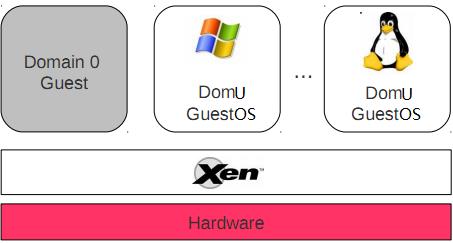
目录：  
   1. Xen的简介  
   1.1  Xen的大体结构  
   1.2 Xen对VM的称呼  
   1.3 Xen对CPU和内存的虚拟化过程  
   1.4 Xen对IO设备的虚拟化过程  
   1.5 Linux Kernel对Xen的支持  
   1.6 Xen版本发布简史  
   1.7 Xen的工具栈  
   1.8 XenStore  
   1.9  虚拟化中的四种网络模型  
   1.10 Xen的安全问题导读  
   2. Xen的安装及配置文件说明  
   2.1.1  在CentOS6.6上运行Xen的条件  
   2.1.2  Xen的配置  
   2.2.1  Xen 启动DomU的配置文件说明  
   2.2.1.1  如何创建一个Xen PV模式的VM 【注:HVM模式的VM创建参见试验部分】  
   3. 使用libvirt实现Xen虚拟机的图形管理  
   4. PV DomU的根文件系统可以以多种不同的方式安置

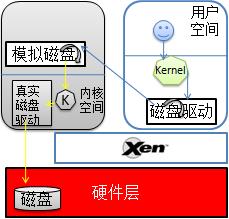
1. Xen的简介  
         Xen是一个开源的可直接运行于硬件层之上的虚拟化软件,它可在传统虚拟技术极度不友好的X86架构上也有上佳的表现  
  它是英国剑桥大学开发的开源虚拟化软件,它的初衷是在一台物理机上运行上百台虚拟机;  
         Xen的设计十分精巧,它属于虚拟化type-I ,因为Xen实际是一个简化版的Hypervisor层;相对于Type-II类型的基于Host的  
   虚拟化(如:VMware Workstattion),其性能相对会较好;Xen仅对CPU和Memory直接接管，而其它IO硬件驱动则由其上运行的  
   第一个虚拟机来提供支持.这样做的原因是: Xen无法为众多IO设备开发驱动,而硬件设备的开发商也不会专为Xen提供驱动,  
   因此Xen采用了这样一种特别的设计方式。

         Xen默认认为自己是直接运行于硬件层之上的虚拟化软件,并且可以直接驱动CPU和内存,需注意CPU和内存是所有想要运行  
  的操作系统必须能直接支持的,但Xen为保证自身的小巧,它并没有提供虚拟机的管理接口,因此它采用了一种独特的方式,先运行一台特权虚拟机,且这台VM必须支持Kernel的修改,因此选择开源的Linux做为特权VM是最合适的,这样也可方便采用Linux所支持的方式来开发虚拟机管理接口,实现与Xen Hypervisor层直接交互来完成为VM分配CPU和内存资源 及 创建、删除、停止、启动VM的管理接口;通常这台特权虚拟机一定会采用当前比较流行的Linux发行版,因为它能支持更多IO硬件设备,如:网卡,磁盘,显卡,声卡等; 到目前为止，NetBSD, GNU/Linux, FreeBSD和Plan 9，OpenSolaris等系统已经支持半虚拟化方式运行在Xen的DomU中；并且目前Xen已经支持x86，x86\_64和ARM等平台，并正在向IA64、PPC移植。移植到其他平台从技术上是可行的，未来有可能会实现。  
    
         Xen虚拟机支持在不停止的情况下在多个物理主机之间实时迁移。在操作过程中，虚拟机在没有停止工作的情况下内存被反复的复制到目标机器。虚拟机在最终目的地开始执行之前，会有一次60-300毫秒的非常短暂的暂停以执行最终的同步化，给人无缝迁移的感觉。类似的技术被用来暂停一台正在运行的虚拟机到磁盘，并切换到另外一台，第一台虚拟机在以后可以恢复。  
              
1.1  Xen的大体结构:  
    Xen的组成分为三部分:  
           (1) 硬件层之上的Xen Hypervisor层：负责直接驱动CPU和Memory这些基础硬件,  
                             为其它所有虚拟机提供CPU、内存、Interrupt(中断)管理,并且还提供了HyperCall的调用。  
           (2) 第一个虚拟机: 此虚拟机在Xen中称为特权虚拟机: 它有整个虚拟化环境的访问权,并负责创建用户级虚拟机,  
                             并为其分配I/O设备资源.它的Kernel是经过特别修改的VM,它可以直接访问IO硬件也可访问其它用户VM。  
           (3) 其它众多虚拟机: 这些虚拟就是用户级虚拟机: 它们是实际提供给用户使用的虚拟机,也是相关隔离的VM。  
                       需要注意：Xen支持三种虚拟化,当然此处的虚拟化特指CPU的半虚拟化或完成虚拟化.  
                               <1> Para-Virtualization(半虚拟化): 在这种虚拟化中,CPU不要求支持HVM特性,  
                                               但GuestOS的Kernel必须允许被修改.否则将无法支持该GuestOS运行在DomU中。  
                                               这是因为必须让GuestOS知道自己是运行在虚拟化环境中,这样它在进行特权指令操作硬件时，  
                                               会直接发起HyperCall向Xen Hypervisor发起请求来完成所谓的硬件操作。  
                                               在PV技术下，能够运行在DomU上的OS包括：  
                                                   Linux, NetBSD, FreeBSD, OpenSolaris  
                                                     
                               <2> HVM(基于硬件的完全虚拟化): 此种虚拟化需要借助于Intel的VT-x 或 AMD的AMD-v 的HVM技术  
                                               及Qemu的IO硬件模拟，才能支持GuestOS的kernel不修改,就可直接被DomU支持。  
                                               这是因为Xen Hypervisor是运行在CPU的环-1上的，GuestOS的kernel是运行在CPU的环0上,  
                                               GuestOS向环0上发起的所有假特权指令调用都由CPU直接捕获,并交由环-1上的  
                                               Xen Hypervisor处理,最后由Xen代其执行  
                                               这样DomU若发起关机指令时,Xen仅会切断该GuestOS的电源,而不会影响其它GuestOS。  
                                               在HVM技术下，能够运行在DomU上的OS包括： 所有支持X86架构的OS.

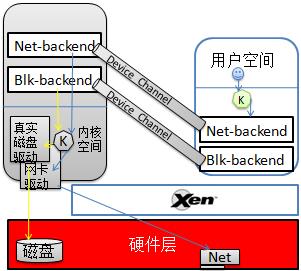
                       <3> PV on HVM: I/O设备半虚拟化运行，CPU运行于HVM模式  
                           此中方式是为了解决HVM方式中IO设备也必须完全模拟而带来的性能低下问题；通过让CPU进行  
                           完全虚拟化，而I/O设备则采用在GuestOS中安装相应的IO驱动实现IO的半虚拟化的方式来提高效率。  
                           在PV on HVM的技术下，能够运行在DomU上的OS包括：  
                                 只要OS能驱动PV接口类型的IO设备,即可.  
1.2 Xen对VM的称呼:  
       Xen对VM统称为Domain.  
       第一台特权VM,在Xen中通常称为: Domain0,简称为Dom0, 别名: Privileged Domain.  
       其它后续用户级VM,在Xen中称为: Domain1,Domain2,…., 它们有一个统称为DomU,别名:Unprivileged Domain.  
       

1.3 Xen对CPU和内存的虚拟化过程【注: 关于虚拟化中CPU和内存的虚拟化在附件加入了一些补充.】：  
       Xen在给VM提供CPU的虚拟化时,它采用的也是在Xen hypervisor层启动一个线程,并将这些线程映射到某个物理核心上,  
  当然通过DomU的配置文件中的cpus可以指定将这些模拟CPU的线程绑定到某几个物理核心上；而内存的虚拟化  
  则是内存页的映射,将物理内存上多个连续或不连续的内存页映射给VM,让VM看来这就是一个完整的连续的内存空间.

1.4 Xen对IO设备的虚拟化过程：  
       当启动一个用户VM(DomU)时, 该VM所需的CPU和内存都有Xen Hypervisor提供,而它若需要使用IO设备时,则向特权VM  
  发起请求,特权VM(Dom0)会为该用户VM创建一个模拟的硬件设备线程,并运行于特权VM的用户空间,当用户VM向该IO硬件  
  发起调用时,特权VM上相应的模拟设备接收请求并将其转化为特权VM对IO硬件的操作,交给特权VM的内核来代为  
  完成其操作。这里需注意这些虚拟IO硬件需要由Qemu来模拟,Xen本身并没有提供相应的模拟功能。  
  (注：特权VM的CPU和内存也是有Xen Hypervisor提供.)  
    
       Qemu模拟IO设备(完全虚拟化方式): 假如用户VM向特权VM请求磁盘,特权VM可以将一个分区、文件等,  
通过Qemu将其模拟成一个磁盘设备,就拿文件来说,特权VM先创建一个映像文件,再通过Qemu为该文件模拟一个磁盘控制器芯片,然后,将其映射到用户VM上,当然模拟的这个磁盘控制器芯片一定是一个最常见的,用户VM的Kernel一定支持的,但需注意: 模拟的磁盘可能会与实际的物理磁盘不同,因为要尽可能兼容。这样一来用户VM假如要写数据到磁盘的过程如下：  
       用户VM-APP—>用户VM-Kernel调用虚拟磁盘的驱动进行写数据前的准备  
                               (如:数据写入到磁盘中的扇区位置/数据编码等)—>  
                               用户VM-Kernel将编码后的信息发给特权VM的模拟磁盘进程—>  
                               特权VM的模拟磁盘进程再将编号信息还原后发给特权VM-kernel—>  
                               特权VM-kernel调用真实物理磁盘的驱动对数据进行写前准备—>最后磁盘驱动调度磁盘完成写入.  
      摘录补充:（<http://my.oschina.net/davehe/blog/94039?fromerr=mOuCyx6W>）  
           Xen向Domain提供了一个抽象层，其中包含了管理和虚拟硬件的API。Domain 0内部包含了真实的设备驱动（原生设备驱动），可直接访问物理硬件，Xen 提供的管理 API 可与其交互，并通过用户模式下的管理工具(如：xm/xend、xl等)来管理 Xen 的虚拟机环境。



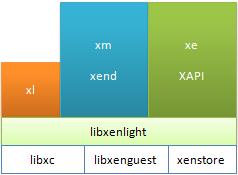
       半虚拟化的IO设备：它与模拟最大不同是DomU知道自己是运行在虚拟化环境中的,并且知道这个磁盘不是真正的磁盘  
  它只是Xen模拟的一个磁盘前端驱动(Disk Frontend),它要写数据时，直接将数据交给Disk Frontend,而不再去调用磁盘  
  驱动进行数据编码,当特权VM端的Disk backend收到来自DomU的数据时,也是直接转给特权VM-Kernel,由其直接调用  
  物理磁盘驱动来对这些原始数据进行处理并写入磁盘。  
   摘录补充：  
           Xen2.0之后，引入了分离设备驱动模式。该模式在每个用户域中建立前端（front end）设备，在特权域(Dom0)中建立后端(back end)设备。所有的用户域操作系统像使用普通设备一样向前端设备发送请求，而前端设备通过IO请求描述符(IO descripror ring)和设备通道（device channel）将这些请求以及用户域的身份信息发送到处于特权域中的后端设备。这种体系将控制信息传递和数据传递分开处理。  
           半虚拟化客户机（Domain U PV）的PV Block Driver接收到要向本地磁盘写入数据的请求，然后通过Xen Hypervisor将自己与Domain 0共享的本地内存中的数据写入到本地磁盘中。在Domain 0 和半虚拟化Domain U之间存在事件通道(我的理解:Device Channel包含Event Channel)，这个通道允许它们之间通过存在于Xen Hypervisor内的异步中断来进行通信。Domain 0将会接收到一个来自于Xen Hypervisor的系统中断，并触发Domain 0中的Block Backend驱动程序去访问本地系统内容，并从自己与半虚拟化客户机的共享内存中读取适合的数据块后，随即被写入到本地磁盘的指定位置中。



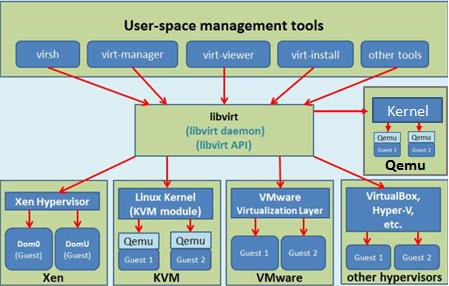
        但无论采用模拟或半虚拟化最终都是对物理磁盘的操作,假如当前只有一个物理磁盘,众多用户VM都在进行大量的  
  读写请求,此时,为了避免用户VM无限制的向特权VM发起请求,特权VM中采用一个环状缓存区,每到一个IO请求,就先将其  
  塞入这个环状缓冲区的槽位中,若缓冲区满了,就会告诉用户VM IO设备繁忙。当然其它各种IO设备大致都采用这种机制  
  来控制。  
   
1.5  Linux Kernel对Xen的支持:  
           》Linux2.6.37  : kernel开始对Xen进行支持,并加其加入到Kernel中。  
           》Linux3.0 ：Kernel开始对Xen的关键部分进行优化.  
     RHEL对Xen的支持概况:  
       Redhat系列对Xen的支持情况:  
           RHEL5.7 ~ 及以前版本: 默认的企业虚拟化技术为Xen.  
                   但Redhat提供了两种内核:  
                       kernel-…      :这是仅允许RHEL系统的内核，不能运行在DomU中。  
                       kernel-xen..  :这是需要部署XenServer时,使用的Kernel版本.  
           RHEL6 ~ 及以后版本: 默认支持KVM(收购自以色列的一款虚拟化工具),并且不在对Xen做任何支持，但允许自己运行在DomU中.

