Hypervisor 中内存回收技术的改进①

吴 岳

(国家林业局 林产工业规划设计院, 北京 100010)

摘 要: 虚拟化是云计算的关键技术. Hypervisor 在虚拟机与主机硬件之间提供了一个抽象层, 允许用户为运行着的虚拟机分配的内存总值超过主机的可用内存, 这种技术称为内存过量分配. 为了能够降低这个技术对虚拟机性能的影响, hypervisor 必须提供高效率的内存回收机制. 在本论文中, 作者提出了一种解决方案: 使用非易失性内存作为 hypervisor 交换页面数据的缓存设备. 作者从系统内存中划分出空间模拟了非易失性内存设备, 修改了 KVM 模块中的算法, 并制定了五种测试环境. 通过实验数据证明, 相比现有的 Ballooning 技术与 Hypervisor swapping 技术, 使用非易失性内存并配合低优先级队列算法时, 虚拟机性能可提高 30%和 50%左右.

关键词: 虚拟化; Hypervisor; 非易失性存储器; 内存回收

Improvement of Memory Reclaim Technology in Hypervisor

WU Yue

(Planning and Design Institute of Forest Products Industry, State Forestry Administation, Beijing 100010, China)

Abstract: Virtualization is the key technology of cloud computing. Hypervisor provides an abstraction layer between the virtual machine and the host hardware, which allows the user to allocate memory for running the virtual machines over the available memory of a host. This technique is called memory overcommitment. In order to reduce the impact of this technology on the performance of the virtual machines, hypervisor must provide an efficient memory reclamation mechanism. In this paper, the author proposes a solution: using the nonvolatile memory as a cache device for hypervisor to exchange page file data. The author divides the space from the system memory to simulate the nonvolatile memory devices, and modifies the algorithm in the KVM module, and sets five kinds of testing environments. The experiments show that compared with the existing Ballooning technology and Hypervisor swapping technology, using the nonvolatile memory and with low priority queue algorithm, the virtual machine performance can be improved by 30% and 50%.

Key words: virtualization; Hypervisor; nonvolatile memory; memory reclamation

服务器虚拟化软件可以一台物理服务器(主机)上运行多个虚拟机(VM), 这些虚拟机各自运行不同的操作系统, 彼此独立却共同分享主机的物理资源. 这种直接在物理硬件上运行的服务器虚拟化软件被称为虚拟机监控器(hypervisor), 它为虚拟机分配并管理物理主机的资源, 如 CPU、内存、网络与存储等.

在高级的操作系统中, 进程对连续的地址空间进行寻址时不必访问底层的物理内存. 操作系统维护着一个分页表, 记录了虚拟分页文件与物理页面的映射. 目前所有的 CPU 都包含内存管理单元(MMU)与分页缓存(TLB)来优化虚拟内存的性能.

在单个主机系统上运行多个虚拟机时, 需要更加复杂的内存虚拟化技术. 为了支持多个客户机操作系统, 内存管理单元被虚拟化. 每个客户机操作系统控制着客户机虚拟内存地址与客户机物理内存地址的映射, 但是客户机操作系统不能直接访问主机的物理内存. hypervisor 通过影子页表(Shadow Page Table)管理虚拟机物理内存与主机物理内存的映射. hypervisor 使用分页缓存来加速虚拟机虚拟内存与主机物理内存间的转换, 而不是频繁进行虚拟机虚拟地址-虚拟机物理

① 收稿时间:2016-01-10;收到修改稿时间:2016-02-25 [doi: 10.15888/j.cnki.csa.005331]

地址-主机物理地址间的两级变换. 当虚拟机操作系统更改了它的虚拟内存到物理内存的映射时, hypervisor 更新影子页表来确保直接寻址[1].

1 hypervisor中的内存回收技术现状

虚拟机监视器支持物理内存过量分配(over-commitment)技术, 即所有正在运行的虚拟机内存总数可以超过主机物理内存总数, 因此可以在一台主机上运行更多的虚拟机. 为了有效地支持这个技术, hypervisor 必须提供高效率的内存回收技术.

目前的 hypervisor 大多支持以下内存回收技术:

Transparent Page Sharing: 当多个虚拟机正在运行, 它们可能会有相同的内存内容. 例如, 几个虚拟机可能运行相同的操作系统, 具有相同的应用程序或包含相同的用户数据. hypervisor 可以回收冗余副本, 只保留一组数据, 从而释放主机内存.

