

杭州电子科技大学

硕士学位论文

题目: KVM 客户机主动共享的内
存超量使用策略研究

研究生 杜 炜

专 业 计算机软件与理论

指导教师 万 健 教授

完成日期 2013 年 12 月

杭州电子科技大学
学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

论文作者签名：

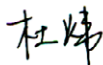


日期：2014年3月15日

学位论文使用授权说明

本人完全了解杭州电子科技大学关于保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属杭州电子科技大学。本人保证毕业离校后，发表论文或使用论文工作成果时署名单位仍然为杭州电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。（保密论文在解密后遵守此规定）

论文作者签名：



日期：2014年3月15日

指导教师签名：



日期：2014年3月15日

杭州电子科技大学硕士学位论文

**KVM 客户机主动共享的内存超量
使用策略研究**

研 究 生： 杜 炜

指导教师： 万 健 教授

2013 年 12 月

Dissertation Submitted to Hangzhou Dianzi University
for the Degree of Master

Overcommitting Memory by Initiative Share from KVM Guests

Candidate: Du Wei

Supervisor: Prof. Wan Jian

December, 2013

摘 要

随着云计算产业的快速发展,云服务器集群中的计算资源的管理已成为人们关注的重点。相比于传统的服务器集群,云计算通过虚拟化等技术改进服务器的软硬件资源的管理和分配的方式,提高了硬件资源的整体利用率和服务的可靠性。然而,虚拟化技术在应用于服务器资源管理的过程中也遇到了一些挑战,其中的挑战之一就是虚拟机内存资源分配的合理性。合理高效的虚拟机内存资源分配管理是保障云计算环境整体效率的关键之一。内存作为计算和存储的中间环节在虚拟化平台中显得尤为重要,进一步优化虚拟机内存管理将是提高虚拟化平台整体资源利用率的关键。

然而,由于现有的 KVM(Kernel-based Virtual Machine)虚拟化平台中内存管理的方式无法在物理机上分辨客户机内部内存页面的具体状态,客户机内存页面仅以被访问的频率和内容上的重复性的形式被物理机所感知,可能出现一些不合理的情况,各种现有的内存超量使用策略也都存在一定的局限性或缺陷。

本文提出了新的超量使用内存策略,即以修改客户机系统内核的方式使客户机主动提出与物理机共享其中未使用部分内存,以修改物理机上 KVM 模块的方式回收客户机愿意共享部分的内存,从而排除此部分客户机内存对物理机内存调度算法的干扰。

本策略可充分利用现有 KVM 平台的各种策略模拟操作系统级虚拟化技术的内存管理特点,保留物理机操作系统的页面交换策略,让其感知客户机操作系统的内存访问频率;以 KSM 策略合并客户机中重复页面的方式模拟操作系统级虚拟化中的加载程序或文件的去重特点;以本设计回收客户机中未使用内存的方式模拟操作系统级虚拟化中客户机程序将未使用内存释放回物理机操作系统内核内存管理单元;以本设计关于客户机操作系统内核物理内存页块优先级的处理模拟操作系统级虚拟化中统一的伙伴页块链表管理。

在与其他现有的内存超量使用策略对比时,本文提出了基于微观内存页面行为的模型,随后展开介绍了两种不同的设计方案和内存外部碎片对本设计的影响,最后通过 KVM 虚拟化平台下模拟内存占用的实验论证了本设计可取得良好的效果。宏观测试实验部分从系统整体的角度对比了开启与未开启 IMR 策略的情况下,客户机启动与动态内存使用场景下物理机内存使用,以及物理机交换区空间使用状态方面的差别。

本设计主要适用于匿名内存使用量波动较大的计算环境,文中选择了几个主

流开源软件进行匿名内存用量的测试，验证了本设计的基础；在基于微观页面行为的测试中对比了页面交换和 **KSM** 策略，本策略在处理客户机空闲物理内存块的速度上至少提高了三个数量级。

关键词：KVM，虚拟机，内存，超量使用，回收，主动共享

ABSTRACT

Along with the rapid development of cloud computing industry, the compute resource management in cloud server clusters has already become the focus. Comparing to traditional server clusters, cloud computing has optimized the management and allocation of software and hardware resources, as well as the enhancement in hardware resource utilization and service reliability via virtualization technology. However, the application of virtualization technology has encountered some challenges, one of which is the rationality in virtual machine memory allocation. Efficiency in virtual machine memory management is one of the key components in guaranteeing the total efficiency of cloud computing platforms. The memory resource plays an important role as the medium between compute and storage, further memory management optimizations are the key to increase the overall productivity of virtualization.

However, since the memory management method on the current KVM platform is not able to distinguish the detailed status of guest memory pages on the host, the guest memory pages can only be detected by the visit frequency and duplication in content on the host, which could be unreasonable, and the existing memory overcommit strategies have different drawbacks or limits when dealing with this problem.

A new design of memory overcommit strategy is introduced in this thesis that modifies the guest kernel to let it share the unused memory initiatively, and reclaim these memory in the modified KVM module, thus eliminating the corresponding interference to the memory scheduling algorithm on the host.

This design has the capacity to utilize almost all the existing memory overcommit mechanisms in KVM platform to simulate the operating system level virtualization that reserves swapping and let the host detect guest page frequency automatically; it simulates the loader with KSM strategy to reduce redundancy; it reclaims the unused physical memory pages on the guest like ordinary memory return from applications; and it keeps pace with the guest operating system in memory buddy priority.

When comparing to other existing memory overcommit strategies, we come up

with a model based on actions of a single memory page, meanwhile discussing two implementations and the influence from external memory fragment. Finally specific tests have been designed for the memory overcommitting strategy described above, proving its effectiveness. The macro scale experiment section has a comparison between the original KVM platform and KVM with IMR to show the difference in guest system startup and dynamic memory usage scenarios, as well as the swap space utilization on the host.

The design in this thesis is mostly made for situations with big changes in anonymous memory usage, and tests on anonymous memory usage are made in some mainstream open source software to demonstrate the basis of this thesis; when dealing with the unused guest memory pages, the improvement is at least a thousand times faster than swap and KSM in microcosmic level tests.

Keywords: KVM, virtual machine, memory, overcommit, initiative share

目 录

摘 要.....	I
ABSTRACT.....	III
目 录.....	V
第一章 绪论.....	1
1.1 研究背景.....	1
1.2 系统虚拟化简介.....	2
1.3 虚拟化技术分类.....	4
1.3.1 硬件仿真虚拟化.....	4
1.3.2 完全虚拟化.....	5
1.3.3 半虚拟化.....	5
1.3.4 硬件辅助虚拟化.....	5
1.3.5 操作系统级虚拟化.....	6
1.4 KVM 虚拟化平台简介.....	6
1.4.1 CPU 管理.....	7
1.4.2 内存管理.....	7
1.4.3 设备 I/O 管理.....	9
1.5 虚拟机内存研究现状.....	10
1.5.1 KVM 内存管理现状.....	11
1.5.2 其他虚拟化平台内存管理现状.....	12
1.5.3 虚拟化技术发展趋势.....	12
1.6 论文结构.....	13
第二章 KVM 客户机主动共享的虚拟机内存管理.....	14
2.1 研究内容.....	14
2.1.1 改进半虚拟化环境.....	15
2.1.2 改变操作系统级虚拟化环境.....	16
2.2 各虚拟化平台内存管理策略分析.....	17
2.3 框架结构设计.....	19
2.3.1 KVM 虚拟机内存回收原理.....	19
2.3.2 回收 KVM 客户机主动共享内存的设计框架.....	20

