虚拟化技术的分类

虚拟化技术主要分为以下几个大类 [1]：

平台虚拟化（Platform Virtualization），针对计算机和操作系统的虚拟化。

资源虚拟化（Resource Virtualization），针对特定的系统资源的虚拟化，比如内存、存储、网络资源等。

应用程序虚拟化（Application Virtualization），包括仿真、模拟、解释技术等。

我们通常所说的虚拟化主要是指平台虚拟化技术，通过使用控制程序（Control Program，也被称为 **Virtual Machine Monitor** 或 **Hypervisor）**，隐藏特定计算平台的实际物理特性，为用户提供抽象的、统一的、模拟的计算环境（称为**虚拟机**）。虚拟机中运行的操作系统被称为客户机操作系统（Guest OS），运行虚拟机监控器的操作系统被称为主机操作系统（Host OS）

平台虚拟化技术又可以细分为如下几个子类：

全虚拟化（Full Virtualization）

全虚拟化是指虚拟机模拟了完整的底层硬件，包括处理器、物理内存、时钟、外设等，使得为原始硬件设计的操作系统或其它系统软件完全不做任何修改就可以在虚拟机中运行。操作系统与真实硬件之间的交互可以看成是通过一个预先规定的硬件接口进行的。全虚拟化 VMM 以完整模拟硬件的方式提供全部接口（同时还必须模拟特权指令的执行过程）。举例而言，x86 体系结构中，对于操作系统切换进程页表的操作，真实硬件通过提供一个特权 CR3 寄存器来实现该接口，操作系统只需执行 "mov pgtable,%%cr3" 汇编指令即可。全虚拟化 VMM 必须完整地模拟该接口执行的全过程。如果硬件不提供虚拟化的特殊支持，那么这个模拟过程将会十分复杂：一般而言，VMM 必须运行在最高优先级来完全控制主机系统，而 Guest OS 需要降级运行，从而不能执行特权操作。当 Guest OS 执行前面的特权汇编指令时，主机系统产生异常（General Protection Exception），执行控制权重新从 Guest OS 转到 VMM 手中。VMM 事先分配一个变量作为影子 CR3 寄存器给 Guest OS，将 pgtable 代表的客户机物理地址（Guest Physical Address）填入影子 CR3 寄存器，然后 VMM 还需要 pgtable 翻译成主机物理地址（Host Physical Address）并填入物理 CR3 寄存器，最后返回到 Guest OS中。随后 VMM 还将处理复杂的 Guest OS 缺页异常（Page Fault）。比较著名的全虚拟化 VMM 有 Microsoft Virtual PC、VMware Workstation、Sun Virtual Box、Parallels Desktop for Mac 和 QEMU。

超虚拟化（Paravirtualization）

这是一种修改 Guest OS 部分访问特权状态的代码以便直接与 VMM 交互的技术。在超虚拟化虚拟机中，部分硬件接口以软件的形式提供给客户机操作系统，这可以通过 Hypercall（VMM 提供给 Guest OS 的直接调用，与系统调用类似）的方式来提供。例如，Guest OS 把切换页表的代码修改为调用 Hypercall 来直接完成修改影子 CR3 寄存器和翻译地址的工作。由于不需要产生额外的异常和模拟部分硬件执行流程，超虚拟化可以大幅度提高性能，比较著名的 VMM 有 Denali、Xen。

硬件辅助虚拟化（Hardware-Assisted Virtualization）

硬件辅助虚拟化是指借助硬件（主要是主机处理器）的支持来实现高效的全虚拟化。例如有了 Intel-VT 技术的支持，Guest OS 和 VMM 的执行环境自动地完全隔离开来，Guest OS 有自己的“全套寄存器”，可以直接运行在最高级别。因此在上面的例子中，Guest OS 能够执行修改页表的汇编指令。Intel-VT 和 AMD-V 是目前 x86 体系结构上可用的两种硬件辅助虚拟化技术。