1.6  Xen版本发布简史:  
  10年4月Xen4.0.0发布,改进后Xen的DomU最大可支持虚拟CPU 64颗,Xen主机可支持1TB内存和128颗物理CPU,磁盘可支持快照和克隆; HVM客户机支持虚拟内存页共享;  
   11年4月发布的Xen4.1版后,xm/xend开始被提示废弃,xl这个更轻量级的Xen VM管理工具逐渐成为主流.  
   15年为止已经发布Xen4.5版本.目前yum源可用的最新版Xen是4.6.1版的(<http://mirrors.skyshe.cn/centos/6.7/virt/x86_64/xen-46/).>

1.7  Xen的工具栈：  
       xend : 这是Xen Hypervisor的Dom0上运行的服务,此服务用来监控xm命令发来的指令,并完成相应的动作。  
       xm : Xen Management,用来管理VM的创建、删除、启动、快照、删除、停止等的管理工具。  
       xl : 这是一个基于libxenlight库的一个轻量级VM管理工具，它从Xen4.1开始出现,从4.3以后,它被作为主要的VM管理  
           工具,而xm这个重量级管理工具开始被提示废弃.  
       以下为xm、xl的对比图:

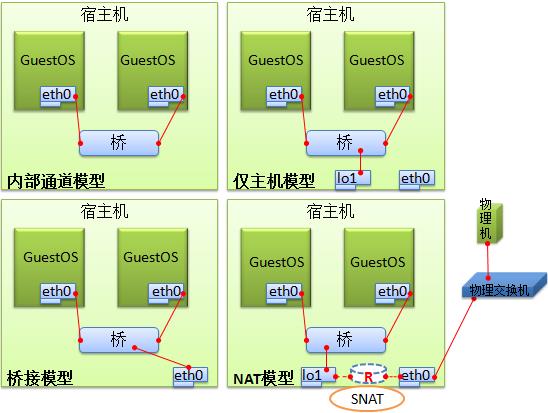


        xl 和 xm都需要调用libxenlight，但xl不需要运行任何服务,它可直接调用libxenlight完成相关操作。  
      xe/XAPI，是xend的一个API管理接口,通常用于Xen Cloud环境中：Xen Server, XCP  
        
     virsh/ libvirt : 这是Redhat发起开发的一套用于管理众多不同类别的VM的管理工具。  
                       virsh : 这是一个命令行工具  
                       libvirt: 则是一个lib库, libvirtd守护进程用于监听virsh命令操作,并调用lbvirt完成相关操作.



1.8  XenStore:  
       为各Domain之间提供共享信息存储的空间,同时它也是一个有着层级结构的名称空间数据库；  
   它运行于Dom0上,由Dom0直接管理,它支持事务和原子操作。XenStore通常用来控制DomU中设备的机制,  
   并通过多种方式对其进行访问。  
   摘录补充(<http://blog.chinaunix.net/uid-26299858-id-3134495.html>)：  
         XenStore是Xen提供的一个域间共享的存储系统，它以字符串形式存放了管理程序和前、后端驱动程序的配置信息。  
   Dom0管理所有的数据，而DomU通过共享内存，向Dom0请求与自己相关的键值，以此实现域间通信。Xen提供了多种  
   接口用来操作XenStore：命令行的xenstore-\*命令、用户空间的xs\_系列函数、内核的XenBus接口，都可以用来方便地  
   操作XenStore的数据。

1.9  虚拟化中的四种网络模型  
       在虚拟化环境中虚拟网络是十分重要但又比较难,需要特别注意;  
       在Linux中实现虚拟网络的方法中比较常用的工具有两个：bridge-utils 和 openvswitch，它们创建的虚拟网络设备是  
       不能相互使用的，比如：bridge-utils创建的桥设备，openvswitch是无法识别的。  
       用下图来做简单说明



        注：lo1: 为loopback接口,但实际测试时，发现loopback接口无法桥接到虚拟网桥上。

1.10  Xen的安全问题导读  
      补充见附件：

2. Xen的安装及配置文件说明

2.1.1  在CentOS6.6上运行Xen的条件:  
       方式一:  
           (1) 编译3.0以上版本的内核,启动对Dom0的支持.  
           (2) 编译xen程序  
       方式二：  
           使用相关Xen运行环境快速部署的项目：  
           (1) xen4contos : 这是Xen官方提供的开源项目。  
                 xen环境部署的RPM包镜像站: <http://mirrors.aliyun.com/centos/6.7/xen4/x86_64/>      
           (2) xen made easy  
                       
2.1.2 Xen的配置：  
           (1) 修改grub.conf  
                # Xen是直接运行于硬件层之上的,因此必须修改grub.conf,手动添加以下参数:  
                title  Xen Server Linux 3.7.4  
                       root (hd0,0)  
                       **kernel  /xen.gz  dom0\_mem=1024M  cpufreq=xen  dom0\_max\_vcpus=2  dom0\_vcpus\_pin**  
                       **module**  /vmlinuz-3.7.4-1.el6xen.x86\_64 ro  root=/dev/mapper/vg0-root  rd\_NO\_LUNKS.UTF-8    
                                   rd\_LVM\_LV=vg0/swap  rd\_NO\_MDSYSFONT=latarcyrheb-sun16  crashkernel=auto  rd\_NO\_DM  
                                    KEYBOARDTYPE=pc  KEYTABLE=us  rd\_LVM\_LV=vg0/root   rhgb  quiet  
                       **module**  /initramfs-3.7.4-1.el6xen.x86\_64.img  
                         
               注: kernel 必须指定xen\*.gz为启动的内核, dom0\_mem:设定Dom0启动后可以使用的内存大小,  
                     cpufreq: 设定由Xen来管理CPU,  dom0\_max\_vcpus: 设定Dom0可以使用多少颗CPU,  
                     dom0\_vcpus\_pin: 将Dom0固定在系统启动后,分配给它的CPU上,以避免它去抢占其它物理CPU核心,  
                                               这样其它物理核心就可以分配给DomU来使用了。  
                     详细参数查看:  
                           <http://xenbits.xen.org/docs/unstable/misc/xen-command-line.html>  
                             
2.2.1 Xen 启动DomU的配置文件说明  
       xl  list :      #首先需要了解的第一个命令.  
           xen VM的常见状态:  
               r : running  
               b: block(阻塞)  
               p: pause(暂停)： 类似与睡眠.  
               s: stop  
               c: crash(崩溃)  
               d: dying， 正在关闭的过程中.  
     
2.2.1.1  如何创建一个Xen PV模式的VM:  
           1. Kernel 和 initrd或initramfs ：这些LinuxKernel文件必须要有,但可以不在DomU上,它们可以在Dom0上.  
           2. DomU内核模块(即:/lib/modules/`uname -r`): 这些就需要在DomU根文件系统中存储了。  
           3. 根文件系统  
           4. swap设备: 若条件充足也可以提供.  
             
           以上四步的内容必须定义在DomU的配置文件中.  
           注: xl 和 xm启动DomU的配置文件是存在一些不同的.  
             
           对于xl命令创建VM所需的配置文件说明可查看:  
                   man  xl.cfg  
                   # 注: man  xl.conf    #这是对xl命令所使用的配置文件.  
                     
           xl.cfg的配置文件参数说明：  
                   name : 域的唯一名.  
                   builder: 指明VM的类型,generic：创建PV类型的VM;  HVM：创建HVM类型的VM  
                   vcpus： 指定CPU个数.  
                           maxvcpus：指定最大可使用CPU的个数,这些CPU可在需要是手动启动。  
                           cpus: 指定VM的vcpu线程可以运行在哪些物理核心列表上.  
                                   #如：cpus="0-3,5,^1"   这表示:VCPU可运行在0,2,3,5上,但不能运行在1上.  
                                   #建议将vCPU绑定到固定的物理核心上,这样可减少vCPU线程在多个物理核心上切换.  
                   memory: 指定DomU启动时预分配的内存大小[单位:M]  
                           maxmem: 指定最大给DomU分配的内存大小. [单位:M]  
                   on\_poweroff: 指定DomU关机时,实际采用的动作.  
                           destroy: 直接将DomU断电关机.  
                           restart: 重启  
                           rename-restart: 改名后重启  
                           preserve: 将这个DomU保存,以便下次再启动。  
                           coredump-destroy: 将DomU的运行中的内核数据备份出来后,再关机。  
                           coredump-restart: 先备份内核数据，再重启.  
                   on\_reboot: 重启DomU时实际采用的动作。  
                   on\_crash: 当DomU崩溃后，实际采用的动作。  
                   uuid: DomU的UUID,非必须.  
                   disk:指定DomU的磁盘列表  
                           如: disk=[ "/img/disk/DomU1.img" , "/dev/sda3" , …]  
                   vif : 指定DomU的网卡列表  
                           如: vif=[ "NET\_SPEC\_STRING" , "NET\_SPEC\_STRING" , …]  
                   vfb: 指定DomU的显示器, vfb:virtual frame buffer(虚拟帧缓冲)  
                           如: vfb=[ "VFB\_SPEC\_STRING" , "VFB\_SPEC\_STRING" ,…]  
                   pci :指定DomU的PCI设备列表.  
                           如:pci=[ "PCI\_SPEC\_STRING" , "PCI\_SPEC\_STRING" ,…]  
           PV模型的专用指令:  
                   kernel : 指定内核文件路径,此路径为Dom0的路径.  
                   ramdisk: 指定initrd或initramfs文件的路径.  
                   root: 指定根文件系统. 此参数仅与kernel和ramdisk一起使用,因为,kernel和ramdisk都是在Dom0上的,  
                           并没有grub.conf来告诉kernel根文件系统在哪里，因此这里需要指定。  
                   extra: 与root参数类似,它是指定kernel启动时的其它参数的.  
                # 以上4个参数用于kernel文件在Dom0上, 下面固定格式用于DomU有自身的Kernel文件.  
                   bootloader="pygrub": 若DomU使用自己的kernel和ramdisk，则此时需要一个Dom0中的应用程序来  
                            实现其bootloader功能; 这个不用指定root,因为,DomU的启动所需的所有文件都在自己  
                            的镜像文件中,它可以从grub.conf中指定根文件系统的位置.

                注：  
                     # 让DomU通过网络之间安装操作系统,需要注意:  
                     #  kernel 和 ramdisk所需要的文件必须是：安装光盘目录下/isolinux/{vmlinuz,initrd.img}  
                       kernel="/images/kernel/vmlinuz"  
                       ramdisk="/images/kernel/initrd.img"                         
                       extra="ks=http://1.1.1.1/centos6.6\_x86\_64.ks"       #注:给内核传递的ks文件。  
                另注:【注: 没有亲自测试,若有偏差还望指正.】  
                        cloud-init\*.rpm工具包可以将安装的OS中的特有信息(如:MAC, 磁盘信息等)删除,并且可以在下次启动时，  
                 自动生成这些信息.是制作云模板OS镜像的工具。  
     
           磁盘参数的指定方式：  
                   xl方式创建VM时,磁盘指定格式: http://xenbits.xen.org/docs/unstable/misc/xl-disk-configuration.txt  
                   [<target>, [<format>,[<vdev>,[<access>]]]]  
                   target:  表示磁盘映像文件或设备文件路径:  
                             如: /images/xen/linux.img  
                                  /dev/vg-xen/lv-linux  
                   format: 表示磁盘的格式:  
                             如: raw、qcow2..等,具体格式可查看: qemu-img  –help |grep 'Supported formats'  
                   vdev: 指定添加的虚拟磁盘被DomU识别为何种类型的磁盘; 支持:hd, sd, xvd(xen-vritual-disk)  
                           注: 指定是需要写成: sda 或 sdb等,即要指定这是第几块磁盘.  
                   access: 访问权限,除CDROM为'r'只读外，其它都为'rw'  
                     
                   示例:  
                        disk=["/images/xen/linux.img,qcow2,xvda,rw",  "/iso/CentOS6.7.iso,,hdc,devtype=cdrom"]  
                       # 使用Dom0中物理磁盘分区做为DomU的存储空间  
                        disk=['/dev/vg-data/lv-bbox,raw,xvda,rw']  
                    
                   创建虚拟磁盘文件:  
                       #注qemu-img创建的磁盘映像文件是采用稀疏磁盘格式创建的.因此用:du -sh  linux.img 可看到其大小为0.  
                       #【qemu-img 的子命令简介详见附件】  
                       qemu-img  create  -f  raw  -o  size=2G   /images/xen/linux.img        
                       或  qemu-img  create  -f  raw  /images/xen/linux.img  2G  
                       若想知道当前指定的磁盘格式支持哪些额外参数:  
                               qemu-img  create  -f  qcow2  -o  ?   /images/xen/linux.img     
                                       size : 指定qcow2格式的虚拟磁盘大小。  
                                       backing\_file: 指定其后端存储快照信息的映像文件位置.  
                                       backing\_fmt: 指定后端映像文件的格式。  
                                       encryption: 是否对创建的映像文件加密  
                                       cluster\_size: 指定qcow2格式的簇大小.  
                                       preallocation: 指定创建qcow2格式的磁盘映像文件时,需要预先做些什么:  
                                                   > off : 什么也不做,创建一个空磁盘.  
                                                   > metadata: 预先创建qcow2的元数据信息,建议设置.  
                                                   > full: 直接填充成指定大小的磁盘文件,它包含了metadata.  
                      