2.4 对比 KVM 其他内存超量使用策略及建模.....	21
2.4.1 与页面交换策略的比较.....	21
2.4.2 与 KSM 策略的比较.....	22
2.4.3 与内存膨胀策略的比较.....	23
2.5 本章小结	23
第三章 客户机空闲内存页面标记与回收.....	24
3.1 修改客户机操作系统的方式	24
3.1.1 平铺记录法.....	24
3.1.2 链表记录法.....	25
3.1.3 客户机信息记录方式对比分析.....	25
3.1.4 空闲内存页块释放优先级及其影响.....	27
3.2 具体实现与算法描述	27
3.2.1 修改客户机操作系统内核.....	27
3.2.2 修改物理机 KVM 内核模块	28
3.2.3 相关算法描述.....	29
3.3 内存碎片的影响	30
3.4 本章小结	31
第四章 性能测试与分析.....	32
4.1 实验环境	32
4.2 实验结果及分析	32
4.2.1 各应用程序匿名内存用量分析.....	33
4.2.2 对比非超量使用内存环境.....	34
4.2.3 对比页面交换策略.....	35
4.2.4 对比 KSM 策略.....	36
4.2.5 IMR 策略微观测试	37
4.2.6 IMR 策略宏观测试	38
4.3 本章小结	39
第五章 总结与展望.....	41
5.1 工作总结	41
5.2 研究展望	42
致 谢.....	43
参考文献.....	44
附录.....	47

第一章 绪论

1.1 研究背景

在传统计算环境中,大型服务器集群的基础架构往往过于复杂和脆弱。相当比例的资金成本主要用于系统维护。随着用户对系统可靠性和管理成本的要求越来越高,计算机底层架构需要一种更好的策略来降低维护的复杂性,提升计算的能效。从服务器负载的角度看,由于计算机硬件性能的快速提升和特定服务对计算资源需求的不均衡性^[1],服务器集群中往往大量存在各种资源被闲置的情况。为了解决这种资源浪费的现象,虚拟化技术^[2]应运而生,在将计算隔离,保持应用间无干扰的基础上,实现了对低负载计算的整合,降低了整体能耗和空间占用^[3],同时也简化了系统管理的复杂度。

为提升计算效能和方便用户操作,云计算提出了 IaaS^[4](Infrastructure as a Service, 基础架构即服务)的服务模式,即在依靠虚拟化技术提高计算密度的基础上通过互联网向用户提供各种基础计算资源。虚拟化技术通过实现对各种硬件资源的模拟,隔离了不同的虚拟计算环境,特别适用于低负载型服务器的集中计算和管理。而硬件资源的超量使用技术^[5]则是进一步提高虚拟化平台整体计算密度的关键所在。

云计算相比传统的虚拟化平台更强调计算资源配置^[6]的灵活性,它将更大规模的各种硬件计算资源统一管理调度^[7],从各种资源池中按需分配,以服务的形式提供计算^[8]。而这一切对用户而言是透明的,每个用户仍然认为自己所获得的是一个普通的服务器,仍然可以按以往的方式使用。所以,云计算时代的企业一般不再需要专门的信息技术管理团队,无需购买、放置、维护有形的、配置相对难以改变的服务器,可以随时、快捷地改变计算需求,减少包括安全性防护在内的计算设备投资成本。而云计算服务供应商可通过大幅提高软硬件、空间、人力等资源的利用率的方式降低计算成本,从中获利,形成良性发展的计算机基础架构生态链。

云计算时代的来临,与以往相比,硬件平台资源的获取变得更加廉价和便捷,亚马逊、微软、VMware、IBM、阿里巴巴等知名企业提供的商业计算服务和各种底层的开源项目(如 OpenStack^[9]等)也应运而生。而造就 IaaS 的一切变化都来源于强大、成熟、可靠的虚拟化技术。虚拟化技术的核心价值在于对不同计算环境的隔离和硬件资源的共享使用,但这两方面往往是不可兼得的,关键在于

如何权衡利弊,不同的虚拟化环境中对于不同的计算场景作了相应的侧重,然而仍然存在进一步提升隔离环境中计算资源共享效率的空间,这也是本文研究的重点方向。

虚拟化技术通过实现对各种硬件资源的模拟,隔离了不同的虚拟计算环境^[10],特别适用于低负载型服务器的集中计算和管理。而硬件资源的超量使用技术则是进一步提高虚拟化平台整体计算密度的关键所在,但是由于虚拟化环境中虚拟层和客户机对所掌握资源管理信息往往存在不一致性,在一些情况下可能导致虚拟机运行效率的降低;而过于强调虚拟机对硬件资源的需求又会导致虚拟层被动应对,无法协调调度,降低了资源的整体利用率。对于内存而言,为了更高效地超量使用内存资源,就必须在虚拟层软件中准确把握客户机操作系统中内存的实际使用情况,以便更好的进行调度。本研究针对 KVM^[11]客户机中未使用部分内存作了精确的管理,设计了一种由客户机操作系统主动提出共享内存请求的新机制,以极小的代价实现此类客户机内存的重新分配,进一步提升了虚拟机内存的整体效率。

1.2 系统虚拟化简介

虚拟机监视器是系统虚拟化技术的核心,它是向虚拟机提供硬件资源的平台,位于客户机操作系统和物理硬件(或物理机操作系统)之间,主要功能是将真实的硬件资源透明地^[12,13]提供给虚拟机,使每个客户操作系统在逻辑上独立于实际硬件环境,处理器、内存、输入输出和设备都须要由虚拟机监视器进行管理,而绝非由各虚拟机自行占用,最终的目的是以统一管理硬件资源的形式提升整体效率。

在传统计算机上,除了最早期的计算机是靠程序员用机器指令直接和硬件交互之外,计算机硬件资源一般都由操作系统进行管理调度的;在虚拟化计算环境中,主要有两种类型的硬件资源管理方式,其一为虚拟机监视器取代传统操作系统运行于硬件平台之上,在虚拟层上方运行若干个客户机操作系统,由虚拟机监视器根据客户机操作系统的请求统一管理各种硬件资源,尽管客户机操作系统仍然认为是自己在管理硬件资源;其二为硬件平台之上仍然是传统的操作系统,但这个操作系统中包含了虚拟机监控器软件,或仅仅是一些硬件资源访问方面的逻辑控制机制。

总之,虚拟化技术相当于硬件资源的软件抽象,方便用户管理硬件资源,在多客户机的虚拟化环境中提升硬件资源的使用效率,它主要有以下一些特性:

(1) 隔离性。不论是一个操作系统中的逻辑隔离,还是用虚拟机监视器(Virtual Machine Monitor, VMM)软件保证对不同实体的隔离,隔离性是系统虚拟化的首要前提,保证不同实体间相互不干扰。

(2) 并行性。在隔离的基础上,虚拟机存在的最重要的意义就在于可在一台物理机上存在多个逻辑机器同时运行。这也是虚拟化计算提高能效,减少空间占用的基本方式,如果只有一台虚拟机运行,既无法减少对空间的占用,也会在虚拟层软件上损耗一定的性能。

(3) 低损耗。虚拟层软件一般不应当消耗过多资源。为了尽量减少这种无谓的性能损失,业界在软硬件方面都作了深入的探索,软件层面有许多虚拟化平台在设计上直接让 VMM 紧贴硬件运行,相对于一个只包含虚拟化功能的基本操作系统;硬件层面一些厂商直接将一些计算较复杂的底层计算嵌入硬件中,如处理器上会存在关于虚拟化方面的硬件辅助,Intel 的 VT-x 和 AMD 的 AMD-V 用于减少 VMM 对特权指令模拟的开销,Intel 的 EPT^[14](Extended Page Table, 扩展页表)和 AMD 的 NPT^[15](Nested Page Table, 嵌套页表)技术用于加速虚拟机访问内存页面,再如主板上的芯片组若支持 Intel 的 VT-d 或 AMD 的 AMD-Vi 则可有效地加速虚拟机的 I/O^[16]。