内存虚拟化

计算机的内存管理使用的是页式管理，应用程序使用到的内存地址是虚拟地址，

虚拟地址需要经过内存管理单元(MMU) 的翻译转换成物理地址才能真正访问内存。其

中的地址转换用到的是页表，页表保存在内存中，为了降低地址转换所带来的访存数，

在处理器里又添加了转译后备缓冲器(TLB) 加速地址转化。

而在虚拟化环境下，这样一层地址转化是不够的，虚拟机之间感受不到对方的存

在，他们会认为自己独占整个物理内存空间，因此虚拟机的地址翻译可能会将同一块

物理地址分配给不同的虚拟机，这就无法实现虚拟机之间的隔离和数据的安全。为了

实现内存虚拟化，让客户机使用一个隔离的、从零开始且连续的内存空间，虚拟机管理

器会引入一层额外的地址空间，称为客户机物理地址空间(Guest Physical Address,GPA)。

这个地址空间并不是真正的物理地址空间，只是虚拟机虚拟地址空间映射到的物理地

址空间，对于虚拟机来说GPA 是连续且从零开始的，GPA 还需要进一步的地址转换映

射到真正的物理机的物理地址上。GPA 不能用作物理机内存寻址，还需要做一个转化，

在全虚拟化下通常有两种方法能够做到，影子页表和EPT 页表，他们都通过建立GPA

到物理地址映射来实现。

虚拟化技术实现了客户机操作系统(GuestOS)对硬件资源的共享,但是应用程序行为具有很强的可变性,很有可能导致部分虚拟机缺少可用物理内存而部分虚拟机的物理内存资源过剩的情况.如果没有一个有效的物理内存管理以及调度机制,数据中心的内存资源会出现利用率低下或者资源浪费的问题,从而导致性能下降.现在流行的虚拟机管理器(virtual machine monitor,简称 VMM)多采用静态的内存分配方法,在初始化时就限定了内存的大小,当一台虚拟机内存超载时,它不得不将一些物理页面交换到磁盘上去,从而严重降低了系统性能. 为了改善或者避免类似的情况,最合适的方法是让虚拟化环境下的资源管理者、分配者 VMM 能够为多台虚拟 机进行内存调控.一些现有的技术,如气球技术(ballooning)[1]、远程内存[2]、虚拟机迁移[3]已经广泛地应用于动 态均衡多个虚拟机或者物理机的负载.然而以上的方法都只是提供了调控内存的底层功能,并未提出完整的多 机内存调度方案.为了能对一个数据中心的内存进行全局的调控及优化,我们必须要有一个机制来准确地预测 不同虚拟机的内存需求,并进一步进行内存的动态调度.

设计与实现

我们采用的是全虚拟化下影子页表的内存虚拟化方式，并且通过给页表项置位来截获页面，所以在设计与实现中，首先要实现的是内存的截获。在影子页表模式下，虚拟机在进行地址翻译的时候首先会通过CR3寄存器获得页表的起始位置，但是实际读取到的是影子页表的起始位置CR3而非虚拟机页表起始位置，这是因为写CR3寄存器是特权指令，当虚拟机写CR3的时候会被虚拟器管理器截获，虚拟机管理器写入影子页表起始位置。

我们知道影子页表模式下虚拟机地址翻译用到的是影子页表，但是影子页表是如何和虚拟机页表同步的呢？特别是在进程切换的时候，如何同步到对应的新进程的影子页表上去？不同于普通页表是每个进程都有一份，虚拟机管理器只维护一份影子页表。所以在进程切换的时候影子页表存在同步问题

由于进程切换是非常频繁的，每次进程切换都删除原来影子页表重建影子页表开销太大，其实进程切换之后并不会用完原本所有的影子页表项，新的进程可能只会使用到一小部分影子页表，方法二的同步方法由于只是给页表项加标记，标记为不存在，当新进程使用页面会因为页面不存在而陷入虚拟机管理器，这个时候同步影子页表项，所以只是新的进程使用到的页面才会更新，当进程切换回原来的进程的时候，没有更改过的页表项还可以正常使用。

还有一个需要同步的地方是页面的access位和dirty位，如果页面一开始是只读没有写的，那么当虚拟机更改页面的时候，影子页表无法同步dirty位，所以在进程切换的时候还会给所有页面加入写保护，一个页面一开始是只读，那么会因为页表项不存在而失效，当页面被写入的时候会因为写保护而再次失效，影子页表同步dirty位

AET原理

缓存系统使用的替换算法通常是LRU 算法，根据LRU 算法，无论缓存是如何组

织的，缓存命中或者是缓存失效都会导致缓存块的移动。AET 模型和缓存块的平均淘

汰时间相关，缓存块在被淘汰之前也许会发生多次重用，而淘汰时间是在缓存块最后

一次被访问到至被淘汰的时间。

根据失效率曲线的定义，缓存大小为c时的失效率即为重用时间大于缓存大小c的平均淘汰时间的比例。F(x)为重用时间为x的比例，通过Hash表能够快速获得重用时间，所以f(x)很容易得到，P(t)为重用时间大于等于t的比例，在离散化之后通过f(x)也容易获得。而缓存大小c的计算在离散化之后也很容易获得，当对右边的累加公式不断累积求和到c的时候的t就是缓存大小为c的平均淘汰时间，这个时刻的P(t)就是缓存大小为c的失效率