                   创建虚拟网络接口:  
                       #虚拟网卡的创建直接在配置文件中使用vif指定即可。  
                       #格式: vif=[ "NET\_SPEC\_STRING" , "NET\_SPEC\_STRING" , …]  
                       NET\_SPEC\_STRING:的格式如下：  
                               key=value  
                               key包含以下几个:  
                                   》mac :指定网卡的MAC地址,注:MAC地址必须以:00:16:3e 开头,这是IANA分配给Xen的MAC厂商前缀.  
                                   》bridge: 指定此网卡要桥接到Dom0上的那个桥设备上.  
                                   》model: 指定模拟网卡的芯片类型:[rtl8139 |e1000]  
                                   》vifname:指定在Dom0中显示的接口名, 注: 在DomU中显示还是eth0…  
                                   》script: DomU在创建网络接口时,预先执行的脚本.注: 此脚本的路径也是Dom0上的路径.  
                                   》ip：指定要注入DomU中的IP地址。  
                                   》rate: 指定网卡的设备速率.  
                                           如: rate=10Mb/s   
                                                 rate=1MB/s@20ms      #20毫秒内只能传输1M/0.02s=20000字节/20ms  
        
                  图形窗口的启用:  
                       可直接修改DomU的启动配置文件,并在其中添加:  
                           (1) vfb=['sdl=1']       #这是直接在本地启动一个VNC窗口来输出DomU的图形桌面,且只能本地连接.  
                           (2)Dom0上启动一个VNC进程,并监听在5900端口上,可通过密码连接.  
                                   vfb=['vnc=1,vnclisten=0.0.0.0,vncpasswd=123456,vncunused=1,vncdisplay=:1']  
                                   #注: vncunused: 为非0值,则表示vncserver监听在大于5900的第一个没被占用的端口上.  
                                   # vncdisplay: VNC显示号,默认为当前域的ID,当前域的VNC服务器将监听在5900+此显示号的端口上.

3. 使用libvirt实现Xen虚拟机的图形管理  
       yum install  libvirt  libvirt-deamon-xen  virt-manager  python-virtinst  libvirt-client  
       注： virt-manager: 是图形管理VM的接口界面,类似与VMware，可创建、启动、停止等  
               python-virtinst: 此工具提供了virt-install命令行管理VM的接口.  
               libvirt-client: 提供了一个virsh命令来管理VM。  
                 
       service  libvirtd  start    #要使用libvirt工具集，此服务必须首先启动.  
       virsh  dumpxml  busybox     #将busybox的信息输出为virsh的可用的xml配置文件,可修改此文件做模板来创建新VM实例.

4. PV DomU的根文件系统可以以多种不同的方式安置:  
   (1) 虚拟磁盘映像文件  
       (a)方便制作成通用模板  
               当用qemu-img创建好一个虚拟磁盘映像文件,并向其中安装好OS后,可以使用cloud-init这种工具对其进行  
         修改,去除其中OS的唯一性的数据信息(如:MAC地址等),需要注意的是,磁盘映像文件实际上是由标准格式的并且  
         在其上创建完文件系统后,就可以通过FS的API接口在不挂载的情况下对其中的文件进行修改.  
       (b)方便实现实时迁移  
                1. 将模板映像文件放置于FTP/NFS/Samba等共享存储上,且有多台Xen Hypervisor:  
                     场景一: 需要快速创建一台VM.  
                           假如当前业务中MySQL的从库读压力过大,需要扩容,此时就可以直接通过自行开发的脚本工具来判断  
                       当前那个Xen Hypervisor更空闲,然后,直接下载从共享存储上下载一个模板磁盘映像,直接就可以启动起来.  
                       并且这是配置好的从库模板,启动后它可以直接连接到主库等等.  
                      场景二: 快速迁移  
                           假如当前某台Xen Hypervisor的CPU、Memory或其他资源突然升高,此时,可直接将该Xen Hypervisor上  
                        某些VM的内存数据导出到共享存储上,然后直接在另一台比较宽裕的Xen Hypervisor上，下载模板映像文  
                        件并直接将该共享存储上导出的内存数据与模板映像文件关联,就可恢复到此Xen Hypervisor上实现快速迁移。  
                2.分布式文件系统上存储磁盘映像文件，且有多台Xen Hypervisor:  
                       场景一: 故障快速恢复  
                            假如当前Xen Hypervisor上运行了多台VM,但Xen Hypervisor所在物理机突然故障,这时,我们完全可以直接  
                       将分布式存储系统上的磁盘映像文件直接让正常的Xen Hypervisor接管,已实现快速迁移.  
                       场景二：快速迁移  
                             如上面场景二类似,但此处我们无需在下模板磁盘映像文件,直接dump出运行中VM的内存数据到共享存储上  
                       就可以直接在另一台Xen Hypervisor上启动起来。  
    (2) 使用Dom0上空闲的物理磁盘分区  
    (3) 使用Dom0上空闲的逻辑卷  
    (4) 使用iSCSI设备提供的块级别网络文件系统 或 分布式文件系统。  
    (5) 网络文件系统,如: NFS, Samba

实验目录:  
   1. 环境说明  
   2. Xen的安装和配置  
   3. 未解决的问题  
       1.在DomU的配置中添加sdl=1，启动报错,改为vnc=1,正常.  
       2.配置文件中指定IP,但启动DomU后，IP并没有自动配置上.  
       3.不能动态调整vCPU的个数.  
       4.以上问题若有高手知道，非常愿意交流。  
   4. 创建第一台Xen DomU  
       4.1 创建并启动DomU  
       4.2 基本功能测试  
           1. 测试启动DomU的图形窗口(仅本地可连的VNC(sdl=1) 和 可远程连接的VNC(vnc=1)).  
           2. 测试动态添加和移除磁盘  
           3. 测试动态添加和移除网卡  
           4. 测试挂起DomU(即:将DomU的内存数据导出到文件.)  
           5. 测试恢复DomU.  
           6. 测试动态增加内存和CPU  
           7. 测试创建克隆DomU.  
               1) 类似VMware中的连接克隆  
               2) 直接复制HVM DomU的img映像文件来克隆DomU.  
           8. 测试迁移DomU  
           9. Xen的提供的批量操作DomU的脚本简单分析: xendomains  
           10. xenstore的简单操作说明.  
     
   以上测试因本人对Xen的理解实在有限,不能更深入测试,以下测试仅本人亲自测试所作,若有错误或遗漏,还望多多指出,仅做抛砖引玉。

测试环境:  
   Dom0：CentOS 6.7（Kernel:2.6.32）  
   Xen安装源: <http://mirrors.aliyun.com/centos/6.7/xen4/x86_64/>  
   Xen环境:  
       Kernel： 3.18.21-16.el6.x86\_64  
       Xen： xen-4.4.3-3.el6.gz  
         
Xen的安装:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # 配置yum源:         [root@centos6 ~]# cat  /etc/yum.repos.d/xen.repo                      [Aliyun]                      name=aliyun-xen-mirrors                      baseurl=http://mirrors.aliyun.com/centos/6.7/xen4/x86\_64/                      enabled=1                      gpgcheck=0 |

|  |  |
| --- | --- |
| 1  2 | # 安装Xen         [root@centos6 ~]# yum install xen |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | # 配置启动Xen:  [root@centos6 ~]# cat  /boot/grub/grub.conf     default=0     timeout=5     splashimage=(hd0,0)/grub/splash.xpm.gz     hiddenmenu     title Xen-Dom0-CentOS (3.18.21-16.el6.x86\_64)         root (hd0,0)         kernel   /xen.gz dom0\_mem=1024M cpufreq=xen dom0\_max\_vcpus=2 dom0\_vcpus\_pin         module  /vmlinuz-3.18.21-16.el6.x86\_64 ro root=UUID=11753f30-71af-4c50-8f64-32ddbed68711                 nomodeset rd\_NO\_LUKS KEYBOARDTYPE=pc KEYTABLE=us.UTF-8 rd\_NO\_MD SYSFONT=latarcyrheb-sun16                 crashkernel=auto rd\_NO\_LVM rd\_NO\_DM rhgb quiet         module  /initramfs-3.18.21-16.el6.x86\_64.img     title CentOS 6 (2.6.32-573.el6.x86\_64)         root (hd0,0)         kernel /vmlinuz-2.6.32-573.el6.x86\_64 ro root=UUID=11753f30-71af-4c50-8f64-32ddbed68711 nomodeset                 rd\_NO\_LUKS KEYBOARDTYPE=pc KEYTABLE=us.UTF-8 rd\_NO\_MD SYSFONT=latarcyrheb-sun16 crashkernel=auto                 rd\_NO\_LVM rd\_NO\_DM rhgb quiet         initrd /initramfs-2.6.32-573.el6.x86\_64.img |

创建一台cirros虚拟机:  
       (1) 下载Cirros: <http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img>  
       (2) 创建image存储目录

|  |  |
| --- | --- |
| 1  2 | [root@centos6 ~]# mkdir  -pv  /images/xen/                [root@centos6 ~]# cp  cirros-0.3.4-x86\_64-disk.img   /images/xen/cirros.img |

       (3) 创建Xen DomU的启动配置文件:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | [root@centos6 ~]# cd  /etc/xen                [root@centos6 ~]# cp  xlexample.pvlinux   cirros                # 修改cirros后配置如下:                [root@centos6 ~]# grep -Ev '^$|^#'   cirros                      name = "cirros-001"                      bootloader="pygrub"                      memory = 512                      vcpus = 2                      vif = [ 'bridge=xenbr0' ]                      disk = [ '/images/xen/cirros.img,qcow2,xvda,rw' ] |

       (4) 创建桥接口

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #注: 在创建桥接口前，需要先注意禁止NetworkManager服务启动.                 #       service  NetworkManager  stop  &&  chkconfig  NetworkManager  off                 #桥接口的作用:                 #     DomU启动后,它若需要与Dom0或外网通信就需要通过Dom0来实现,而Dom0实现的方式就是                 # 创建桥接口,桥接后,系统会将Dom0的物理网卡模拟成一个二层交换机,然后,创建一个虚接口代替                 # ethX, 这样当物理网卡收到目的MAC是自己的数据包时,就转发到虚接口上.若不是则转发给后端DomU.                 [root@centos6 ~]# cat ifcfg-eth0                      DEVICE=eth0                      TYPE=Ethernet                      ONBOOT=yes                      NM\_CONTROLLED=no                      BOOTPROTO=none                      BRIDGE=xenbr0                      USERCTL=no                   [root@centos6 ~]# cat ifcfg-xenbr0                      DEVICE=xenbr0                      TYPE=Bridge                      ONBOOT=yes                      NM\_CONTROLLED=no                      BOOTPROTO=none                      IPADDR=192.168.137.132                      NETMASK=255.255.255.0                      GATEWAY=192.168.137.1                      DNS1=114.114.114.114                      USERCTL=no                      DELAY=0 |

       (5) 测试启动

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # 注：测试启动后,请注意网卡的MAC地址. 由于上面配置文件中没有指定MAC地址,因此每次启动后,        #       其MAC地址会被Xen自动重新分配.        #   -c : 启动的同时进入控制台.        #   -n : 检查配置文件,非真正启动.        [root@centos6 ~]# xl  create  cirros  -c  #启动并登录DomU的终端后,若要退回的Dom0,按: Ctrl+],                                                  # 若退回到Dom0后, Xshell 等连接工具不能正常显示:可输入reset来重置.        [root@centos6 ~]# xl  console  cirros-001  #手动登录cirros的控制台 |

       基本命令使用说明

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | xl  help  create        #查看创建DomU的帮助             xl  -v  create  /etc/xen/cirros  -n            #测试配置文件是否符合需求.             xl  create  /etc/xen/cirros  -c                 #创建并启动成功后,立刻连接到其控制台上.             xl  -v  create  /etc/xen/cirros                 #启动cirros虚拟机             xl  list        #查看当前运行的VM情况             xl  console   cirros-001           #连接到刚刚启动的cirros虚拟机的控制台.              ctrl + ]                                      #退回到Dom0.              exit                                          #拆除cirros,即销毁cirros.             xl  destroy  cirros-001            #关机并销毁cirros, 注:销毁后,下次再用配置文件启动DomU时,就是一个新的DomU了. |