(4) 可靠性。虚拟机区别于物理机的另一个重要因素在于它的可靠性。在应对突发状况时,虚拟机由于具有规范性可以实现快速动态迁移,从而保证了服务的可靠性。

(5) 易管理。相比物理机,虚拟机在管理方面的优势十分明显。例如可用虚拟机管理软件实现批量操作,无须特殊硬件即可实现远程开机等。

从硬件资源的角度看,虚拟化技术主要包括 CPU 虚拟化、内存虚拟化、设备和 I/O 虚拟化这三个部分。

CPU 虚拟化方面,本文只考虑 x86 架构。这是一项可将单个物理 CPU 模拟成多个虚拟 CPU 的技术,不同客户机操作系统上的虚拟 CPU 之间没有干扰。在 x86 架构中,一共有 4 个特权等级,0 级权限最高,用于执行操作系统内核以及虚拟层软件中的指令,3 级权限最低,用于执行用户空间程序的指令。客户机操作系统内核代码运行于 1 级权限。

内存虚拟化技术可实现更高效地调度管理物理内存,而虚拟机始终认为自己访问的是整块的连续内存,但在实际硬件内存上可以不连续。

由于客户机无权直接访问物理机内存,VMM 中须要有内存访问控制的方法。不同的虚拟化平台在内存虚拟化方面有不同的实现机制,以 KVM 虚拟机为例,主要有以下两种页表机制:

(1) 影子页表

影子页表机制中,VMM 以客户机操作系统的页表为样本,在虚拟层上制作该页表的副本,每当客户机的页表项或 CR3 寄存器发生改变时,虚拟层的副本会同步更新。这是一种纯软件的页表机制,在不同的虚拟化平台上有比较广泛的

应用。但是，因为须要维护客户机操作系统和虚拟层两处页表项的同步，访问和脏页标志位的管理，及大量缺页异常的管理，以及保持 SMP(Symmetric Multi-Processing, 对称多处理)客户机不同处理器上地址转换的一致性，影子页表机制的额外开销很大，在特定情况下可能占据虚拟层总额外开销的 75%^[15]。

(2) 二维页表^[15]

二维页表机制可有效避免影子页表中为保持两处页表一致性而引起的额外开销。具体实现方式是做两层地址转换，第一层为客户机虚拟地址到客户机物理地址的转换，第二层为客户机物理地址到物理机物理地址的转换。第二层转换当前主要靠硬件加速，如 Intel 的 EPT 和 AMD 的 NPT，否则须要先把客户机物理地址转换到物理机虚拟地址上，再从物理机虚拟地址转换成物理机物理地址。这种二维页表地址转换机制的缺点是页表项的访问的次数增加了许多，如果快表的缓存不够大，内存访问的速度会受到一定影响。

设备和 I/O 虚拟化的功能方面，要求同一虚拟化平台上虚拟机所见的设备符合统一的标准，以便于虚拟机的管理，比如虚拟机的迁移等；性能方面，要求尽可能的高效，目前业界在硬件上已作了优化，如 Intel 的 VT-d 和 AMD 的 AMD-Vi 均为对虚拟机 I/O 的加速设计。

1.3 虚拟化技术分类

不同的云计算应用场景往往需要不同的虚拟化技术支持，有的是面向高层应用服务的环境，有的是面向开发者，有的是面向企业内部的私有云计算。为了满足不同的需求，虚拟化技术的实现形式是多种多样的。x86 架构的虚拟化技术可分为以下五类：硬件仿真虚拟化、完全虚拟化、半虚拟化、硬件辅助虚拟化和操作系统级虚拟化。

1.3.1 硬件仿真虚拟化

硬件仿真虚拟化是在物理硬件上建立一个硬件模拟层，即硬件虚拟机。主要用于硬件仿真，也就是在特定硬件架构之上运行不同架构的操作系统软件的方式，优点是便于交叉调试，客户机操作系统所见为仿真封装后的物理硬件。但这是一种极为复杂，且运行效率极为低下的虚拟化方式。QEMU^[17]就是该类虚拟化技术的典型实现之一。

1.3.2 完全虚拟化

相比于硬件仿真虚拟化技术，完全虚拟化技术仅针对特定硬件架构实现，存在一个支持底层物理硬件架构的虚拟层软件或包含虚拟机层软件的操作系统，所有客户机操作系统都运行于该虚拟层之上。完全虚拟化技术采用动态二进制代码翻译技术处理客户机操作系统中的特权指令^[18]。该技术可在客户机操作系统执行敏感指令前，将代码的执行切换到虚拟机监控器中，然后由虚拟机监控器把这些敏感指令翻译成相应功能的非敏感指令后执行。完全虚拟化技术的优点在于无须修改客户机操作系统，底层物理硬件仍然是被封装的，尽管指令的动态转换仍然有一定程度的开销，但因为非敏感指令无需转换，完全虚拟化技术在性能方面相比硬件仿真虚拟化技术有了明显的提升。Virtual PC、VMware Workstation、Virtual Box 等虚拟化软件都属于完全虚拟化技术的实现。

1.3.3 半虚拟化

半虚拟化技术^[19]是完全虚拟化技术进一步的发展产物。和完全虚拟化技术一样，半虚拟化技术也需要一个紧贴真实硬件运行的虚拟机监控器或包含虚拟机监控器的操作系统，以此实现不同虚拟机对底层硬件平台的共享。但是，为了进一步提高虚拟机系统的执行效率，半虚拟化技术不再捕获翻译客户机操作系统中的敏感指令，而是以修改客户机操作系统的方式，使之与虚拟层软件协作，将客户机操作系统中的敏感指令替换成对虚拟层软件中接口的超级调用，从而进一步提高虚拟机的代码执行效率。半虚拟化技术的缺点是客户机操作系统须与虚拟机监控器软件兼容，客户机操作系统能感知自己出于虚拟化环境中，对于一些私有且不开源的操作系统，半虚拟化技术不一定适用，视提供这些操作系统的厂家支持情况而定。VMware ESX Server、Xen、Hyper-V 等虚拟化平台都是半虚拟化技术的实现。

1.3.4 硬件辅助虚拟化

不论是硬件仿真虚拟化、完全虚拟化还是半虚拟化，它们都是纯软件实现的虚拟化技术，对物理处理器的硬件架构没有任何要求。但不论上述哪种虚拟化技术，都存在或多或少的中间层软件转换的开销。为了进一步减少这种开销，业界采用了硬件辅助虚拟化技术^[16]减少原本由软件完成的一些中间环节。硬件辅助虚拟化技术是一种建立在全虚拟化或半虚拟化技术基础之上的辅助技术，而非独

立的一类虚拟化技术。Intel 和 AMD 这两家处理器厂商提出的关于这种技术的基本思想就是引入新的虚拟化扩展指令和处理器运行模式来取代完全虚拟化技术的动态二进制代码翻译和半虚拟化技术的超级调用,从而进一步避免一些中间环节的额外开销。例如 Intel VT 技术在处理器硬件中增加了虚拟机扩展指令集 (Virtual Machine Extensions, VMX), 定义了两种运行模式, 根模式和非根模式, 虚拟机管理器运行于根模式中, 客户操作系统运行于非根模式。在完全虚拟化和半虚拟化技术方案中所提及的所有虚拟化软件平台现在都有 Intel 和 AMD 这两大处理器厂商的硬件辅助虚拟化技术的支持。

1.3.5 操作系统级虚拟化

操作系统级虚拟化技术^[20]是一种非常特殊的虚拟化技术,与前面所介绍的各种技术不同,该方法中不存在关于物理硬件的模拟,也不存在多个操作系统,所有的客户机操作系统本质上就是物理机操作系统,只是这个物理机操作系统被修改过,增加了“容器”的概念来实现虚拟客户机操作系统,目标是实现资源访问的控制,主要适用于对操作系统本身无修改需求的高层软件应用服务。此外,由于不存在虚拟层,所有的资源调度管理都在物理机操作系统内核中进行,所以,操作系统级虚拟化是所有虚拟化技术中整体运行效率最高的。但这种虚拟化技术的缺点是各个虚拟环境之间的隔离性较弱,且所有虚拟机都必须为相同的操作系统,在客户机内部无法实现系统的更新,特别是当物理机操作系统存在安全漏洞时,客户机的安全性取决于物理机管理员的操作。FreeBSD 下的 jail 和 Linux 下的 OpenVZ 等都是基于操作系统级虚拟化技术的实现。

