内存虚拟化

> 本文首发于我的公众号 **Linux云计算网络（id: cloud_dev）** ，专注于干货分享，号内有 **10T** 书籍和视频资源，后台回复
> 取，欢迎大家关注，二维码文末可以扫。

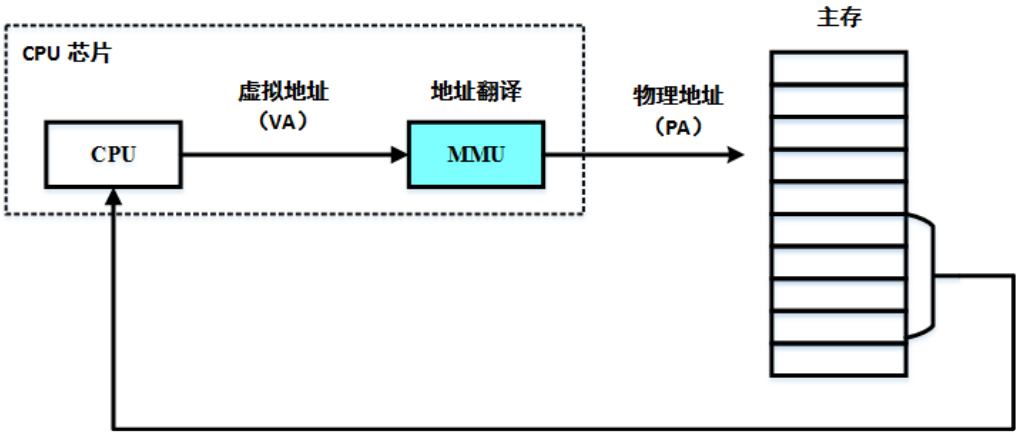
CONTENTS

虚拟内存

我们知道，早期的计算机内存，只有物理内存，而且空间是极其有限的，每个应用或进程在使用内存时都得小心翼翼，不
存区。

为了避免这些问题，就提出了虚拟内存的概念，其抽象了物理内存，相当于对物理内存进行了虚拟化，保证每个进程都被赋予一块连续的，超大的（根据系统结构来定，32 位系统寻址空间为 2^32，64 位系统为 2^64）虚拟内存空间，进程可以毫无顾忌地使用内存，不用担心申请内存会和别的进程冲突，因为底层有机制帮忙处理这种冲突，能够将虚拟地址根据一个页表映射成相应的物理地址。

这种机制正是虚拟化软件做的事，也就是 MMU 内存管理单元。



本文要说的不是这种虚拟内存，而是基于虚拟机的内存虚拟化，它们本质上是一样的，通过对虚拟内存的理解，再去理解内存虚拟化就比较容易了。

结合前面的文章，我们知道，虚拟化分为软件虚拟化和硬件虚拟化，而且遵循 intercept 和 virtualize 的规律。

内存虚拟化也分为基于软件的内存虚拟化和硬件辅助的内存虚拟化，其中，常用的基于软件的内存虚拟化技术为「影子页表」技术，硬件辅助内存虚拟化技术为 Intel 的 EPT（Extend Page Table，扩展页表）技术。

为了讲清楚这两门技术，我们从简易到复杂，循序渐进，逐步揭开其神秘面纱。

常规软件内存虚拟化

虚拟机本质上是 Host 机上的一个进程，按理说应该可以使用 Host 机的虚拟地址空间，但由于在虚拟化模式下，虚拟机处于非 Root 模式，无法直接访问 Root 模式下的 Host 机上的内存。

