-python libvirt-client virt-install virt-viewer bridge-utils qemu-kvm-tools

### libvirt 与kvm

libvirt是一套开源的虚拟化的管理工具，主要包括一套API的Lib库支持c,python,ruby；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（KVM，XEN，VMware ESX，HyperV）

libvirt管理主要分为：

1. 存储池资源管理，支持本地文件系统目录，裸设备，LVM等，在虚拟机上支持qcow2, vmdk, raw
2. 网络资源管理，支持linux桥，vlan，多网卡绑定，ipen vSwitch。

创建一个磁盘，

[root@pre /opt]# qemu-img create -f raw /opt/centos-7.1.raw 10G Formatting '/opt/centos-7.1.raw', fmt=raw size=10737418240

创建虚拟机。

[root@pre /opt]# virt-install --name Centos-7.1-x86\_64 --virt-type kvm --memory 1024 --vcpus 1 --cdrom /opt/Centos.7.2\_x86.64.iso --disk path=/opt/centos-7.1.raw --network network=default --graphics vnc,listen=0.0.0.0 --noautoconsole

Starting install...

Creating domain... | 0 B 00:00:00

Domain installation still in progress. You can reconnect to

the console to complete the installation process.

修改配置

virsh edit Centos-7.1-x86\_64

virt-install --name=Ubuntu-16-04 --file=/var/lib/libvirt/images/ubuntu16-04.dsk --file-size=20 --nonsparse --graphics spice --vcpus=2 --ram=2048 --cdrom=ubuntu-16.04-server-amd64.iso --network bridge=br0 --os-type=linux --os-variant=generic

virtsh

--config 写到配置中，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，改变当前状态下的配置。