1.4 KVM 虚拟化平台简介

本文针对具有硬件辅助虚拟化技术支持的 KVM 虚拟化平台的内存资源管理策略进行了研究。首先对 KVM 虚拟化平台的基本原理进行全面介绍。

KVM 是 x86 架构下的 Linux 操作系统中的一个虚拟化平台^[21],仅在处理器硬件支持虚拟化技术的环境中运行,最早出现在 Linux 2.6.20 版本中。它包括一个可加载的内核模块,该模块提供了核心的虚拟层软件控制;还包括一个用户空间中的修改版 QEMU 模拟器,用于用户空间的上层操作,如创建虚拟机、创建虚拟处理器等,这些操作通过“/dev/kvm”这个设备文件完成和 KVM 内核模块的交互。

KVM 虚拟机在物理机上表现为一个普通的进程,与其他进程一样参与物理机操作系统的资源分配和任务调度。相比于 VMware ESX Server、Xen、Hyper-V

等直接运行于硬件之上的虚拟化平台，KVM 的设计极为精简，可以直接利用 Linux 系统现有的各种驱动程序和其他软件。但作为云计算时代服务器集群虚拟化的基础，这样的设计在细粒度的控制上显然还有进一步提升的必要。

KVM 中的资源管理主要包含 CPU、内存和设备 I/O 的管理，下面就这几项资源的管理方式分别进行讨论。

1.4.1 CPU 管理

在 CPU 的调度上，KVM 平台没有作任何干涉，各虚拟机的运行只取决于物理机操作系统内核的调度。在访问权限控制方面，相比传统的 0 级的内核模式和 3 级的用户模式，增加了 1 级的客户模式，从而确保了用户空间的程序无法直接访问客户模式的虚拟机内核数据，客户模式的虚拟机操作系统也无法直接访问物理机操作系统内核数据，提高了可靠性和隔离性。KVM 平台中每个客户机可有一个或多个虚拟的 CPU，即 VCPU，以线程的形式运行，该结构内容固定，不可修改，由支持虚拟化技术的处理器硬件访问。从客户机操作系统时间的角度分析，存在被物理机或其他虚拟机窃取的时间，客户机操作系统真实运行时间加上被外部窃取的时间才是总时间，即内核代码中的“墙上时间”。

1.4.2 内存管理

内存是介于计算和存储的最重要的中间环节，也是最频繁被访问的设备之一。在虚拟化环境中，内存管理的效率对整体性能有着至关重要的影响，如何进一步提高内存虚拟化的性能一直以来都是业界关注的重点^[22]。

就内存资源调度方面而言，和 CPU 的资源调度一样，KVM 平台也同样按照物理机操作系统内存管理的机制进行，KVM 客户机所在进程对应的物理内存页面可能出于物理机内存压力的原因被交换到硬盘上。但除此之外，KVM 平台还有其他一些针对内存管理方面的优化，如用于相同内存页面合并的 KSM，以及用于转移内存压力的内存膨胀策略等。

KSM 是 Linux 中用于相同页面合并的一种内存管理策略。该策略有一个名为 ksmd 的内核守护进程，定期进行内核内存页面扫描，将相同内存页面合并为只读的页面副本，在合并后的页面发生改变时可以写时复制的方式生成新内存页面。虽然这种内存管理策略在某些冗余度很高的情况下效果显著，但其计算复杂度较高，即便被指定了内存扫描的区域。

内存膨胀是 KVM 平台中的一种内存协调机制。它的实现原理并不复杂，只须在客户机中安装气球驱动，本质就是在内核中申请一些物理内存空间，但并不

实际使用,从而根据内存调度算法将最不重要的其他客户机内存页面交换到虚拟机的硬盘上,然后在虚拟机监视器中把这些在客户机中该驱动所占内存对应的物理机物理页面释放,最后再切换回虚拟机中,由气球驱动释放先前申请的客户机物理内存。

从 KVM 虚拟化平台的内存页表管理逻辑看,主要可分为两种,第一种是早期纯软件的影子页表方式,第二种是有虚拟化硬件支持的二维页表方式。这两种页表管理方式均以每个客户机操作系统对应的 kvm 结构体中 memslots 结构体数组作内存区域的映射,将客户机操作系统中的虚拟地址(guest virtual address, gva)转换到物理机操作系统的物理地址(host physical address, hpa)上。

影子页表在许多虚拟化平台上都有应用,正如前面 CPU 管理部分提到的,因为客户机操作系统的权限为 1 级,无法直接访问 0 级的物理机操作系统中的内存数据,这种页表管理机制在虚拟机管理器中构造了与客户机操作系统完全相同的页表,并保持两处页表一致,从而实现通过客户机操作系统中的页表访问外部物理机操作系统管理的内存页面。在地址转换的过程中, KVM 虚拟机监控器会拦截客户机操作系统对 VCPU 的 CR3 寄存器的访问,每个 VCPU 的 CR3 寄存器指向影子页表的根目录物理地址,客户机中的页表实际记录的是物理机物理内存页面的地址。图 1.1 所示为影子页表的地址转换,其中 PGD、PUD、PMD 和 PTE 分别为页全局目录、页上级目录、页中间目录和页表项。

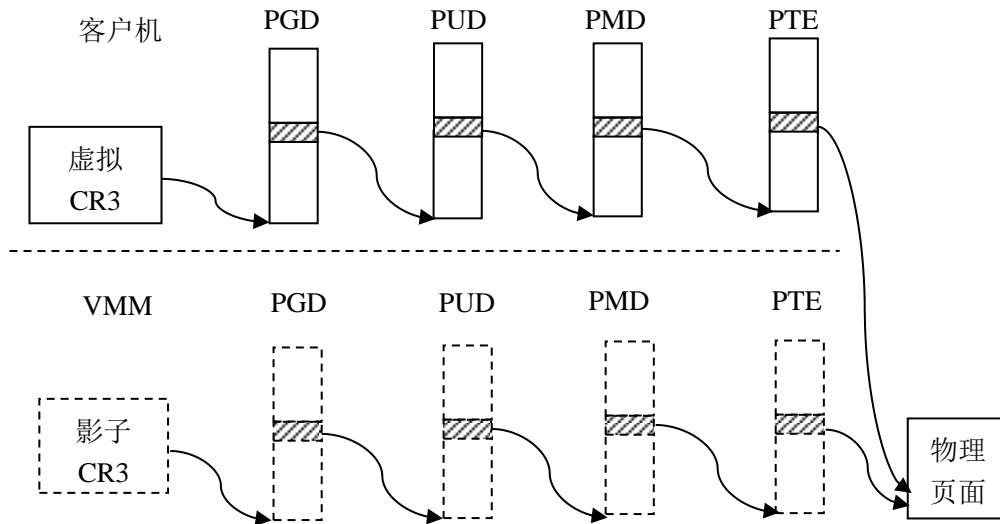


图 1.1 影子页表地址转换

图中 VMM 的影子页表与客户机内存页表相同,但这种方式在两处页表同步方面的开销较大,但好处是内存访问的步骤没有增加,相当于物理机操作系统可以直接在快表(Translation Lookside Buffer, TLB)中缓存客户机操作系统虚拟地址对应的物理内存页面。

为解决影子页表中页表频繁同步开销较大的问题，二维页表机制设计了两层页表转换的方法，这种方法可以避免虚拟机监控器对客户机内部页表的控制，只须处理从客户机物理地址到物理机物理地址的第二层转换。Intel 和 AMD 两大处理器厂商在第二代硬件加速虚拟化技术推出时分别提出了 EPT 和 NPT 的硬件加速方案。这两种方案设计基本相同，客户机操作系统可以自行管理内部的页表，而页表的内容仍然在物理机内存页面中，每次须要读取其中的数据时，不论是在客户机页表的 PGD、PUD、PMD 还是 PTE 中的，都须要通过硬件把相应的客户机物理地址转换成物理机物理地址，这也就形成了一个二维的页面查找过程。这种页表机制在硬件的辅助之下，效率比影子页表有了显著的提升。