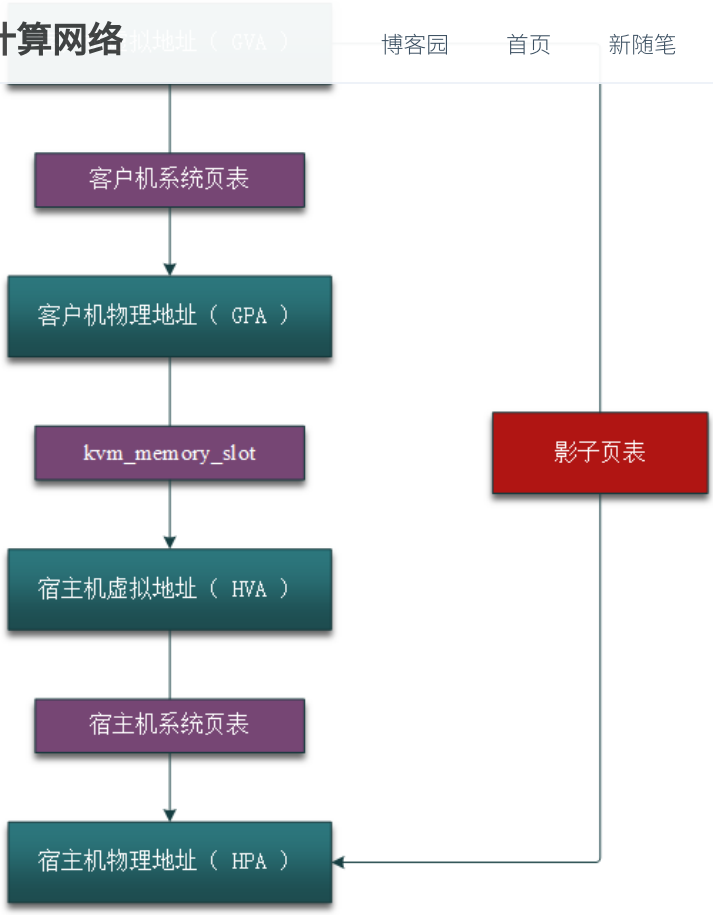
这个时候就需要 VMM 的介入，VMM 需要 intercept （截获）虚拟机的内存访问指令，然后 virtualize（模拟）Host 上的内存，相当于 VMM 在虚拟机的虚拟地址空间和 Host 机的虚拟地址空间中间增加了一层，即虚拟机的物理地址空间，也可以看作是 Qemu 的虚拟地址空间（稍微有点绕，但记住一点，虚拟机是由 Qemu 模拟生成的就比较清楚了）。

所以，内存软件虚拟化的目标就是要将虚拟机的虚拟地址（Guest Virtual Address, GVA）转化为 Host 的物理地址（Host Physical Address, HPA），中间要经过虚拟机的物理地址（Guest Physical Address, GPA）和 Host 虚拟地址（Host Virtual Address）的转化，即：

```
GVA -> GPA -> HVA -> HPA
```

Copy

其中前两步由虚拟机的系统页表完成，中间两步由 VMM 定义的映射表（由数据结构 kvm_memory_slot 记录）完成，它可以将连续的虚拟机物理地址映射成非连续的 Host 机虚拟地址，后面两步则由 Host 机的系统页表完成。如下图所示。



这样做得目的有两个：

1. 提供给虚拟机一个从零开始的连续的物理内存空间。
2. 在各虚拟机之间有效隔离、调度以及共享内存资源。

影子页表技术

接上图，我们可以看到，传统的内存虚拟化方式，虚拟机的每次内存访问都需要 VMM 介入，并由软件进行多次地址转换，其效率是非常低的。因此才有了影子页表技术和 EPT 技术。

影子页表简化了地址转换的过程，实现了 Guest 虚拟地址空间到 Host 物理地址空间的直接映射。

要实现这样的映射，必须为 Guest 的系统页表设计一套对应的影子页表，然后将影子页表装入 Host 的 MMU 中，这样当 Guest 访问 Host 内存时，就可以根据 MMU 中的影子页表映射关系，完成 GVA 到 HPA 的直接映射。而维护这套影子页表的工作则由 VMM 来完成。

由于 Guest 中的每个进程都有自己的虚拟地址空间，这就意味着 VMM 要为 Guest 中的每个进程页表都维护一套对应的影子页表，当 Guest 进程访问内存时，才将该进程的影子页表装入 Host 的 MMU 中，完成地址转换。