Ballooning: 该技术需要在虚机上安装特殊的气球驱动. hypervisor 通过驱动从虚拟机回收未使用的物理内存. 由于气球驱动安装在虚拟机操作系统内, 它可以理解虚拟机操作系统的内存需求. 如果主机操作系统缺少内存, 它会轮询其上所有的气球驱动来请求可用内存, 然后从虚拟机抽取资源. 借助虚拟机上的气球驱动,它不会抢夺虚拟机内活动状态的有效内存资源.

Hypervisor swapping: hypervisor 回收使用中的内存页面, 并把数据存储到文件交换区. 由于磁盘的存储速度要比物理内存慢得多, 所以交换的方式会极大降低虚拟机的性能[2].

随着科技的发展, 如 phase-change-memory、spin-transfer-torque RAM 等非易失性内存可以达到接近 DRAM 的读写速度, 数据可以通过读取/存储指令直接访问而不再需要输入/输出块操作. 本文的研究内容是中内存回收技术的改进, 将部门内存空间模拟为非易失性内存(PM)作为 hypervisor 交换页面数据的缓存, 从而降低内存回收过程中对性能的影响, 更有效地将主机系统内存动态分配给虚拟机.

2 搭建测试环境

实验中主机的操作系统是 Linux kernel 3.7.1, 虚拟化软件是 QEMU-KVM 架构, 虚拟机的操作系统是 Ubuntu 14.04 LTS. KVM(Kernel based virtual machine)是 Linux 的一个模块. 但是 KVM 模块缺乏设备虚拟化以及相应的用户空间管理虚拟机的工具, 所以需要与 QEMU 结合使用[3]. QEMU 是一个开源的模拟器程序, 它的特点是可虚拟不同的 CPU[4].

2.1 内存区域划分

BIOS 通过 E820 表向 Linux 内核传递系统内存映射表. 每个物理地址段与地址类型相关联, 表明了操作系统要如何使用该段物理地址. 为了从内存中模拟出一个非易失性内存, 我们在映射表中增加一个新的地址段, 称为 AddressRangePM, 这样操作系统可以区分出系统内存与模拟的非易失性内存.

在 x86 结构中, 内存被划分为三个区域: 内存最开始的 16MB 称为 ZONE_DMA; 16MB 到 896MB 称为 ZONE_NORMAL, 即 Kernel memory; 896MB 以后称为 ZONE_HIGHMEM, 即 User memory. 系统内存被划分为 4096 字节大小的固定块, 叫作 page frame. 为了模拟非易失性内存, 从 ZONE_HIGHMEN 中分离出新的区域, 称为 ZONE_PM. 设两个全局变量 pmstart_pfn 与 pmend_pfn 用于在 ZONE_HIGHMEM 中初始化新的区域, 地址段 AddressRangePM 对应在区域 ZONE_PM 中.

作者在实验中的一项主要改进是重新划分内存区域. 实验用计算机物理内存为 8GB, 如果使用 2GB 模拟非易失性内存, 那么修改后的内存区域如图 1 所示: ZONE_DMA(最开始的 16MB)、ZONE_NORMAL(16MB 到 896MB)、ZONE_HIGHMEM(896MB 到 5956MB) 和 ZONE_PM(5956MB 以后).

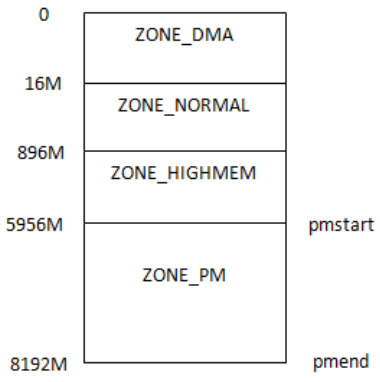

图 1 划分后的内存区域映射

2.2 Qemu-Kvm 结构扩展

作者在实验中的另一项主要改进是修改 KVM,

才能从内存映射表中ZONE_PM区域中交换页面文件. 目的是当内存出现压力时, 页面文件被交换到ZONE_PM, 当内存负载正常时, 页面文件回到内存中.