当然，这其中还存在一个问题，它的每次读取内存页面过程的步骤数量比影子页表增加了很多，几乎达到了平方的关系，一旦某个页面没有在 TLB 中缓存(但一般 TLB 缓存较大时很少出现这种情况)，原本只需 4 次额外(除最终目的内存页面之外)页表内存访问的过程将变成 24 次额外内存访问，如图 1.2 所示，第一个虚线箭头为客户机所见 PGD 至 PUD 的转换，但实际内存访问为下面实线箭头部分的 5 次内存访问过程，客户机的 PUD、PMD、PTE 的访问过程同理，除最终的目的页面外，共有 24 次内存访问。

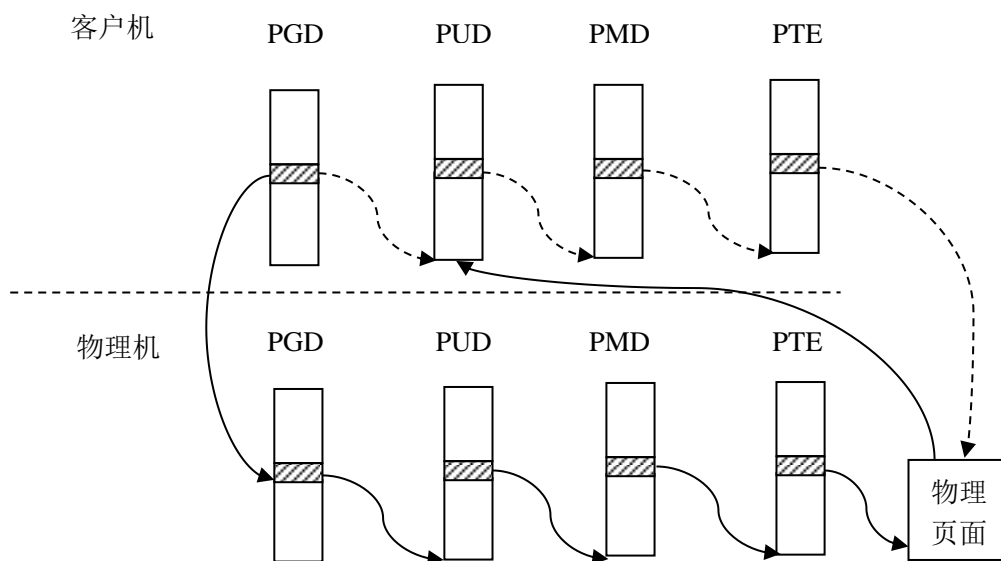


图 1.2 二维页表地址转换

1.4.3 设备 I/O 管理

KVM 早期采用仿真传统物理设备的方式支持虚拟机 I/O。这种方式的特点就像硬件仿真虚拟化技术一样，客户机操作系统能看到和物理机操作系统相同规格的设备，能直接使用现有的设备驱动访问这些虚拟设备，不要求修改客户机操作的

系统。这种实现的过程是在虚拟机监控器上拦截客户机操作系统访问设备 MMIO(Memory mapping I/O, 内存映射输入/输出)和 PIO(Port I/O, 端口输入/输出)的空间, 然后以不同的过程分别处理 MMIO 或 PIO 访问, 这两种方式的区别是 MMIO 须要占用内存地址空间, 有统一的内存和 I/O 地址空间, 可以像访问内存那样访问设备, 而 PIO 不须要占用内存地址空间, 有独立的 I/O 地址空间和特定的 I/O 访问指令。

虚拟机 I/O 的另一种实现方法是虚拟功能设备。就是指虚拟机 I/O 设备只要能够实现操作系统需要的功能, 而不必完全按照实际的物理设备处理。这样的方式需要客户机操作系统配合, 无法直接利用现有的一些驱动。不同的 VMM 实现往往有自己独立的, KVM 的虚拟设备的实现机制是 VirtIO^[23]的技术。同样是因为访问权限的因素, 客户机操作系统须告知虚拟机监控器其交互数据区域, 从而实现信息的沟通。这种方式可有效减少 PIO 和 MMIO 带来的切换出虚拟机的次数, 从而提高客户机的运行效率。

以上两种方式由于仍然需要客户机操作系统和虚拟机监视器间的交互, 性能上仍然存在进一步优化的空间。在最新的硬件平台上, 若有 Intel 的 VT-d 和 AMD 的 AMD-Vi 技术支持, 客户机操作系统可直接操作 PCI(Peripheral Component Interconnect)物理设备, 从而使虚拟机的 I/O 效率最大化。

1.5 虚拟机内存研究现状

大规模服务器集群, 特别是一些数据中心的资源利用率较低已成为业界关注的重点。其中的原因在于在真实的计算环境中, 只有特定情况下, 服务端程序才会达到峰值负载, 但为了满足峰值计算的需求, 在传统服务器中的硬件配置一般都要达到峰值计算对应的规格。而即便在应用了虚拟化计算的环境中, 整体资源利用率仍然偏低^[24], 这就须要人们进一步挖掘虚拟化计算的潜力, 提高计算的密度。

服务器硬件资源根据占用的情况可分为两大类。一类是持久占用型资源, 如内存和磁盘空间等, 即使不使用也可以保持占用状态; 另一类是短暂占用型资源, 如处理器和输入输出等, 只在使用时占用, 用完后由操作系统管理分配给其他程序使用。短暂占用型资源具有天然的时分复用特性, 而持久占用型资源的时分复用代价非常巨大, 甚至不可行, 这就违背了虚拟化计算对硬件资源复用提高计算密度的初衷。

1.5.1 KVM 内存管理现状

KVM 是 Linux 操作系统的一个内核模块，为客户机提供与物理机交互的接口。分配给客户机的内存为物理机上的一段线性地址区间，初始状态不分配物理页面。KVM 虚拟化平台允许内存超量使用^[25]，主要通过以下几种方式提高超量使用的效率，否则客户机在物理机上对应的内存页面量只增不减，即使客户机中释放过。如图 1.3 所示：

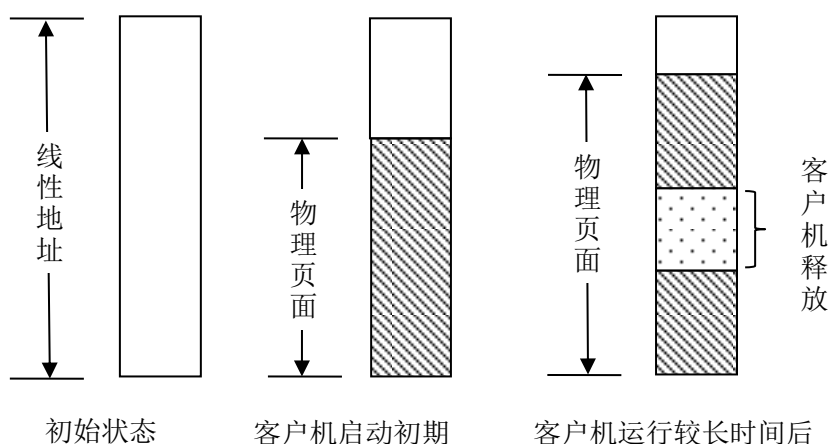


图 1.3 KVM 虚拟机使用内存的状态转换图

在没有任何超量使用内存策略参与下，图 1.3 只能从左向右进行状态转换。业界设计了以下一些内存超量使用的策略，使物理机更高效的利用内存资源。

为提高计算密度并缓解上述问题，KVM 平台当前超量使用内存的策略有以下几种：

(1) 页面交换：物理机有内存压力时进行，根据 LRU 算法将较长时间未使用的页面交换到磁盘上。物理机的内存压力能及时降低，但有磁盘访问的开销。另外，物理机对客户机内存的实际使用情况并不了解，无法做到高效地管理客户机内存。