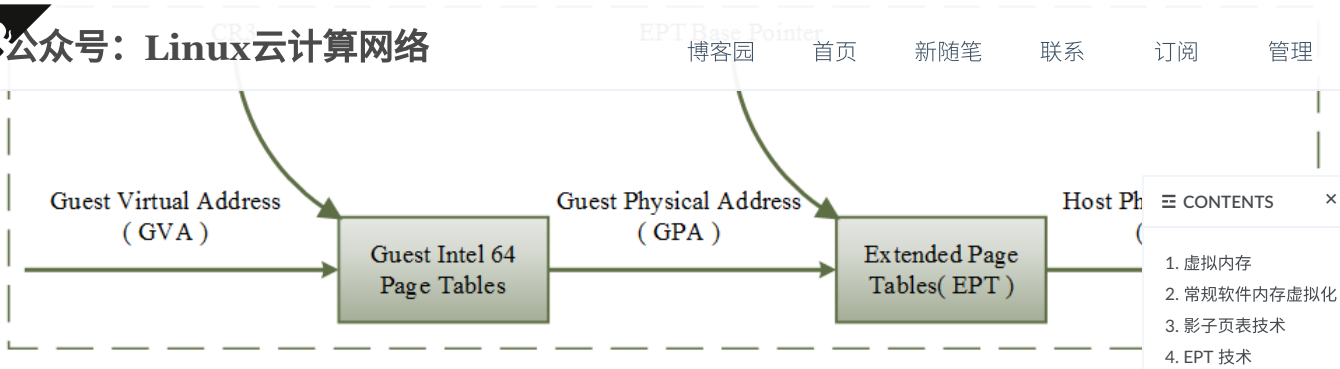
我们也看到，这种方式虽然减少了地址转换的次数，但本质上还是纯软件实现的，效率还是不高，而且 VMM 承担了太多影子页表的维护工作，设计不好。

为了改善这个问题，就提出了基于硬件的内存虚拟化方式，将这些繁琐的工作都交给硬件来完成，从而大大提高了效率。

EPT 技术

这方面 Intel 和 AMD 走在了最前面，Intel 的 EPT 和 AMD 的 NPT 是硬件辅助内存虚拟化的代表，两者在原理上类似，本文重点介绍一下 EPT 技术。

如下图是 EPT 的基本原理图示，EPT 在原有 CR3 页表地址映射的基础上，引入了 EPT 页表来实现另一层映射，这样，GVA->GPA->HPA 的两次地址转换都由硬件来完成。



这里举一个小例子来说明整个地址转换的过程。假设现在 Guest 中某个进程需要访问内存，CPU 首先会访问 Guest 中的 CR3 页表来完成 GVA 到 GPA 的转换，如果 GPA 不为空，则 CPU 接着通过 EPT 页表来实现 GPA 到 HPA 的转换（实际上，CPU 会首先查看硬件 EPT TLB 或者缓存，如果没有对应的转换，才会进一步查看 EPT 页表），如果 HPA 为空呢，则 CPU 会抛出 EPT Violation 异常由 VMM 来处理。

如果 GPA 地址为空，即缺页，则 CPU 产生缺页异常，注意，这里，如果是软件实现的方式，则会产生 VM-exit，但是硬件实现方式，并不会发生 VM-exit，而是按照一般的缺页中断处理，这种情况下，也就是交给 Guest 内核的中断处理程序处理。

在中断处理程序中会产生 EXIT_REASON_EPT_VIOLATION，Guest 退出，VMM 截获到该异常后，分配物理地址并建立 GVA 到 HPA 的映射，并保存到 EPT 中，这样在下次访问的时候就可以完成从 GVA 到 HPA 的转换了。

总结

内存虚拟化经历从虚拟内存，到传统软件辅助虚拟化，影子页表，再到硬件辅助虚拟化，EPT 技术的进化，效率越来越高。

---

stay hungry stay foolish ----jobs 希望多多烧香！

分类: 云计算, 虚拟化
标签: 虚拟化, 云计算

« 上一篇： CPU 虚拟化
» 下一篇： 网络虚拟化

posted @ 2017-12-04 12:07 CloudDeveloper 阅读(8939) 评论(0) 收藏

CONTENTS ×