由于页面替换策略不会按照进程不同区别对待, 属于任何进程的任何页面都有可能被回收. 所以作者通过引入低优先级队列来跟踪优先级较低的 Qeum 进程的页面分配, 然后使用 LRU(Least Recently Used)算法回收这些进程的页面文件[5].

当系统启动时, kswapd_init()函数启动了称为 kswapd 的内核线程, 这个线程会连续执行 kswapd()函数. 函数代码如下:

```
static int kswapd(void p) {
    if(!zone−>balanced) {
    move_inactive_pages_to_pm(&zone);
    }
}
void move_inactive_pages_to_pm (struct zone_t zone) {
    struct page_t page;
    spin_lock_irq(zone−>lpri_lock);
    while( size ( low_pri_queue)>0 && pages_freed) {
    page = dequeue (low_pri_queue);
    if(page−>is_locked() && page−>PG_launder) {
    wait_on_page (page−>page_id);
    continue;
    }
    if( page−>PG_dirty ) {
    page−>PG_launder = 1;
    page_PG_dirty = 0;
    writepage ( page−>page_id );
    }
    copy_to_pm ( page );
    mmu_notifier_modify_pte ( );
    spin_lock_irq ( zone−>lru_lock );
    refill_inactive_zone ( page );
    spin_unlock_irq ( zone−>lru_lock );
    pages_freed;
    }
    spin_unlock_irq (zone−>lpri_lock);
    }
```

当区域中空闲页面的数量达到 pages_low 时, kswapd 线程被物理页面分配器唤醒. 它释放一定数量的页面并重置直到下一次被唤醒. 这个过程会持续直到区域中的空闲页面数量达到 pages_high. kswapd 通过访问全局结构 low_pri_queue 发挥如下功能:

① 如果页面被锁定, 并且 PG_launder 被设置, 那么说明这个页面被进程的 I/O 操作锁定. kswapd 在这个区域标记 wait_queue, 并下一次操作时移动到 ZONE_PM.

② 如果页面失效, 并且 PG_dirty 被设置, kswapd 立刻刷新页面内容, 并标记为 writepage()函数可用.

③ 如果页面没有失效或者没有进程使用, 页面被移动到 ZONE_PM.

当系统出现低内存状态时, kswapd 首先检查迁移列表, 列表中记录了所有准备迁往 ZONE_PM 区域的低优先级页面. 在之后的迭代操作中, kswapd 迁移系统所有 QEUM 进程中的低优先级页面到 ZONE_PM, 并释放内存中的这些页面来确保高优先级进程的消耗, 直到迁移列表为空. 当系统回复正常内存状态时, kswapd 检查可用页面的数量是否等于高优先级页面的数量, 如果是的话, 将 ZONE_PM 中的页面迁回内存, 并更新页面表.

3 实验设计

在实验中, 作者配置了多个虚拟机操作系统用来运行测试, 通过KVM控制组(cgroups), 可以限制为某个虚拟机操作系统分配的物理资源数量, 从而维持虚拟机们的优先级顺序. 测试使用 Linux kernel compile(LKC)与 Parallel bzip2(Pbzip2)作为宏观评测工具, 在每组测试中运行 5 次不同操作取平均值.

实验中一共使用了5种测试环境, 分别使用S0-S4表示.

S0: 使用 ballooning 技术的基准值.

配置这个环境的目的是建立使用 ballooning 技术作为内存回收机制时的基准值. ballooning 技术是 Qemu-KVM、VmWare-ESX 与 Xen 中常用的内存回收技术. 在这个环境下, 每个虚拟机操作系统中加载 ballooning 驱动.

S1: 使用 hypervisor swap 技术交换页面到磁盘.

配置这个环境的目的是建立使用 hypervisor swap 技术作为内存回收机制时的基准值. hypervisor swap 技术是 Qemu-KVM、VmWare-ESX 与 Xen 中最后使用

的内存回收技术. 在这个环境下, 每个虚拟机操作系统禁用 ballooning 驱动. 当系统内存低时, hypervisor 将不活动的页面文件交换到硬盘.

S2: 使用 hypervisor swap 技术交换页面到 ZONE_PM.