(2) 内存膨胀^[26] (memory ballooning)：客户机内安装有驱动程序，由物理机指示该驱动在客户机上控制内存“气球”的大小，在物理机上管理“气球”所对应的内存。内存压力从物理机转移到客户机中，可根据客户机内存使用情况较高效地回收内存页面，但须手动执行，不灵活、不精确，可能出现膨胀失败的情况。

(3) KSM^[27] (Kernel Samepage Merging，内核相同页面合并)：用一个内核线程扫描客户机进程所使用的内存区间，合并相同内容页面，写时复制。此特性需要消耗较多处理器资源且时效性较差，不一定能显著减轻内存压力，视页面内容重复程度而定。

1.5.2 其他虚拟化平台内存管理现状

VMware ESX Server(vSphere)中通过对内存使用情况采样分析^[26], 将客户机中未使用(但物理机可能认为已使用, 下同, 并简称未使用)的内存视为空闲内存按闲置内存税算法进行回收。

z/VM 平台中通过修改客户机操作系统标识和回收未使用的内存^[28], 但因其设计为针对合作模式的虚拟化平台, 除回收未使用内存外, 还对其内存使用情况细化分类, 允许一定限度内客户机锁定的内存页面在物理机上同步锁定, 虚拟层的内存管理一定程度上被动服从于客户机程序的实际内存使用情况。

OpenVZ^[29]是一种操作系统级的虚拟化技术, 在同一个内核中将不同虚拟环境的各种资源隔离, 由此内核统一管理所有虚拟环境的内存。

1.5.3 虚拟化技术发展趋势

内存资源方面, 从传统的物理机服务器到非超量使用型虚拟机的转变使计算变集中, 节约了硬件所占的空间; 从非超量使用型虚拟机到超量使用型虚拟机则进一步提高了计算的密度, 但也随之带来了较大的额外开销, 如页面交换方式的磁盘访问和页面合并方式的 CPU 计算。如图 1.4 所示, 进一步降低这种额外开销将是本研究的重点。

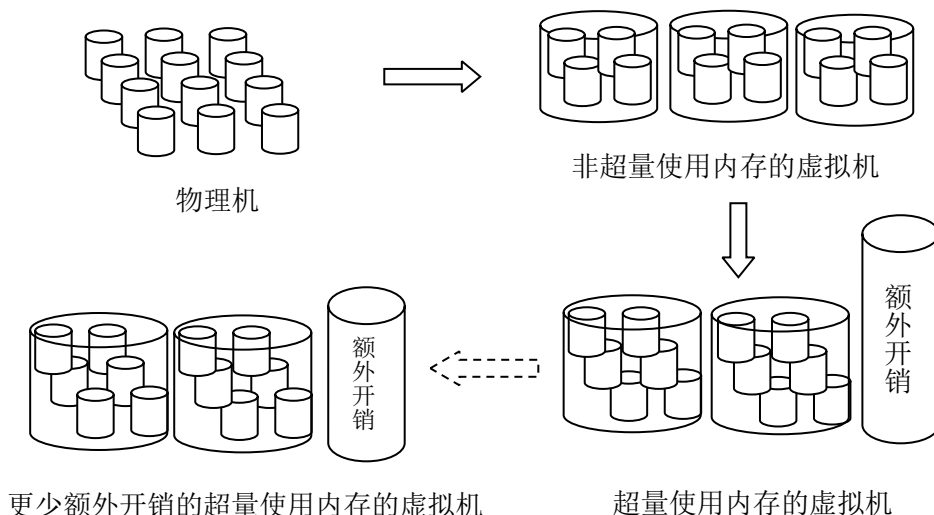


图 1.4 提高计算密度的方案演变

除操作系统级虚拟化技术外, 若要更高效的超量使用内存资源, 就必须准确把握客户机操作系统中内存的实际使用情况。所以, 为了达到处理计算密度和额外开销之间的矛盾的目的, 本文决定对持久占用型的内存资源管理方式进行改进, 主要工作包含以下几个方面:

(1) 对各种主流的虚拟化平台进行调研，了解它们在内存管理方面的策略，并进行分析。

(2) 选定 KVM 平台作为实验环境，针对内存资源在虚拟层和客户机间建立了沟通的机制，有选择的处理虚拟机内存需求变化，设计实现一种能有效降低内存超量使用额外开销的管理策略。

(3) 通过具体实验测出相关数据，建立关于本设计的数学模型，根据测试数据验证本设计方案的合理性。

1.6 论文结构

本文一共分为五章，具体内容如下：

第一章首先概述了虚拟化技术的相关研究背景、系统虚拟化的概况和分类，然后对 KVM 平台的虚拟资源管理和当前几种主要的虚拟化平台内存管理的研究现状作了重点介绍。

第二章先具体讲解了本设计的形成过程，包括对设计过程中其他一些方案的可行性分析。随后分析了各虚拟化平台内存管理的策略，提出了本设计的基本框架结构，最后将本设计与 KVM 和其他平台现有的内存超量使用管理方式进行了比较，并在此处提出了一个基于微观内存页面行为的模型。

第三章是关于本设计的具体实现的介绍，包括关键代码和数据结构、算法方面的讲解，分析了内存外碎片对本设计的影响。

第四章实验测试与分析，简要介绍了实验的方法，根据实际测试结果对之前的定性判断进行了更为精确的定量分析，以验证本设计的实际效果。此外，还对本设计的意义进行了分析。

第五章总结与展望，对本文作了总结，对未来虚拟化技术的发展进行了展望。

第二章 KVM 客户机主动共享的虚拟机内存管理

虚拟化技术是云计算的基础，它有效地提高了服务器集群的计算密度，降低了整体能耗，在软件层面上为提高系统资源利用率指明了研究方向。

然而，不同的计算机硬件资源在使用时有着不同的特点，现有的许多虚拟化平台往往只按照各硬件资源原有的时分复用特性进行资源调度。对于 CPU 和 I/O 等天然时分复用型资源，在使用完毕后，这些资源的使用权由虚拟机监视器收回，不会继续处于使用状态；但对于内存等空间型资源而言，以 KVM 平台为例，客户机第一次使用某个内存页面时，会向虚拟机监控器申请，但之后的释放或再次使用该页面的情况虚拟机监控器无法感知，也就意味着存在占而不用情况。这种情况将导致一些不必要的页面交换或页面合并等机制额外开销的产生，特别是页面交换操作会严重增加处理器的 I/O 等待时间百分比。

KVM 平台为了简化内存资源的分配和管理，把虚拟机抽象成了进程，采用峰值分配的方式限定虚拟机内存分配的上限。这种分配方式会因为上述占而不用现象造成一些不必要的损耗。为了尽可能的减少这样的损耗，本文采取了修改客户机内核和物理机上 KVM 内核模块的方式对 KVM 平台的内存管理机制作一些优化设计。虽然 KVM 平台原本是完全虚拟化技术的方案，但为了提高效率，业界已经对其进行过半虚拟化的改造，添加了一些半虚拟化的驱动程序，在 Linux 内核编译选项中也有像“CONFIG_KVM_GUEST”这种针对 KVM 平台客户机系统的特定配置选项。

2.1 研究内容

鉴于内存这种硬件资源的占用特性和它在计算机体系结构中的重要性，本文将对内存资源的管理作进一步的优化。不论是影子页表还是有硬件支持的二维页表的内存页面管理方式，相比传统主机操作系统的内存管理，虚拟化环境的内存操作代价要高很多，特别是在允许内存超量使用的虚拟化环境中，超量使用内存还可能会导致一些额外开销的出现，如页面交换会带来额外时间和 I/O 资源的开销，KSM 策略会带来额外 CPU 计算的开销，至于内存膨胀虽然效率较高，但有一定的限制。