           启动cirros虚拟机登录后:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | ifconfig  -a                                                #是看不到任何接口的.                  insmod  /lib/modules/xen-netfront.ko        #安装网卡接口驱动模块                  ifconfig  -a                  ifconfig  eth0  192.168.1.20  up                  ctrl + ]            # 回到到Dom0,查看网络接口状态.                  Dom0:  ifconfig  -a     #将可以看到一个新接口.vif#.@                                          # "#": 表示DomU的ID ；  @:表示Dom0上创建的桥ID.                  Dom0:  brctl  show     #将看到新接口已经被桥接到Dom0的桥设备上了。 |

   xl的其它常用命令:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | xl  shutdown   DomU\_Name          xl  reboot   DomU\_Name          xl  pause   DomU\_Name         #暂停DomU          xl  unpause  DomU\_Name            xl  save  DomU\_Name    /path/to/save\_name.img    /etc/xen/ConfigFile #将DomU挂起。          xl  restore  /etc/xen/ConfigFile   /path/to/save\_name.img            #恢复DomU到运行态，无需指定DomU\_Name.            xl  help  cd-insert              #插入光驱,仅适用于HVM模式          xl  help  cd-eject               #拔出光驱            xl  vcpu-list   DomU\_Name        #显示DomU上有几颗真正使用的虚拟CPU.                                           #CPU Affinity：表示该虚拟CPU线程可以运行在哪些物理核心上.          xl  vcpu-pin  DomU\_ID  0  3      #将DomU的0号VCPU绑定到物理核心3上.          xl  vcpu-set  DomU\_Name  1       #设置DomU当前只能使用1颗VCPU.若当前它有多颗VCPU,则其它CPU将被暂停.                                           #注: 若在配置文件中指定了maxvcpus ,则此命令可动态添加VCPU数量.                                           #      否则只能在当前VCPU总数的基础上减少。            xl  pci-list   DomU\_Name        #显示当前DomU上的PCI设备.          xl  pci-attach  DomU\_Name       #给运行时的DomU添加一块PCI设备.          xl  pci-detach  DomU\_Name       #将运行的DomU中的PCI设备直接移除。            xl  info            #查看当前Xen Hypervisor的摘要描述信息              -- nr\_cpus      #物理核心总数              -- max\_cpu\_id   #最大支持的VCPU个数                              #更详细的信息参见：http://smilejay.com/2012/03/xl-info\_xm-info/            xl  demsg DomU\_Name         #查看DomU启动时信息.          xl  top                     #查看DomU的资源使用情况.            xl  network-list   DomU\_Name                      #查看DomU的网络接口列表          xl  network-attach  DomU\_Name  bridge="xenbr1"    #给运行中DomU添加一块虚拟网卡.          xl  network-detach  DomU\_Name  1                  #拆除网卡编号为1的网卡.            qemu-img  create  -f  qcow2  -o  size=3G,preallocation=metadata  /images/xen/busybox.2.img          xl  block-list   DomU\_Name            #查看当前DomU\_Name的磁盘列表.          xl  block-attach  DomU\_Name   '/images/xen/busybox.2.img,qcow2,xvdb,rw'    #给DomU添加一个磁盘设备.          xl  block-detach  DomU\_Name   DevID   #移除一块磁盘.block-list中Vdev项就是DevID.            xl  uptime  DomU\_Name       #查看DomU的运行时长. |

基本功能测试部分:  
   (6) 测试打开本地图形窗口:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | [root@centos6 ~]# echo  "vfb='['sdl=1']'" >> /etc/xen/cirros      #开启仅本机访问的VNC窗口.    # 测试失败,报一下错误:    # Parsing config from cirros    # libxl: notice: libxl\_numa.c:494:libxl\_\_get\_numa\_candidate: NUMA placement failed, performance might be affected   # libxl: error: libxl\_dm.c:1401:device\_model\_spawn\_outcome: domain 1 device model: spawn failed (rc=-3)   # libxl: error: libxl\_create.c:1189:domcreate\_devmodel\_started: device model did not start: -3   # libxl: error: libxl\_dm.c:1505:kill\_device\_model: Device Model already exited     # 测试使用VNC远程连接打开.   # 开启可远程连接的VNC服务.  [root@centos6 ~]# echo "vfb=['vnc=1,vnclisten=0.0.0.0,vncpasswd=123456']"   >>  /etc/xen/cirros |

   (7) 测试动态添加和拆除磁盘

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | [root@centos6 ~]# qemu-img  create  -f  raw  -o  size=1G   /images/xen/cirros-01.img             [root@centos6 ~]# xl  block-attach   cirros-001  '/images/xen/cirros-01.img,raw,xvdb,rw'             [root@centos6 ~]# xl  console  cirros-001             [root@centos6 ~]#  fdisk -l |grep 'Disk /dev'                  Disk /dev/xvda: 41 MB, 41126400 bytes                  Disk /dev/xvdb: 1073 MB, 1073741824 bytes       #可以看到已经添加成功.               [root@centos6 ~]# xl block-list cirros-001                  Vdev  BE  handle state evt-ch ring-ref BE-path                  51728 0   2      4     17     793      /local/domain/0/backend/vbd/2/51728    #一定注意:新加的磁盘不一定是显示在末行.                  51712 0   2      4     15     8        /local/domain/0/backend/qdisk/2/51712               [root@centos6 ~]# xl block-detach cirros-001 51728            #拆除磁盘.                  DEBUG libxl\_\_device\_destroy\_tapdisk 105 type=aio:/images/xen/cirros-01.img disk=:/images/xen/cirros-01.img               [root@centos6 ~]# xl -v block-list cirros-001                  Vdev  BE  handle state evt-ch ring-ref BE-path                  51712 0   2      4     15     8        /local/domain/0/backend/qdisk/2/51712 |

   (8) 测试动态添加网卡

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [root@centos6 ~]# xl network-attach  cirros-001           #不加参数也可以添加虚拟网卡.            [root@centos6 ~]# xl network-attach cirros-001 'ip=1.1.1.2,bridge=xenbr1'         #添加参数，但IP参数没有生效.不知为何？            [root@centos6 ~]# xl network-list cirros-001                  Idx BE Mac Addr.         handle state evt-ch   tx-/rx-ring-ref BE-path                  0   0  00:16:3e:0e:ae:e1     0     4     16   775/774         /local/domain/0/backend/vif/1/0                  1   0  00:16:3e:68:3e:4f     1     4     17  1280/1281        /local/domain/0/backend/vif/1/1                  2   0  00:16:3e:77:0b:eb     2     4     18  1792/1793        /local/domain/0/backend/vif/1/2 |

   (9) 保存DomU当前的内存数据到指定位置.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | # 注:默认挂起后,会destroy DomU.              #    -p : 保存当前时刻的内存状态后，将DomU暂停在内存中.              #    -c : 保存当前时间的内存状态后, DomU继续运行.              [root@centos6 ~]# xl save -p cirros-001 /tmp/cirros-04070526.img /etc/xen/cirros                  Saving to /tmp/cirros-04070526.img new xl format (info 0x0/0x0/1307)                  xc: Saving memory: iter 0 (last sent 0 skipped 0): 131072/131072  100%              [root@centos6 ~]# xl list                  Name                                        ID   Mem VCPUs State Time(s)                  Domain-0                                     0  1024     2     r-----     610.2                  cirros-001                                   2   512     2     --p---       1.5              [root@centos6 ~]# xl console cirros-001                  [10489.060379] Freezing user space processes ... (elapsed 0.01 seconds) done.                [root@centos6 ~]# xl save  -c  cirros-001 /tmp/cirros-04070538.img /etc/xen/cirros                  Saving to /tmp/cirros-04070538.img new xl format (info 0x0/0x0/1307)                  xc: Saving memory: iter 0 (last sent 0 skipped 0): 131072/131072  100%              [root@centos6 ~]# xl list                  Name                                        ID   Mem VCPUs State Time(s)                  Domain-0                                     0  1024     2     r-----     682.5                  cirros-001                                   2   512     2     -b----       2.1 |

   (10) 恢复保存的DomU状态

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | [root@centos6 ~]# xl restore /etc/xen/cirros /tmp/cirros-04070538.img                  Loading new save file /tmp/cirros-04070538.img (new xl fmt info 0x0/0x0/1307)                   Savefile contains xl domain config                  Parsing config from /etc/xen/cirros                  libxl: notice: libxl\_numa.c:494:libxl\_\_get\_numa\_candidate: NUMA placement failed, performance might be affected                  xc: Reloading memory pages: 131072/131072  100%              # 注: 在保存内存数据前,我事先为cirros添加了一块磁盘 和 一块网卡.              #      请注意: 恢复DomU后，动态关联的网卡和磁盘都已经被去除了。              [root@centos6 ~]# xl network-list cirros-001                  Idx BE Mac Addr.         handle state evt-ch   tx-/rx-ring-ref BE-path                  0   0  00:16:3e:48:55:6d     0     4     16   775/774         /local/domain/0/backend/vif/5/0              [root@centos6 ~]# xl block-list cirros-001                  Vdev  BE  handle state evt-ch ring-ref BE-path                  51712 0   5      4     15     8        /local/domain/0/backend/qdisk/5/51712 |

   (11) 测试动态增加内存和CPU：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | [root@centos6 xen]# xl list                  Name                                        ID   Mem VCPUs State Time(s)                  Domain-0                                     0  1024     2     r-----     932.5                  cirros-001                                   6   512     1     -b----       4.3                  busybox-001                                  7   256     1     -b----      10.0             [root@centos6 xen]# grep -Ev '^$|^#' cirros                  name = "cirros-001"                  bootloader="pygrub"                  memory = 512                  maxmem = 1024                  vcpus = 1                  maxvcpus = 3                  cpus = "0-2"                            #将CPU绑定到0,1,2这三颗物理核心上.                  vif = [ 'ip=10.1.0.10,bridge=xenbr0' ]  # IP参数设置后,测试发现总是不生效,不知原因为何？                  disk = [ '/images/xen/cirros.img,qcow2,xvda,rw' ]                  vfb = [ 'vnc=1,vncpasswd=123456,vnclisten=0.0.0.0' ]              [root@centos6 xen]# grep -Ev '^$|^#' busybox                  name = "busybox-001"                  kernel = "/boot/vmlinuz"                  ramdisk = "/boot/initramfs.img"                  extra = "selinux=0 init=/bin/sh "                  root = "/dev/xvda ro"                  memory = 256                  vcpus = 1                  vif = [ 'ip=10.1.0.11,bridge=xenbr1' ]                  disk = [ '/images/xen/busybox.img,raw,xvda,rw' ]              [root@centos6 xen]# xl mem-set cirros-001 600m            [root@centos6 xen]# xl mem-set busybox-001 300m                  libxl: error: libxl.c:4106:libxl\_set\_memory\_target: memory\_dynamic\_max must be less than or equal to memory\_static\_max            [root@centos6 xen]# xl list                  Name                                        ID   Mem VCPUs State Time(s)                  Domain-0                                     0  1024     2     r-----     945.7                  busybox-001                                  7   256     1     -b----      10.2                  cirros-001                                   8   600     1     -b----       2.8            # 上面测试结果显示: 必须在配置文件中启动最大内存(maxmem)参数,才能使用命令来动态调整Memory.          #  能否使用mem-max动态增加最大内存那？          [root@centos6 xen]# xl mem-max busybox-001 512m          [root@centos6 xen]# xl list              Name                                        ID   Mem VCPUs State Time(s)              Domain-0                                     0  1024     2     r-----     947.4              busybox-001                                  7   256     1     -b----      10.2              cirros-001                                   8   600     1     -b----       2.9          [root@centos6 xen]# xl mem-set busybox-001 300m              libxl: error: libxl.c:4106:libxl\_set\_memory\_target: memory\_dynamic\_max must be less than or equal to memory\_static\_max          [root@centos6 xen]# xl list              Name                                        ID   Mem VCPUs State Time(s)              Domain-0                                     0  1024     2     r-----     947.7              busybox-001                                  7   256     1     -b----      10.2     #测试结果: 必须修改配置文件.              cirros-001                                   8   600     1     -b----       2.9            [root@centos6 xen]# xl vcpu-set cirros-001 3          [root@centos6 xen]# xl vcpu-list cirros-001              Name                                ID  VCPU   CPU State   Time(s) CPU Affinity              cirros-001                           8     0    0   -b-       2.7  all      #注: 动态调整VCPU个数后,并没有动态激活更多的VCPU.              cirros-001                           8     1    -   --p       0.1  all       #    问题原因,目前还不清楚...              cirros-001                           8     2    -   --p       0.1  all       # 在DomU中查看 cat  /proc/cpuinfo,也只有一颗. |

   (12) 测试创建克隆DomU.  
           1) 类似VMware中的连接克隆.

|  |  |
| --- | --- |
| 1  2  3  4  5 | #注: 通过qemu-img来创建一个关联后端文件的img镜像文件.             #    来测试连接克隆,但测试结果是失败的。             # 尝试发现: 将连接镜像文件做为磁盘启动后,再里面所做的修改,在关机后会同步到后端镜像文件中。             # 另注: 个人认为连接克隆不适合线上操作, 仅个人使用时节省空间是比较好的选择。因此,可不必深究.             qemu-img  create  -f  qcow2  -o  backing\_file=/images/xen/cirros.img    /images/xen/cirros.link.img |

           2) 直接复制HVM DomU的img映像文件来克隆DomU.  
               # Linux测试没有发现太多异常,除了MAC会在每次create时重新生成外,相互访问正常,水平有限,望高手勿喷。  
               # Windows的测试,此处采用WinXP来测试,WinXP仅支持HVM模式(完全虚拟化)下测试.  
               (a) 我的CPU支持VT-x,因此在VMware中开启了VT-x输出给VM的功能:



                (b) 提供WinXP的基本配置文件

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | [root@centos6 xen]# grep -Ev '^$|^#' winxp                   builder = "hvm"                   name = "WinXP-001"                   memory = 1024                   vcpus = 2                   vif = [ 'bridge=xenbr0' ]                   disk = [ '/images/xen/winxp.img,qcow2,xvda,rw' , '/mnt/win7/windows/WinXPSP3-Vol.iso,,hdc,cdrom']                   vnc = 1 |

              (c)启动WinXP后,连接VNC开始从CDROM安装WinXP.

|  |  |
| --- | --- |
| 1  2 | #注:我的笔记本内存只有8G,分配给Xen VM 4G,但安装WinXP依然巨慢,安装了一个晚上才基本装好.                 vncviewer  127.0.0.1:5900 |

              (d)克隆WinXP的映像文件

|  |  |
| --- | --- |
| 1 | cp  -a   /images/xen/winxp.img   /images/xen/winxp-2.img |

              (e)测试结果:  
                   > WinXP可以启动,MAC地址可以在配置文件中指定来保障其不是每次重启后都改变.  
                   > 但查询Windows的SID时, 直接复制的映像文件与原有的映像文件的SID是相同的.  
                       这也是应该是必然,Xen还不太可能每次create时去修改Windows的SID.  
                  注:  
                       Windows查看当前用户的SID:  
                           开始–>运行–>regedit–>HKEY\_USERS 下面两个就是当前用户的SID.  
                              如: S-1-5-21-861567501-1326574676-682003330-1003  
                    Windows SID的重要性: <http://www.cnblogs.com/mq0036/p/3518542.html>  
                    SID = Security Identifiers，安全标识符，是标识用户、组和计算机帐户的唯一的号码。  
             如果两台电脑的SID相同，在一个局域网里就会发生冲突，比如自己GHOST了系统，然后还原到其它电脑上，  
             这时候的SID是相同的，就会产生冲突。另外A和B两台PC机的管理员SID相同,则A上仅管理员可访问的共享,  
             B上的管理员也将可访问,当然A也能访问B.

   (13) Xen的实时迁移

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # xl  命令做迁移比较简单,直接使用以下命令即可实现:              # -C : 指定DomU的配置文件,非必须.              # 192.168.137.151: 目标Xen Hypervisor的IP.              # 注: 迁移的条件:              #       (1) busybox-001的磁盘映像文件，在目标Xen Hypervisor上必须能访问到.              #       (2) 两边的/etc/hosts文件最好一致.因为,xl 的迁移实际上是采用ssh隧道来完成的.              # 另注: 由于测试时,启动两台Xen Hypervisor后,总是会导致其中一台在不确定的时间挂掉.              #        导致没能做的太详细,我在迁移时,先是将busybox-001的磁盘映像文件拷贝到目标机上后,              #        在执行以下命令就成功了。但要注意目录必须与busybox配置文件中一致.              # 当然,最好是先用iSCSI或NFS，Samba等将磁盘映像文件共享,在做以下操作会更真实.                  xl  migrate  -C  /etc/xen/busybox   busybox-001   192.168.137.151 |

   (14) 关于Xen提供的批量启动、停止或迁移DomU的脚本—-xendomains的分析说明:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246 | #注: 此脚本存在一个小Bug,需要小小的修改下.              #     下面分析仅仅对start 和 stop函数做简略说明，其它部分可自行参看: /etc/init.d/xendomains              #      xendomains: 会读取/etc/sysconfig/xendomains这个默认参数配置文件.              parseln()                  {                      if [[ "$1" =~ '(domain' ]] || [[ "$1" = "{" ]]; then                          name=;id=                      elif [[ "$1" =~ '(name' ]]; then                          name=$(echo $1 | sed -e 's/^.\*(name \(.\*\))$/\1/')                      elif [[ "$1" =~ '(domid' ]]; then                          id=$(echo $1 | sed -e 's/^.\*(domid \(.\*\))$/\1/')                      elif [[ "$1" =~ '"name":' ]]; then                          name=$(echo $1 | sed -e 's/^.\*"name": "\(.\*\)",$/\1/')                      elif [[ "$1" =~ '"domid":' ]]; then                          id=$(echo $1 | sed -e 's/^.\*"domid": \(.\*\),$/\1/')                      fi                        # 此处即上上面所说的小Bug:                      #此处是手动添加的,因为,parseln函数是用来统一获取DomU的name、id的,                      #但从脚本中看,作者是希望从这里获取DomU的运行状态的,但似乎忘记写了.                      state=$(xl list |awk -v DomUname=$name '$1==DomUname{print $5}')                        [ -n "$name" -a -n "$id" ] && return 0 || return 1                  }                  stop()                  {                  exec 3>&2 2> /dev/null#此句是将当前进程的文件描述符 绑定到系统标准错误输出.                      # Collect list of domains to shut down                      if test "$XENDOMAINS\_AUTO\_ONLY" = "true"; then  rdnames; fi                      echo -n "Shutting down Xen domains:"                      name=;id=                      while read LN; do                  parseln "$LN" || continue                  if test $id = 0; then continue; fi                  echo -n " $name"                  # XENDOMAINS\_AUTO=/etc/xen/auto/                  # XENDOMAINS\_AUTO\_ONLY='true' 此为默认值.                  # 自动关闭/etc/xen/auto/目录下所有的DomU. 此目录下放置DomU的配置文件.                  # 但此处的并没有真正实现自动关闭。                  # NAMES: 是rdnames函数定义的变量.它保存了/etc/xen/auto中所有DomU的名字,                  # NAMES="DomU1|DomU2|DomU3|...."                  if test "$XENDOMAINS\_AUTO\_ONLY" = "true"; then                  #这里原本是关闭:/etc/xen/auto/目录下所有打算自动启动的DomU.                  #但stop函数中并没有实际去关闭,而是留空了.                  fi                  # XENDOMAINS\_SYSRQ chould be something like just "s"                  # or "s e i u" or even "s e s i u o"                  # for the latter, you should set XENDOMAINS\_USLEEP to 1200000 or so                  # /etc/sysconfig/xendomains中对XENDOMAINS\_SYSRQ的注解:                  # The xendomains script can send SysRq requests to domains on shutdown.                  # If you don't want to MIGRATE, SAVE, or SHUTDOWN, this may be a possibility                  # to do a quick and dirty shutdown ("s e i u o") or at least sync the disks                  # of the domains ("s").                  # SysRq:的含义我没有弄明白.若有知情着,还望赐教.                  if test -n "$XENDOMAINS\_SYSRQ"; then                  for sysrq in $XENDOMAINS\_SYSRQ; do                  echo -n "(SR-$sysrq)"                  # CMD=/usr/sbin/xl                  XMR=`$CMD sysrq $id $sysrq 2>&1 1>/dev/null`                  if test $? -ne 0; then                  echo -e "\nAn error occurred while doing sysrq on domain:\n$XMR\n"                  rc\_failed $?                  echo -n '!'                  fi                  # usleep just ignores empty arg                  usleep $XENDOMAINS\_USLEEP                  done                  fi                  # 这里是判断是否存在僵尸DomU, 需要注意: state变量并未定义,此可能是个Bug.                  # 可自行修复下,在parseln函数中添加"state=$($CMD list |awk -v DomUName=$name '$1==DomUName{print $5}')"                  if test "$state" = "-b---d" -o "$state" = "-----d"; then                  echo -n "(zomb)"                  continue                  fi                  #这里是用来判断MigrateServer是否指定了,若指定了,则开始批量迁移DomU到指定的XenHypervisor.                  #指定XENDOMAINS\_MIGRATE,可直接修改/etc/sysconfig/xendomains。                  if test -n "$XENDOMAINS\_MIGRATE"; then                  echo -n "(migr)"                  #启动一个监视迁移的线程.默认若在300秒内没有迁移完成,则监控线程将认为迁移超时,                  #而强制kill掉迁移线程.                  watchdog\_xencmd migrate &                  #获取监视迁移线程的PID,若在300秒内迁移完成,则kill掉监视迁移的线程.                  WDOG\_PID=$!                  # CMD=/usr/sbin/xl                  #启动DomU迁移.                  XMR=`$CMD migrate $id $XENDOMAINS\_MIGRATE 2>&1 1>/dev/null`                  if test $? -ne 0; then                  echo -e "\nAn error occurred while migrating domain:\n$XMR\n"                  rc\_failed $?                  echo -e '!'                  kill $WDOG\_PID >/dev/null 2>&1                  else                  kill $WDOG\_PID >/dev/null 2>&1                  echo -e .                  usleep 1000                  #若迁移被设置了,则所有即将被关闭的DomU都将被迁移到指定Xen Hypervisor上.                  #下面的save 和 shutdown就不会被执行了。                  continue                  fi                  fi                  if test -n "$XENDOMAINS\_SAVE"; then                  echo -n "(save)"                  # 将Shell函数放到后台运行.                  watchdog\_xencmd save &                  # 获取这个后台执行的函数线程的PID.                  WDOG\_PID=$!                  # Default：XENDOMAINS\_SAVE=/var/lib/xen/save                  mkdir -p "$XENDOMAINS\_SAVE"                  # 这里开始对DomU进行挂起,挂起操作的默认超时时间是300秒,watchdog\_xencmd负责监控下面命令执行是否超时.                  # 若超时,watchdog\_xencmd会将其自动强制结束.                  XMR=`$CMD save $id $XENDOMAINS\_SAVE/$name 2>&1 1>/dev/null`                  # 若以上命令执行成功,则强制将watchdog\_xencmd线程结束掉.                  if test $? -ne 0; then                  echo -e "\nAn error occurred while saving domain:\n$XMR\n"                  rc\_failed $?                  echo -e '!'                  kill $WDOG\_PID >/dev/null 2>&1                  else                  kill $WDOG\_PID >/dev/null 2>&1                  echo -e .                  usleep 1000                  continue                  fi                  fi                  # XENDOMAINS\_SHUTDOWN=--wait                  if test -n "$XENDOMAINS\_SHUTDOWN"; then                  # XENDOMAINS\_SHUTDOWN should be "--wait"                  echo -n "(shut)"                  watchdog\_xencmd shutdown &                  WDOG\_PID=$!                  XMR=`$CMD shutdown $XENDOMAINS\_SHUTDOWN $id 2>&1 1>/dev/null`                  if test $? -ne 0; then                  echo -e "\nAn error occurred while shutting down domain:\n$XMR\n"                  rc\_failed $?                  echo -e '!'                  fi                  kill $WDOG\_PID >/dev/null 2>&1                  fi                      done < <($CMD list -l | grep "$LIST\_GREP")                      # NB. this shuts down ALL Xen domains (politely), not just the ones in                      # AUTODIR/\*                      # This is because it's easier to do "；-)" but arguably if this script is run                      # on system shutdown then it's also the right thing to do.                  # all\_zombies返回0,表示没有找到僵尸DomU,返回1,表示找到僵尸DomU.                      if ! all\_zombies && test -n "$XENDOMAINS\_SHUTDOWN\_ALL"; then                  # XENDOMAINS\_SHUTDOWN\_ALL="--all --wait"                  echo -n " SHUTDOWN\_ALL "                  watchdog\_xencmd shutdown 1 false &                  WDOG\_PID=$!                  XMR=`$CMD shutdown $XENDOMAINS\_SHUTDOWN\_ALL 2>&1 1>/dev/null`                  if test $? -ne 0; then                  echo -e "\nAn error occurred while shutting down all domains: $XMR\n"                  rc\_failed $?                  echo -e '!'                  fi                  kill $WDOG\_PID >/dev/null 2>&1                      fi                      # Unconditionally delete lock file                      rm -f $LOCKFILE                  }                  start()                  {                  # LOCKFILE=/var/lock/subsys/xendomains                      if [ -f $LOCKFILE ]; then                  echo -e "xendomains already running (lockfile exists)"                  return;                      fi                      saved\_domains=" "                  # XENDOMAINS\_RESTORE=true                  # XENDOMAINS\_SAVE=/var/lib/xen/save                  # contains\_something :检查指定目录下是否有文件,有则返回0,无则返回1                      if [ "$XENDOMAINS\_RESTORE" = "true" ] &&                         contains\_something "$XENDOMAINS\_SAVE"                      then                  # $(dirname "$LOCKFILE") = /var/lock/subsys                  mkdir -p $(dirname "$LOCKFILE")                  touch $LOCKFILE                  echo -n "Restoring Xen domains:"                  saved\_domains=`ls $XENDOMAINS\_SAVE`                  #这里是用来恢复所有调用stop函数被挂起(save)的DomU。                          for dom in $XENDOMAINS\_SAVE/\*; do                              if [ -f $dom ] ; then                                  HEADER=`head -c 16 $dom | head -n 1 2> /dev/null`                                  #HEADCOMP="LinuxGuestRecord" : 这是xm save DomU后,其保存文件中头部16个字符.                                  #HEADCOMP="Xen saved domain" : 这是xl save DomU后,其保存文件中头部16个字符.                                  if [ "$HEADER" = "$HEADCOMP" ]; then                                      echo -n " ${dom##\*/}"                                      # CMD=/usr/sbin/xl                                      XMR=`$CMD restore $dom 2>&1 1>/dev/null`                                      #$CMD restore $dom                                      if [ $? -ne 0 ]; then                                          echo -e "\nAn error occurred while restoring domain ${dom##\*/}:\n$XMR"                                          rc\_failed $?                                          echo -e '!'                                      else                                          # mv $dom ${dom%/\*}/.${dom##\*/}                                          # 若恢复成功则删除XENDOMAINS\_SAVE中相应的DomU保存的镜像.                                          rm $dom                                      fi                                  fi                              fi                          done                  echo -e                      fi                      # contains\_something :检查指定目录下是否有文件,有则返回0,无则返回1                      if contains\_something "$XENDOMAINS\_AUTO"                  then                  touch $LOCKFILE                  echo -n "Starting auto Xen domains:"                  # We expect config scripts for auto starting domains to be in                  # XENDOMAINS\_AUTO - they could just be symlinks to files elsewhere                  # Create all domains with config files in XENDOMAINS\_AUTO.                  # TODO: We should record which domain name belongs                  # so we have the option to selectively shut down / migrate later                  # If a domain statefile from $XENDOMAINS\_SAVE matches a domain name                  # in $XENDOMAINS\_AUTO, do not try to start that domain; if it didn't                  # restore correctly it requires administrative attention.                  # XENDOMAINS\_AUTO=/etc/xen/auto                  # 这是自动列出/etc/xen/auto/下所有DomU的配置文件,并启动.                  for dom in $XENDOMAINS\_AUTO/\*; do                  # dom=/etc/xen/auto/busybox-001                  # ${dom##\*/} = busybox-001                  echo -n " ${dom##\*/}"                  # $(echo $dom | sed -n 's/^.\*\/\(.\*\)$/\1/p') = busybox-001                  shortdom=$(echo $dom | sed -n 's/^.\*\/\(.\*\)$/\1/p')                  # grep -w :表示单词匹配模式                  echo $saved\_domains | grep -w $shortdom > /dev/null                  #is\_running:判断提供的DomU是否正在运行.0表示正在运行.                  if [ $? -eq 0 ] || is\_running $dom; then                  echo -n "(skip)"                  else                  XMC=`$CMD create --quiet --defconfig $dom`                  if [ $? -ne 0 ]; then                  echo -e "\nAn error occurred while creating domain ${dom##\*/}: $XMC\n"                  rc\_failed $?                  echo -e '!'                  else                  # usleep函数是延时函数.                  # XENDOMAINS\_CREATE\_USLEEP=5000000                  # usleep延时计算方法:sleep $1/1000000                  usleep $XENDOMAINS\_CREATE\_USLEEP                  fi                  fi                  done                      fi                  } |