这个测试环境与 S1 的区别是, 当系统内存低时, hypervisor 将不活动的页面文件交换到内存中的 ZONE_PM 中, 而不是 S1 中的硬盘.

S3: 使用 hypervisor swap 技术交换低优先级页面到 ZONE_PM.

这个测试环境与 S2 的区别是, 使用 cgroups 为多个虚拟机操作系统配置优先级. 当系统内存低时, hypervisor 将低优先级的虚拟机操作系统的不活动的页面文件交换到 ZONE_PM 中.

S4: 使用 hypervisor swap 技术交换低优先级队列页面到 ZONE_PM.

这个测试环境与 S3 的区别是, 当系统内存低时, hypervisor 将全部低优先级队列中的虚拟机操作系统的不活动的页面文件交换到 ZONE_PM 中.

4 实验结果

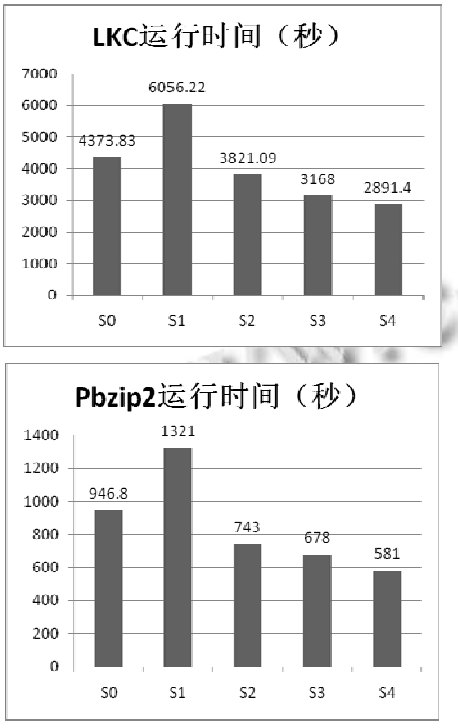

图 2 实验结果

实验结果如图 2 所示, S0 的实验结果很明显优于 S1, 主要因为 hypervisor 没有任何监测虚拟机操作系统分页表的方法, 它使用 LRU 算法从主机的分页表中找出最适合下一次交换到磁盘的页面文件, 而这个过程会降低系统的性能[6].

LKC 操作的内容是编译 linux 内核, 需要从源文件中读取整个目录到内存中. 测试结果显示, S1 的性能最差, 因为 hypervisor 层面的页面交换影响了虚拟机的性能[7]. 在 S2 启用 ZONE_PM 时, 性能比 S0 提高了 12.64%, 比 S1 提高了 36.9%. 尤其是当 S4 中使用了低优先级队列算法时, 性能比 S0 提高了 33%, 比 S1 提高了 50%以上.

Pbzip2 操作的内容是对全部 linux 源文件目录树进行多线程的压缩操作. S4 条件下的性能依然最优, 比 S0 提高了 38.3%, 比 S1 提高了 56%.

5 结语

在现有的虚拟机内存回收技术中, Ballooning, Transparent Page Share 与 Hypervisor swap 都会影响运行速度. 随着数据中心中 Hypervisor 数量的增加, 需要有高效的内存回收技术来保障虚拟机的性能. 非易失性内存设备具有断电不丢失数据、读写延迟接近内存的优点. 通过实验证明了使用这种内存设备, 可以作为 hypervisor 交换数据缓存的理想选择.

参考文献

1 刘典型.多虚拟机下基于内存缓存的动态块迁移算法.计算机应用与软件,2015,3:11–15.
2 张龙.一种高效的虚拟机磁盘快照系统.电脑编程技巧与维护,2015,10:93–95.
3 刘国乐,何建波,李瑜.Xen 与 KVM 虚拟化技术原理及安全风险.保密科学技术,2015,4:24–26.
4 李雪竹,陈国龙.云计算虚拟化平台的内存资源全局优化研究.计算机工程,2015,7:55–59.
5 袁野,赵海燕,曹健,陈庆奎.虚拟机内存迁移技术研究.小型微型计算机系统,2014,2:412–418.
6 胡浩.Linux 内存优化 KSM.计算机光盘软件与应用.2014,6:157–158.
7 赵阳,刘明芳,林曦君.基于 KVM 共享内存的虚拟可信管道的建立方法,2013,3:9–12.