在内存虚拟化方面，效率最高的是操作系统级虚拟化技术^[30]，不论是空间上还是时间上。其原因在于操作系统级虚拟化技术的底层仅有一个内核，不同的

虚拟机都由该内核管理，以软件逻辑的方式隔离，所以效率很高。而对于其他虚拟化技术而言，很难做到像同一个操作系统那样，在加载程序时直接检测被加载对象是否已存在于缓存中，直接避免很多重复数据的产生，而是以变通的方式实现这种空间上的效率的提升，但时间代价很大，如 KSM 技术只针对已在内存中的数据进行合并，内存膨胀策略在客户机内部强行占用一些内存供虚拟机监视器释放，因为虚拟机内部内存使用情况无法像操作系统级虚拟化一样直接被外部感知。

为了让虚拟化平台的内存管理效率更接近于操作系统级的虚拟化技术，同时允许不同虚拟机拥有不同的系统内核，尽可能地减少物理机系统的抖动，从而融合不同虚拟化类型的优势，本文设想了以下几种候选方案。

2.1.1 改进半虚拟化环境

第一种方法是改进半虚拟化环境。类似于页面合并和内存膨胀策略，进一步挖掘虚拟化环境中的不合理的内存管理方式。

此处可以继续细分为两种方式。

第一种方式为将不同客户机操作系统的用户空间程序的加载外置到物理机操作系统上，这样就实现了类似于操作系统级虚拟化环境中的效果，避免了内存合并策略的各种计算开销，也无须像内存膨胀策略那样从客户机操作系统中抽调内存。如图 2.1 所示。

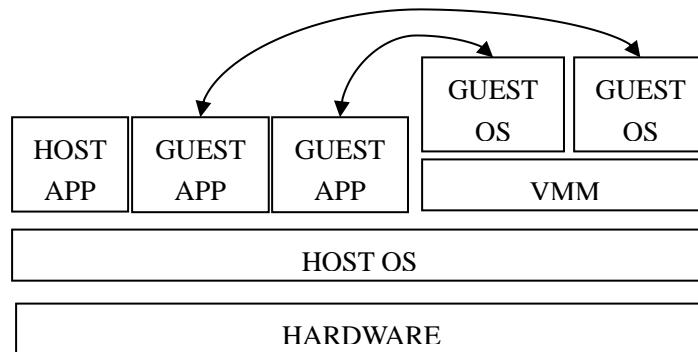


图 2.1 候选方案一

第二种方式为修改客户机操作系统内核，使其汇总记录客户机操作系统的可以较低代价被还原的内存页面，由物理机系统随时访问该记录，根据物理机的需要随时从客户机中抽调走这些记录了的内存页面。利用现有的内存合并机制合并那些未被记录的，真正被客户机操作系统使用中的重复内存页面，变相实现操作系统级虚拟化环境对内存资源管理的高效性。如图 2.2 所示。

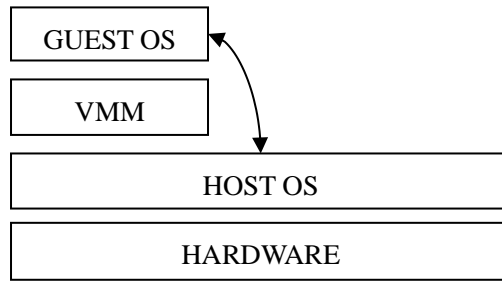


图 2.2 候选方案二

2.1.2 改变操作系统级虚拟化环境

第二种方法是改变操作系统级虚拟化环境。目的是为了允许虚拟机用户自行修改客户机所见的操作系统内核。所需的操作类似于写时复制的机制，在物理机操作系统上记录客户机对其内核所作的修改，以增量存储的方式记录，当客户机执行到其修改过的内核指令时，跳转到其特定的增量内核指令区执行。或者客户机为不同操作系统时仍将客户机操作系统架在虚拟机监视器上。如图 2.3 所示。

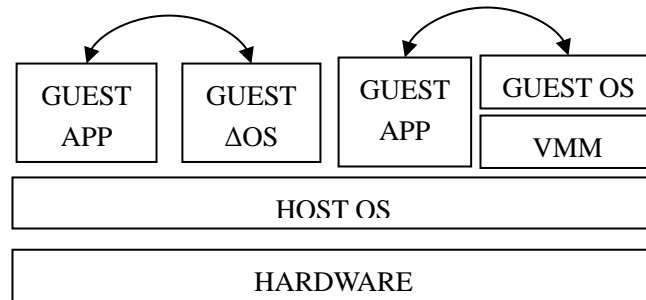


图 2.3 候选方案三

上述几种策略中，改变操作系统级虚拟化环境的方法最不可取，因为需要物理机操作系统在每个被修改处判断上层调用是否为该修改所属客户机操作系统，代价过大且几乎不可行。而改进半虚拟化环境的第一种方式虽然在客户机程序加载外置的操作上较合理，但已外置到物理机操作系统的客户机程序在执行仍然须要再次跳转回客户机操作系统内核处，这样频繁的跳转指令也会使客户机程序的执行效率下降，因此也不太合理。唯有改进半虚拟化环境的第二种方式具有可行性，因为它没有涉及到程序和内核的对应关系的处理，虚拟化环境的计算效率基本不会有影响。又因为 KVM 虚拟化平台的设计与该方法最接近，只须增加从客户机操作系统到虚拟机监控器的特殊内存页面的处理即可，故本文以 KVM 为基础虚拟化平台。

2.2 各虚拟化平台内存管理策略分析

除了 KVM 采用的页面交换、KSM 和内存膨胀策略之外, 还有以下一些独特的虚拟机内存管理策略。

VMware ESX Server(vSphere)中通过对内存使用情况采样分析, 将客户机中未使用(但物理机可能认为已使用, 下同, 并简称未使用)的内存视为空闲内存按闲置内存税算法进行回收。

z/VM 平台中通过修改客户机操作系统标识和回收未使用的内存, 但因其设计为针对合作模式的虚拟化平台, 除回收未使用内存外, 还对其内存使用情况细化分类, 虚拟层的内存管理一定程度上被动服从于客户机程序的实际内存使用情况。但是, 因为这个平台对客户机操作系统内核修改很多, 各种状态的转换极度复杂, 业界又提出了超级内存^[31](transcendent memory)的概念和一些具体的实现^[32], 只须作少量修改, 在原内核上添加一个内存管理的中间层, 负责管理客户机内部的内存以及与外部的虚拟机监控器或物理机操作系统交互, 这项技术主要用于优化客户机的页面交换和缓存, 基本管理单位为页, 该方案在 KVM 平台下的接口已出现在 Linux 内核 3.1 版中。

OpenVZ 是一种操作系统级的虚拟化技术, 在同一个内核中将不同虚拟环境的各种资源隔离, 由此内核统一管理所有虚拟环境的内存。

根据虚拟层对客户机操作系统中具体内存使用情况了解的程度, 虚拟机内存管理可分为估测型和精确型两类策略。估测型指虚拟层无法分辨客户机内存的具体类型, 依靠具体内存页面的访问频率判断其是否活跃; 精确型指虚拟层能明确判断客户机内存页面的具体类型, 如是否正在使用, 是否为文件系统缓存, 是否为内核页面等。

KVM 和 VMware ESX Server(vSphere)平台的客户机内存管理属于估测型策略, 无法明确掌握客户机内存使用的状态, 前者将客户机当作普通进程管理, 后者通过采样的方式判断, 二者都只能估测客户机的内存使用情况。假设客户机中某程序刚释放了一些内存, 而物理机对此无法察觉, 暂时认为这些内存是活跃的而选择优先保留, 同时物理机的剩余内存又无法满足新的需求, 就会导致部分正在被使用(包括文件系统的缓存)但活跃度较低的内存数据先被交换到磁盘上。即便估测正确, 未使用内存的页面交换也会产生耗时的写磁盘操作。