   (15) Xenstore的简单说明:  
       列目录

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | xenstore-ls :递归方式列出xenstored中所有的键值对.             xenstore-list :仅显示当前目录中内容.              如:                  [root@node1 xen]# xenstore-list -s /   #列出xenstored根目录树中的内容.                      tool       : 暂时没有数据.                      local     ：存放了活动虚拟机配置和驱动信息                      vm        :存放虚拟机管理信息                      libxl       :这是一个空文件。用途不详. |

      读写

|  |  |
| --- | --- |
| 1  2 | xenstore-read : 可用来读取xenstored中文件的内容.             xenstore-write : 可用来修改xenstored中的内容。 |

      建立/删除目录

|  |  |
| --- | --- |
| 1  2 | xenstore-mkdir             xenstore-rm |

     监视

|  |  |
| --- | --- |
| 1 | xenstore-watch |

附件1：  
Xen已解决的安全问题【原文:<http://www.51ou.com/browse/xen/50466.html>】  
   （1）Xen的访问控制  
           Xen通过自己的访问控制模块ACM。能有效解决诸多访问不当造成的安全问题。  
           ACM实现了两种策略机制，分别是中国墙（Chinesewall）策略和简单类型强制（simple type enforcement：STE）策略。其中，中围墙策略定义了一组中国墙类型，并因此定义冲突集，然后根据类型定义标签。该策略根据标签来进行判断，若两个虚拟机的标签处在同一个冲突集中。则不能同时在相同的系统上运行。因此该机制主要用于虚拟机之间的信息流控制。STE策略亦定义了一组类型（该类型针对的是域所拥有的资源），然后根据类型定义标签，要求当上体拥有客体标签时。主体才能访问客体。以此来控制资源共享。除此以外，Xen的Domain 0用户可以根据自己的需求制定安全策略。当虚拟机请求与别的虚拟机进行通信或访问资源时。ACM模块能根据用户定义的策略判断,以此达到对虚拟机的资源进行控制以及对虚拟机之间的信息流进行控制的目的。

　　（2）Xen的可信计算  
　　      可信计算技术是一种新兴的信息安全手段，其安全措施是通过构建信任链来防御攻击。通过传递机制，在系统启动时可将B10S中最先启动的代码BIOS boot loader作为整个信任链的起点，依次逐级向上传递系统控制权并构建信任链，直到应用层。可信计算可以有效弥补传统安全防议无法提供的有关通信终点节点完整性与可信性差的问题。  
　　      可信计算平台是能够提供可信计算服务的计算机软硬件实体。它能够提供系统的可靠性、可用性以及信息和行为的安全性。因此，建立可信平台是应对云计算安全的一种重要手段。而可信平台模块（TPM）是可信计算的基石。  
　　      TPM实际上是一个含有密码运算部件和存储部件的系统级芯片，是云计算平台重要的防篡改组件，这能有效保证平台的安全。相对于传统物理平台，可信平台在Xen等虚拟化平台上实现TPM，有一定的优越性。平时计算机装了太多软件，这使得构建信任链变得很复杂。相对而言，使用虚拟机（VM）配合虚拟可信平台模块（vTPM）组成虚拟终端可信平台要方便得多。这是因为虚拟终端可信平台通常只为处理某种特定任务而产生，可由用户自定义其所需功能，所以功能相对简单，更容易构建信任链。此平台可以来处理需要安全保障的在线服务或敏感数据的访问。  
　　      现在，Xen 3.0以上的版本都能支持vTPM的实现。vTPM能实现虚拟计算系统中虚拟机的安全可靠。vTPM可以使平台上的每个虚拟机利用其功能。让每个需要TPM功能的虚拟机都感觉是在访问自己私有的TPM一样。在平台搭建中,可以创建多个虚拟TPM，这样每一个如实地效仿硬件TPM的功能，可有效维护各个虚拟机的安全。从而使Xen搭建的云计算平台处于较稳定状态。

　　Xen的现有问题及解决策略  
　　目前的Xen的安全性还有较多的安全问题。比如，Domain 0是一个安全瓶颈，其功能较其他域强，所以容易被敌手发起蠕虫、病毒、DoS等各种攻击，如果Domain 0瘫痪或者被敌手攻破，那么将破坏整个虚拟机系统。Xen的隐通道问题没有解决。在Xen上就不可能运行高安全等级的操作系统。虚拟机共享同一套硬件设备，一些网络安全协议可能更加容易遭到恶意破坏和恶意实施。Xen提供了方便的保存和恢复机制。使得操作系统数据的回滚和重放非常容易，但这些将影响操作本身的密码特性。除此之外，在Xen中，由于安全机制做在Guest OS中，所以不能保证VMM的安全。Xen只能限制页表一级的内存I/O地址空间。而中断和I/0端口地址空间的粒度要比页表小得多。如果不同虚拟机中的驱动不幸被分配到同一个页表空间，那么它们就可以访问对方的内存地址空间，造成安全问题。  
　　针对Domain 0的问题。可削弱它的功能。将其功能分解到其他域。这将会适当减少Domain 0的瓶颈作用。对于敏感数据要进行多次擦除防止再恢复。另外，Xen的ACM模块不能完全解决设备隔离和资源隔离问题，将Xen和LaGrande技术结合是一个不错的选择。LaGrande是lnfel将要实施的一种硬件技术，它是一组通用的硬件安全增强组件。用来防止敏感的信息被恶意软件攻击。其安全功能将被整合到处理器和芯片集中。能有效增强设备隔离，实现I/O保护、内存越界保护、键盘、显示的隔离保护等。

　　业界对虚拟化安全的努力  
　　针对传统安全防火墙技术不能有效监控虚拟机流量的问题，mtor Networks公司使用VMware的API来开发虚拟安全分析器，以检测虚拟交换机流量——在虚拟层之上的网络层流量。该公司也开发了虚拟网络防火墙，该防火墙基于虚拟机管理器，可认证有状态的虚拟防火墙，检查所有通过虚拟机的数据分组，组织所有未经批准的连接和允许对数据分组进行更深层次的检查，确保了虚拟机间通信的安全。针对虚拟环境的安全问题，目前Resolution Enterprise公司提出要对虚拟化环境采取深层防护战略。  
　　除此以外。开源Xen管理程序社区Xen.org已经开始实施Xen云平台（XCP）计划。目的是在云环境中利用领先的Xen管理程序。为未来的云服务提供安全的、经过验证的开源基础设施乎台。目前。已经发布了XCPl.0及其修正版，并将慢慢发布更加稳定的版本。  
        Citrix  
Citrix系统公司是全球领先的应用服务器软件方案供应商。Citrix凭借其卓越的技术方案和业务成就，赢得了业界与用户的广泛赞誉。收购XenSource之后，Ctrix在虚拟化领域的能力进一步提高，具有了足够的技术储备应对云计算挑战。Citrix与其它云端服务供货商的最大不同在于，提供端对端的产品服务：包含Dazzle、XenDesktop、XenApp、NetScaler、XenServer与即将推出的XenClient，以及透过遵循开放标准等方式，与储存设备、服务器业者结盟、推广云端服务产品

附件2：  
qemu-img是qemu用来实现磁盘映像管理的工具组件，其有许多子命令，分别用于实现不同的管理功能，而每一个子命令也都有一系列不同的选项。其使用语法格式为“qemu-img  subcommand  [options]”，支持的子命令如下。

◇ create：创建一个新的磁盘映像文件；  
       格式: qemu-img  -f  DISK\_FORMAT  -o  ?   /tmp/test.img        #查看指定的DISK\_FORMAT所支持的额外特性参数.  
       注:  如果“-o”选项中使用了backing\_file这个选项来指定其后端镜像文件，那么这个创建的镜像文件仅记录与后端镜像文件的差异部分，后端镜像文件不会被修改，除非在QEMU monitor中使用“commit”命令或者使用“qemu-img commit”命令去手动提交这些改动; 这种情况下，size参数不是必须需的，其值默认为后端镜像文件的大小。另外，直接使用“-b backfile”参数也与“-o backing\_file=backfile”效果相同。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | qemu-img create -f qcow2 -b /tmp/rhel6u3.img   /tmp/rhel6u3.qcow2          # 或            qemu-img create -f qcow2 -o backing\_file=/tmp/rhel6u3.img  /tmp/rhel6u3.qcow2          # 注: rhel6u3.qcow2 可以作为镜像文件启动, 有点类似VMware的连接克隆VM,          #       但需要注意: 这并非是VMware的连接克隆,因为,当你启动rhel6u3.qcow2的VM后,          #       你在该VM中创建的文件、目录等会在关闭该VM后，自动同步到rhel6u3.img中.          #       若你启动了rhel6u3.img这个VM,可能会造成rhel6u3.qcow2镜像出现错误. |

◇ check：检查磁盘映像文件中的错误；  
   对磁盘镜像文件进行一致性检查，查找镜像文件中的错误，目前仅支持对“qcow2”、“qed”、“vdi”格式文件的检查。  
   (1) qcow2: QEMU 0.8.3版本引入的镜像文件格式，也是目前使用最广泛的格式。  
   (2) qed（QEMU enhanced disk）: 从QEMU 0.14版开始加入的增强磁盘文件格式,  
         它是为了避免qcow2格式的一些缺点,同时提高性能而引入,目前还不够成熟。  
   (3) vdi（Virtual Disk Image）: Oracle的VirtualBox虚拟机中的存储格式。  
       -f fmt： 指定文件的格式，如果不指定格式qemu-img会自动检测，filename是磁盘镜像文件的名称（包括路径）。

|  |  |
| --- | --- |
| 1 | qemu-img  check  -f qcow2  /tmp/rhel6u3.qcow2 |

◇ convert：转换磁盘映像的格式；  
   格式：qemu-img  convert [-c] [-f fmt] [-O output\_fmt] [-o options] filename [filename2 […]] output\_filename  
       -c :对输出的镜像文件进行压缩,仅支持qcow2和qcow格式,且此压缩是只读的,若压缩的扇区被重写,则会被重写为未压缩的数据。  
       -f : 指定源磁盘映像文件的格式.  
       -O : 指定输出磁盘映像文件的格式.  
       -o : 指定输出磁盘映像文件的其它选项:如:后端镜像、文件大小、是否加密等等  
             注: 使用backing\_file选项来指定后端镜像，让生成的文件是copy-on-write的增量文件，  
                  这时必须让转换命令中指定的后端镜像与输入文件的后端镜像的内容是相同的，  
                  尽管它们各自后端镜像的目录、格式可能不同。  
       注: 一般来说，输入文件格式fmt由qemu-img工具自动检测到，而输出文件格式output\_fmt根据自己需要来指定，默认会被转换为raw文件格式（且默认使用稀疏文件的方式存储以节省存储空间）。另外如果使用qcow2、qcow、cow等作为输出文件格式来转换raw格式的镜像文件（非稀疏文件格式），镜像转换还可以起到将镜像文件转化为更小的镜像，因为它可以将空的扇区删除使之在生成的输出文件中并不存在。

|  |  |
| --- | --- |
| 1 | qemu-img   convert   /tmp/my-vmware.vmdk    /tmp/my-kvm.img |

◇ info：显示指定磁盘映像的信息；  
   注: 如果文件是使用稀疏文件的存储方式，也会显示出它的本来分配的大小以及实际已占用的磁盘空间大小。如果文件中存放有客户机快照，快照的信息也会被显示出来。

|  |  |
| --- | --- |
| 1 | qemu-img   info   /tmp/rhel6u3.img |

◇ snapshot：管理磁盘映像的快照；  
   格式: qemu-img  snapshot [-l | -a snapshot | -c snapshot | -d snapshot] filename  
                   -l： 选项是查询并列出镜像文件中的所有快照  
                   -a  snapshot  ： 是让镜像文件使用某个快照  
                   -c  snapshot  ： 是创建一个快照  
                   -d  snapshot  ： 是删除一个快照。

◇ commit：提交filename文件中的更改到后端支持镜像文件（创建时通过backing\_file指定的）中去；  
   格式: qemu-img  commit [-f fmt] filename

◇ rbase：改变镜像文件的后端镜像文件，只有qcow2和qed格式支持rebase命令；  
   格式: qemu-img  rebase [-f fmt] [-t cache] [-p] [-u] -b backing\_file [-F backing\_fmt] filename  
                   -b backing\_file：指定作为后端磁盘映像的文件  
                   -F backing\_fmt：指定将后端映像文件转换为何种磁盘映像格式。  
               注：rbase操作可工作于两种模式下，安全模式（Safe Mode）【默认模式】和 非安全模式【Unsafe Mode】  
                   在安全模式下qemu-img会去比较原来的后端镜像与现在的后端镜像的不同进行合并处理；  
                   -u 可指定非安全模式（Unsafe Mode），此模式主要用于将后端镜像进行了重命名或移动位置之后对前端镜像文件的修复处理,由用户去保证后端镜像的一致性。

◇ resize：增大或缩减磁盘映像文件的大小；  
   格式: qemu-img  resize filename [+ | -]size  
       + : 增加  
       – ：减少  
       size： 单位支持:K/M/G/T  
       缩小镜像的大小前，需保证客户机里面的文件系统有空余空间，否则会数据丢失，  
 另外，qcow2格式文件不支持缩小镜像的操作。在增加了镜像文件大小后，也需启动客户机到  
 里面去使用“fdisk”、“parted”等分区工具进行相应的操作才能真正让客户机使用到增加后的镜像空间。  
 不过使用resize命令时需要小心（最好做好备份），如果失败的话，可能会导致镜像文件无法正常使用而造成数据丢失。

|  |  |
| --- | --- |
| 1 | qemu-img  resize  /tmp/rhel6u3-a.img  +2G |

附件3：  
   1.1 创建一个Busybox虚拟机示例:  
       (1) 创建一个虚拟磁盘映像文件:

|  |  |
| --- | --- |
| 1 | qemu-img  create  -f  raw  -o  size=2G   /images/xen/busybox.img |

       (2) 将其格式化并挂载:

|  |  |
| --- | --- |
| 1  2  3 | cd  /images/xen                  mkfs.ext2  busybox.img                  mount  -o  loop   /images/xen/busybox.img   /mnt |

       (3) 编译busybox

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # 下载busybox:                   tar  xvjf  busybox-\*.tar.bz2                   cd  busybox\*                   make  menuconfig                  # 注:若busybox需要进行静态编译则需要安装: yum  install  glibc-static ；                  #  此RPM包可以支持静态编译,使编译的应用程序可直接包含所需要的外部库,而非连接载入外部库.                   busybox Settings-->Build Optons-->Build BusyBox as a static binary(no shared libs)                                                     |-->Installation Options（”make install" behavior)                   make                   make  install    #注:安装后,busybox会文件安装到当前编译目录下的\_install目录中.                   cd  \_install                   cp  -a  \*  /mnt  #将busybox的所有文件copy到busybox.img中,作为一个模拟的根文件系统.                   # 提供以下目录,也是十分重要的:                   mkdir  -pv  /mnt/{proc, var, sys, etc, dev, usr, tmp, home, root, opt} |

     (4) chroot测试:

|  |  |
| --- | --- |
| 1 | chroot   /mnt |

     (5) 在Dom0上创建桥设备

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | # 注: 添加桥,并配置DomU启动时,自动桥接到Dom0上创建的桥设备.        #  这里存在一个Kernel-Bug: 若使用kernel-3.6.10-68、3.6.18都存在将物理网卡桥到桥设备后,Kernel无响应的问题.        #  建议: 做桥实验时,采用kernel-3.6.7版本的。                  yum  install  bridge-utils                  vim  /etc/sysconfig/network-scripts/ifcfg-eth0                              DEVICE=eth0                              NM\_CONTROLLED=no                              TYPE=Ethernet                              BOOTPROTO=none                              ONBOOT=yes                              USERCTL=no                              BRIDGE=xenbr0                   vim  /etc/sysconfig/network-scripts/ifcfg-xenbr0                      DEVICE=xenbr0                      NM\_CONTROLLED=no                      TYPE=Bridge                      BOOTPROTO=none                      IPADDR=192.168.1.10                      NETMASK=255.255.255.0                      GATEWAY=192.168.1.1                      DNS1=8.8.8.8                      ONBOOT=yes                      USERCTL=no                      DELAY=0            service  NetworkManager  stop          chkconfig  NetworkManager  off      #NetworkManager会导致桥创建失败.          service  network  restart |

 (6) 为busybox测试系统提供网卡驱动

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | cd  /mnt/lib/modules/`uname -r`/kernel/drivers/net/        modinfo   xen-netfront.ko       #PV模式下,DomU应该使用网卡的前半段;因此这个驱动需要.                                        # 查看xen-netfront.ko 有没有依赖其他模块,即"depends"段是否有依赖模块.        modinfo  8139too.ko        modinfo  mii.ko                      #8139依赖与mii模块.        mkdir  -pv  /mnt/lib/modules        cp  xen-netfront.ko  8139too.ko  mii.ko   /mnt/lib/modules/        umount  /mnt                         #卸载busybox的映像文件. |

     (7) 创建启动DomU的配置文件:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | ls  /etc/xen          xl.conf             #此配置文件为xl命令的全局通用配置文件.          xlexample.hvm       #这是一个HVM模式运行的DomU的示例配置文件.          xlexample.pvlinux   #这是PV模式运行的DomU配置文件示例.            cp   xlexample.pvlinux   busybox          vim  busybox              【                 name="busybox-001"                 kernel="/boot/vmlinuz"     #注:这是Dom0的/boot下的vmlinuz,此为DomU启动时的Kernel文件.                 ramdisk="/boot/initramfs.img"      #它也是Dom0上的initramfs.img.                 extra="selinux=0  init=/bin/sh"    #指定传递给kernel的额外参数,关闭selinux,系统启动后第一个执行的程序为/bin/sh                 root="/dev/xvda  ro"  #指定根文件系统的位置; 这里需要注意: 因为busybox.img创建后,并没有分区,就直接被格式化了.                                       #    因此就一个分区,不存在其它分区,所以这里直接写成/dev/xvda而不是/dev/xvda1.                                       #    另外 系统在启动初期是以只读方式挂载的.这样需要注意.                       memory=256      #指定内存大小为256M                       #maxmem = 512   #若启用最大内存参数,则可以在DomU运行时,通过以下命令来动态调整DomU的内存.                                       #   xl  mem-max   和  xl  mem-set                       vcpus=2                       disk=['/images/xen/busybox.img,raw,xvda,rw']                       vif=[ 'mac=00:16:3e:00:00:01,bridge=xenbr0' ]      # bridge:是关键参数,其它参数都是可选参数。                   】 |

   (8) 启动busybox虚拟机登录后:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | ifconfig  -a                                #是看不到任何接口的.                  insmod  /lib/modules/xen-netfront.ko        #安装网卡接口驱动模块                  ifconfig  -a                  ifconfig  eth0  192.168.1.20  up                  ctrl + ]             # 回到到Dom0,查看网络接口状态.                  Dom0:  ifconfig  -a  #将可以看到一个新接口.vif#.@                                       # "#": 表示DomU的ID ；  @:表示Dom0上创建的桥ID.                  Dom0:  brctl  show   #将看到新接口已经被桥接到Dom0的桥设备上了。 |

  1.2 创建自有Kernel和initramfs文件的busybox  DomU  
       (1) 提供磁盘映像文件【注: Xen的VM对qcow2格式的磁盘可能无法识别,因此,这里建议自行测试,raw格式是没有问题的.】

|  |  |
| --- | --- |
| 1 | qemu-img  create  -f  qcow2  -o  size=5G,preallocation=metadata  /images/xen/busybox3.img |

       (2) 对该映像文件进行分区

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # losetup  ：这是一个将loop设备文件关联到/dev/loop\*的命令,只有关联到/dev/loop设备后,才能对loop设备文件分区格式化.                  losetup  -a     #查看当前已经被关联的loop设备情况.                  losetup  -f     #列出当前可用的第一个空闲loop设备.                                  # 注:ls  /dev/loop\*  可用看到系统中已经存在的loop设备.                  losetup  /dev/loop0    /images/xen/busybox3.img    #将busybox3.img关联到loop0设备,接着就可以对其做分区格式化了.                                                                     #mount -o loop是先做关联,然后就直接挂载loop设备到挂载点了。                  fdisk  /dev/loop0                  kpartx  -av  /dev/loop0       #刷新loop0的分区信息,并创建分区映射到/dev/mapper中.                  ls  /dev/mapper               #分区刷新完成后,可在这里看到loop0的分区文件                  mkfs.ext3   /dev/mapper/loop0p1                  mkfs.ext3   /dev/mapper/loop0p2     #对loop0p2进行格式化. |

       (3)向 /images/xen/busybox3.img 中boot分区和root分区提供数据文件.:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | # <1> 挂载分区到/mnt                        mkdir  /mnt/{boot, sysroot}                        mount  /dev/mapper/loop0p1  /mnt/boot                        mount  /dev/mapper/loop0p2  /mnt/sysroot                # <2>提供boot分区文件                        cp  /boot/vmlinuz-2.6.\*    /mnt/boot/                        cp  /boot/initramfs-2.6.\*   /mnt/boot/initramfs.img                        #安装bootloader到/dev/loop0,复制启动文件到/mnt/boot/grub。                        grub-install  --root-directory=/mnt   /dev/loop0                        vim  /mnt/boot/grub/grub.conf                          【                                  default=0                                  timeout=3                                  title  Busybox (Linux-2.6.32)                                      root  (hd0,0)                                      kernel  /vmlinuz   root=/dev/xvda1  ro  selinux=0  init=/bin/bash                                      initrd   /initramfs.img                           】                # <3>提供root分区文件                          cp  -a  $BUSYBOX/\_install/\*   /mnt/sysroot/                          mkdir  -pv  /mnt/sysroot/{proc, var, dev, lib/modules, tmp, etc, home, root}                          cp  /lib/modules/`uname -r`/kernel/drivers/net/xen-netfront.ko  /mnt/sysroot/lib/modules/                # <4>手动拆除loop设备                          losetup  -a                 #查看当前已经关联的loop设备                          umount  /mnt/boot                          umount /mnt/sysroot                          kpartx  -d  /dev/loop0      #删除loop0中分区的映射. |

     (4) 提供busybox3的启动配置文件

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | vim  /etc/xen/busybox3                  【                       name="busybox-003"                       # 下面参数都去掉,因为busybox3已经有自己的grub.conf和kernel、initramfs了.                       #kernel="/boot/vmlinuz"     #注:这是Dom0的/boot下的vmlinuz,此为DomU启动时的Kernel文件.                       #ramdisk="/boot/initramfs.img"      #它也是Dom0上的initramfs.img.                       #extra="selinux=0  init=/bin/sh"    #指定传递给kernel的额外参数,关闭selinux,系统启动后第一个执行的程序为/bin/sh                       #root="/dev/xvda  ro"     #指定根文件系统的位置; 这里需要注意: 因为busybox.img创建后,并没有分区,就直接被格式化了.                                                 #    因此就一个分区,不存在其它分区,所以这里直接写成/dev/xvda而不是/dev/xvda1.                                                 #    另外 系统在启动初期是以只读方式挂载的.这样需要注意.                       memory=256         #指定内存大小为256M                       #maxmem = 512                       vcpus=2                       disk=['/images/xen/busybox3.img,qcow2,xvda,rw']                       vif=[ 'mac=00:16:3e:00:00:02,bridge=xenbr0' ] # bridge:是关键参数,其它参数都是可选参数。                       bootloader="/usr/bin/pygrub"      #此pygrub是用python研发的工具,它可以向DomU提供启动引导,                                                         #并自动找到busybox3.img中第一个分区并启动kernel.                       # 注: 关于根文件系统(Root Filesystem):它包含了应用程序、系统组件及配置文件等运行DomU的各种所需文件的                       #     文件系统,其并非必须包含Kernel及initramfs.img,它们完全可以放在Dom0中; 事实上,用于DomU的内核文件必须                       #     要能够允许Dom0访问到,因为其运行时需要与Xen Hypervisor通信,因此这些组件必须位于Dom0能够访问到的                       #     任何文件系统上; 然而目前基于pygrub的方式,允许DomU的内核文件直接放置于DomU的虚拟磁盘映像文件中,                       #     但pygrub实际上只是一种让Dom0可以访问到DomU虚拟磁盘映像文件中的kernel文件的工具.                   】 |

附件4：

虚拟化面临的重要问题：CPU、RAM、I/O的模拟。  
   CPU模拟:  
       (1) 所有OS设计时都认为Kernel是可以控制所有硬件,并可运行CPU的特权指令,即Kernel运行于CPU的Ring0上。  
       (2) 多个OS无法同时直接运行于硬件层之上,它们必须运行在Hypervisor层(下文称:Host)上；就如同不安装操作系统，就不能安装VMware,没有VMware就无法让虚拟机(下文称:Guest)运行起来一样。那问题来了,若Guest必须运行在CPU的Ring0上,Host运行在哪里？  
       (3) OS设计时它认为自己是可以控制所有硬件资源的,那GuestOS之间不就可以相互影响了吗？Guest1要关机,若  
             它直接执行CPU的特权指令关机,那它应该可以关闭整个物理机器,这不是虚拟化所希望的。

       这些问题给CPU虚拟化带来了诸多问题, 但实际上Host一定是真正能执行CPU的特权指令的,Guest运行起来后,它实际控制的CPU是通过软件模拟的CPU,实际上任何物理硬件都是通过集成电路进行运算,通过微代码向外提供输出结果的接口,只有通过软件模拟出这些接口就可以模拟硬件.  
       BT技术: BinaryTranslation(二进制翻译)技术,它可以让Guest在发起特权指令时,直接将其翻译为Host的系统调用,以便加速Guest的执行性能.BT技术出现是因为Guest用户空间的APP要进行特权操作时,Guest上的APP需要先将请求发给Guest的Kernel,Guest的Kernel经翻译转换发给模拟CPU,模拟CPU在将其转换为Host的系统调用,再发给Host的Kernel再进行翻译转换后给物理CPU执行.最后返回,这使得GuestOS的执行性能不高.  
       <> 模拟CPU 和 虚拟CPU  
        》模拟CPU：是完整虚拟,即模拟出CPU的环0,环1,环2,环3；这通常在底层硬件与虚拟环境的硬件不同时采用模拟CPU的方式。  
        》虚拟CPU：仅模拟CPU的环0, Guest的用户空间中的APP可直接运行在物理CPU的环3上,仅环0上的特权指令进行翻译.这是Guest的硬件环境与底层硬件环境一致时用。  
      <> 完全虚拟化 和 半虚拟化  
        》完全虚拟化(Full-Virtulization): 即Guest不知道自己是运行在虚拟化环境中,它一直都认为自己是可以控制全部的硬件资源.  
                   因此需要将GuestOS对特权指令的操作都进行翻译后，由Host带为执行。  
                   VMware的BT技术 和 AMD CPU的AMD-v、Intel的VT-x这两种HVM(硬件虚拟化)技术实际上都是完全虚拟化，  
                   因为它们都是帮助Host高效完成特权指令翻译的技术。  
                   注:AMD-v 和 VT-x实现了将原先只有4环的CPU扩展为5环,并让Host运行在-1环上,腾出环0给Guest用,这样Guest就  
                      认为自己是运行在环0上,并且可直接执行特权指令,但实际上,Guest调用环0上的特权指令时,CPU会直接将其翻译为  
                      真实的特权指令并激活Host的内核来调用环-1来执行特权指令，这进一步缩短了翻译流程。  
   Memory虚拟化:  
       (1) 在物理机中,内存的使用经过虚拟化后，提供给物理机上运行的APP的。  
       (2) 在物理机中,APP看到的内存是：虚拟线性地址空间,即APP认为自己使用的是全部的物理内存,从0-1024(假设内存为1G)  
                               内核看到的内存是：真实物理地址空间  
       (3) 由于物理机的内存已经被虚拟化过了, Guest访问物理内存就需要再次被虚拟一层。  
               注:TLB: 转换后页面缓存   
           Host上的APP访问内存的流程：  
               APP–>发送访问线性地址:10page的数据–>CPU–>将10page的线性地址转给MMU–>  
                         查询线性与物理地址映射表–>缓存映射关系到TLB,并读取物理地址中的数据–>返回。  
            Guest上的APP访问内存的流程：  
                **Shadow Page Table的方式：**  
                       注: 假设将Guest的APP访问的虚拟线程地址称为:GVA.  
                           将Guest虚拟机所获得的虚拟物理内存地址称为：GPA  
                       Guest-APP–>发送访问线性地址(GVA):10page的数据–>虚拟CPU–>将其转给虚拟MMU–>  
                                   查询并缓存映射到虚拟TLB–>GPA的访问请求被发给Host–>  
                                   但GPA并非Host的虚拟线性地址，又非真实物理地址,因此无法由真实CPU处理–>  
                                   Host上不得不采用软件模拟来完成将GPA转换为真实物理内存的物理地址，  
                                   此方式叫 Shadow page table(影子页表),并最终获得物理内存中的数据，返回给Guest.  
               **Intel的EPT(Extended Page Table:扩展页表) 和 AMD的NPT(Nested Page Table:嵌入页表)技术**  
                    》MMU虚拟化 和 TLB虚拟化  
                        注: TLB虚拟化: 由于GuestA和GuestB都被分配了1G的内存,GuestA和GuestB将具有相同的内存地址,  
                                  但GuestA和GuestB它们实际映射到Host上的物理内存的地址段肯定是不同的。但GuestA  
                                  访问它的10page的数据若被缓存在TLB中，GuestB到CPU执行时,它恰巧也要访问它的10page的数据  
                                  那TLB中缓存的条目肯定会对GuestB造成干扰，导致访问错误，因此只能将TLB清空,即GuestB到CPU  
                                  上执行时,TLB是空的,因此CPU只能再通过MMU查映射关系,缓存TLB再访问数据。这导致TLB成了摆设  
                                  没有起到提高访问物理内存效率的作用。这才出现了TLB的虚拟化, 而TLB的虚拟化将原先的只有两个  
                                  字段缓存的TLB改成缓存三个字段: 一个GuestOS标识, 一个GVA，一个HPA(Host中真实物理地址)。  
               Guest-APP–>发生访问线性地址(GVA): 10page的数据—>  
                            这时访问请求被同时发送了两份,并同时分别走了以下路线  
                              路线1–>虚拟CPU–>虚拟MMU–>虚拟TLB–>完成.  
                              路线2–>EPT/NPT硬件实现的MMU–>EPT/NPT实现的TLB–>获取真实物理地址中的数据–>返回给Guest.  
                              这样做后，即没有改变GuestOS访问内存的过程，同时提高了GuestOS访问内存的效率。  
                                          
   I/O虚拟化：  
    (1) 外存设备虚拟化  
               磁盘、光驱、U盘  
    (2) 网络设备虚拟化  
               网卡  
    (3) 输入设备 ： 它的虚拟化采用角点当前被谁捕获来模拟的,若当前被虚拟机捕获,则会动态将模拟键盘或鼠标与物理设备关联.  
               ps/2、USB  
    (4) 显示设备 : 它不能被模拟。  
               VGA  
                 
I/O虚拟化的三种技术:【注: 这三种技术只针对存储和网络，其I/O设备的模拟不采用这些技术。】  
          》模拟 ： 如VMware那一个文件做为VM的磁盘,这就是模拟。  
          》半虚拟化：  
              模拟的过程：Guest-APP去访问I/O设备,如网卡,Guest-Kernel需要先调用虚拟网卡的驱动–>虚拟网卡驱动将数据  
                   转发给虚拟网卡–>虚拟网卡再转给Hypervisor上模拟的网卡进程–>该网卡进程再将数据放入IO Stack(IO栈)中  
                   等待真实网卡驱动将其转给物理网卡并由其将数据转发出去。  
              半虚拟化的过程：Guest-APP去访问I/O设备,如网卡,但此时GuestOS明确知道自己不能直接访问到物理网卡,因此,  
                   直接将数据转给前端设备(它类似一个转发器,但对GuestOS中的用户看来它是一个网卡设备)–>前端设备直接与  
                   Hypervisor上的网卡进程通信–>网卡进程将数据转到IO Stack中–>物理机驱动将其转发给物理网卡。  
                   注: 前端设备(IO frontend)：在Guest端省去了Guest虚拟网卡驱动的步骤。  
                       后端设备(IO backend)：在Hypervisor端从虚拟网卡进程到物理网卡称为后端设备。  
          》I/O透传：  
                   假如规划中该主机上需要运行4台虚拟机,该主机上安装了6块盘,6块网卡,物理机磁盘和网卡都使用两个，  
              剩余的都给虚拟机使用,那完全可以给每个虚拟机分一个物理磁盘和一个物理网卡,这样其性能可几乎接近于硬件  
              访问，但这需要Hypervisor来中的设备管理器来分配。  
                
              x86平台上DMA(直接内存访问)是集中共享式管理的  
                   假如当前Hypervisor管理着4块网卡,现在GuestA和GuestB要发生数据到网卡上,  
              GuestA和GuestB都需要借助DMA来代其完成,但DMA是共享式管理,它仅仅负责接收  
              Kernel发来的命令然后,根据指示将指定内存地址中的数据发送到指定IO设备,如网卡,  
              或将网卡上收到的数据存入到指定的内存地址中。问题是Guest通过复杂的kernel调用  
              过程完成了DMA代其发送数据包到网卡,但GuestOS并不知道DMA将数据包发到那个网卡  
              上了，更没有人知道回应的包到网卡后,应该转发给那个GuestOS.  
                   注: DMA:它的作用是帮助Kernel完成一些需要长时间等待的任务,如：写数据到磁盘,  
                              从磁盘中读数据到内存,将网卡上收到的数据读到内存等; 这对提高Kernel的执行  
                              效率是非常有帮助的。  
                
             IO设备：  
                   每个IO设备上都有控制器,并且每个IO设备的控制器上都有寄存器且都有相应的端口地址。  
             IOMMU: IO内存地址管理单元  
                   它的作用是自动将IO总线与相应的IO端口关联的设备。  
                   若需要将某块网卡绑定给某GuestOS单独使用,就需要在Hypervisor级别来控制对I/O端口的调用  
             就只能接受该GuestOS的调用了，而它的实现就需要借助DMA中实现IOMMU来完成。而Intel的VT-d  
             就是这样一种技术，它在DMA中实现了IOMMU,并自动来完成IO总线与IO端口的映射关联。  
                   IOMMU的映射是完成将物理IO设备绑定给GuestOS的第一步,接着还需完成将物理IO设备的中断信号  
             也要映射给该GuestOS,要知道物理设备在实现中断映射通常采用中断控制器 或 DMA等方式实现,而采用DMA方式  
             则需要DMA必须能够访问全部的物理内存,因为DMA需要将IO设备的寄存器中的数据搬到指定的内存地址空间  
             或反过来；而若将物理硬件绑定给GuestOS,则DMA就需要访问GuestOS的全部物理内存,但实际上GuestOS  
             的内存是虚拟的,这就需要经过复杂的转换后才能访问到其对应的物理内存地址空间，这也进一步加大了  
             中断信号与GuestOS的映射；而这还是完成虚拟机绑定硬件的第二步。最后一步是完成IO设备缓冲区与GuestOS  
             的映射,这里必须知道缓冲区一定是物理内存中的一段空间,如何让虚拟机的内核知道这是自己所管理的物理IO设备的  
             IO缓冲区？；而这一切Intel的VT-d都帮我们实现了。  
               
             Intel的VT-d  
                   简单来说 VT-d 是一种基于北桥的硬件辅助虚拟化技术。  
                     
                     
   Intel的硬件辅助虚拟化技术：  
          CPU: vt-x， EPT， tagged-TLB  
          IO： vt-d，SR-IOV:单根设备虚拟化协议， VMDq  
          这些技术详细分为三类：  
               第一类: 跟处理器相关: VT-x, EPT, tagged-TLB  
               第二类: 跟芯片相关: vt-d  
               第三类: 跟IO输入输出相关: VMDq 和 SR-IOV  
            
          IO虚拟化技术： QEMU(法国天才程序员开发) 、virtio(澳大利亚天才程序员开发)  
          KVM和Xen都需要借助以上两种虚拟化技术实现IO虚拟化。  
            
   虚拟化技术分类：  
       (1) 模拟：【特点:虚拟硬件环境与底层硬件环境无关.】  
               著名的模拟器: PearPC,  Bochs, QEMU  
       (2) 完全虚拟化: 【特点: 虚拟硬件环境与底层硬件环境必须一致。】  
               在完全虚拟化中通常采用两种加速技术:  
                       BT : 二进制翻译加速技术  
                       HVM: 基于硬件的虚拟化  
               知名的产品：VMware Workstation, VMware Server, Parallels Disktop(iMac上用), KVM, Xen(HVM)  
       (3) 半虚拟化：【同样要求虚拟硬件环境与底层硬件环境必须保持一致】  
               知名的开源产品: xen, uml(user-mode  linux)  
       (4) OS级别的虚拟化:  
               OpenVZ,  LXC(LinuX Container), libcontainer, Virtuozzo, Linux V Servers  
               Solaris  Containers  
               FressBSD  jails  
       (5) 库虚拟化:  
               wine