而 z/VM 和 OpenVZ 分别以修改客户机系统内核和共享物理机内核的方式使物理机掌握客户机内存的实际使用情况, 属于精确型内存管理策略。这种做法有一个致命的弱点, 即虚拟机必须根据实际需求正常使用内存。特别是对外提供虚拟机服务(非合作型环境^[33])的情形下, 若大量用户发现客户机中诸如 Linux 的

mlock 系统调用的执行可将自己的内存尽可能多地在物理机中同步锁定(即使这些内存的读写频繁程度并不高),从而令自己的程序性能得到绝对保证,则可致物理机整体的内存超量使用特性退化,实际利用效率降低。可建模论证:设总物理内存为 x , 客户机中锁定内存量为 y , 总内存需求为 z , 其中 $y < x < z$, 设评价指标为共享系数 Q (即可共享物理内存占有所有须要共享内存的比例,该值越小越容易引起物理机操作系统抖动), Q_0 表示无锁定时的共享系数(即 $y=0$ 时), 则:

$$Q_0 = \frac{x}{z} \quad (2.1)$$

$$Q = \frac{x-y}{z-y} \quad (2.2)$$

显然, 只要 $y > 0$, $Q > Q_0$; 视 y 为变量, x 和 z 为常量, Q 单调递减。由此可得, 在须要超量使用内存的环境下, 被锁定的内存越少, 共享系数越高, 越不容易引起系统抖动。因此, 本设计认为不应当将客户机内部的内存锁定操作传递到物理机上。

为了解决上述各虚拟化平台在超量使用内存情况下的不同缺陷, 本文在 KVM 平台上作了修改, 设计了一种可将未使用内存回收的半精确半估测型新策略, 仅对客户机中未使用部分内存情况精确把握, 实现了客户机主动共享内存的机制。这种仅将有利于减轻物理机内存压力, 而对客户机中当前已占用内存的读写性能基本无影响的因素从客户机中传递出来的方式既避免了未使用内存参与物理机的页面交换线程和 KSM 线程的处理, 又不会导致虚拟层的内存管理盲目服从于客户机的内存操作。表 2.1 为本设计和现有虚拟化平台中内存管理特性的对比。其中, 精确或估测是指物理机对客户机内存状态的判断, 本设计既有对客户机中未使用的内存信息的精确把握, 又有 KVM 原有的对客户机内存页面访问频率的感知; 主动或被动指客户机是否主动退还不再使用的页面, 本设计以和 KVM 平台现有被动共享机制合作的方式实现先主动、后被动的共享内存, 在物理机出现较大内存压力前启动, 从而实现与现有的页面交换和 KSM 策略的合作, 减少它们的计算量。

表 2.1 各虚拟化平台内存管理特性对比

虚拟化平台	虚拟层对客户机内存状态了解的形式	客户机内存共享形式
KVM, VMware ESX Server(vSphere)	估测型	被动共享
z/VM, OpenVZ	精确型	先主动共享或占用, 后被动共享
本设计	半精确半估测型	先主动, 后被动共享

2.3 框架结构设计

为实现内存资源更高效的时分复用，避免“占而不用”现象的不利影响，本文设计了一种由客户机主动提出共享其中最无用部分内存的方案。由于客户机对内存页面访问的频率可以被物理机操作系统内核感知，客户机须要告知物理机关于内存使用情况最重要的信息就是那些曾经使用过，但当前不再使用的内存页面。主要的工作是回收客户机操作系统中被释放的内存，消除该部分内存对物理机内存调度算法的干扰，减少了页面交换对宝贵的磁盘 I/O 资源的占用^[34]，降低内存合并操作的计算量。此外，特别须要注意的是，这里所说的客户机中的未使用内存是指客户机操作系统内核所见的空闲内存页面，而非普通应用程序所见的包含缓冲区和缓存的可用内存，两个概念的区别可以从像 Linux 系统的 free 命令中轻易发现。

2.3.1 KVM 虚拟机内存回收原理

Linux 操作系统中有两种类型的内存申请方式，分别对应 mmap 和 brk 这两个系统调用，mmap 用于将硬盘上的文件映像直接加载到内存中，常用于加载程序和文件，是缓存机制^[35]的重要环节；brk 所申请的内存为匿名映射的内存，用于应用程序动态申请内存的分配，与缓存无关，此类内存大量存在于各种应用程序的内存映射中，可用 pmap 命令查看指定进程对应的内存使用情况。本文只关注客户机中的这种匿名映射内存。各种应用程序的内存映射中匿名映射内存均有较大的占比，因其不可被缓存且变化幅度极大，从而说明了本设计的必要性以及适用场景为匿名内存使用量波动较大的情形，相关测试详见第四章。

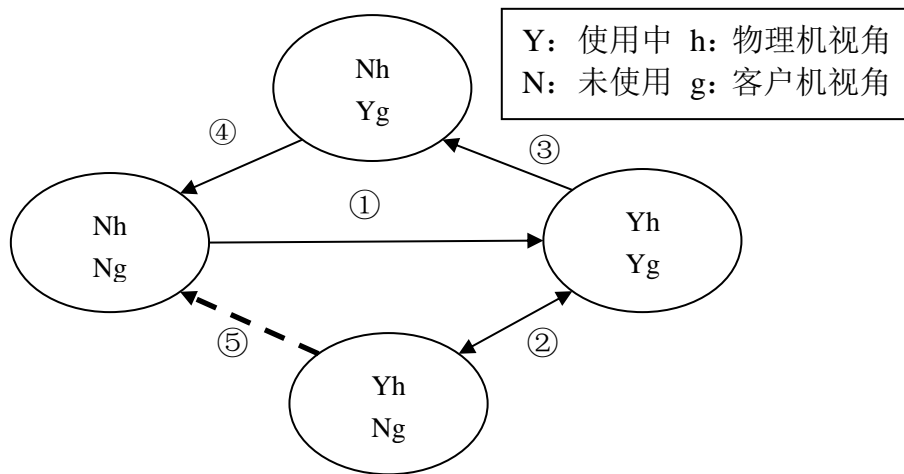


图 2.4 KVM 虚拟机内存状态转换图

虚拟机内存状态转换如图 2.4 中所示，内存的状态由两个字符组合表示，其

中 Y 代表内存页面正在被使用, N 代表未被使用, h 代表物理机内核的视角, g 代表客户机内核的视角。标号①为客户机首次使用某处内存; ②为客户机释放或再次使用此处内存; ③和④为客户机中气球驱动锁定某处内存, 并先后在物理机和客户机上释放此处内存; ⑤为本设计对未使用内存的回收过程。

2.3.2 回收 KVM 客户机主动共享内存的设计框架

本设计以 Linux 作为客户机操作系统。为区别客户机中曾经使用过的内存页面和首次使用(或图 2.4 中的{Nh,Ng}状态)的内存页面, 未使用内存回收的具体实现方式如下: 在客户机系统内核中的释放物理页面的位置记录所释放页面的区间, 基于效率的考虑, 当次释放后此处连续空闲空间(须为 Linux 内存伙伴系统^[36]联合的方式形成, 以避免区间重叠)达到所设的最大连续内存管理块才作记录, 若在切换出客户机前已记录的空闲内存块再次被使用则撤销先前的记录; 在物理机 KVM 模块中根据未使用内存区间的记录信息释放相应的内存页面, 并清除客户机作的记录。

在释放相应物理内存之前, 本设计首先用 down_write 函数对该区间内存执行锁定操作, 出于安全方面的考虑, 释放物理页面前先对该区域内存进行清零, 在释放操作后及时更新物理机的快表, 最后用 up_write 函数解除对该区域内存的锁定, 从而实现了与其他超量使用内存策略的兼容。

图 2.5 为回收客户机主动共享(未使用)内存(Initiatively-shared Memory Reclaim, 下缩写为 IMR)策略的基本框架。客户机主动共享的未使用内存由 KVM 模块直接回收, 重新交给物理机操作系统内核内存管理单元的空闲链表中, 不再参与物理机内核的内存调度, 客户机中未主动共享的(使用中的)内存可与主动共享的(未使用的)内存进行状态互换, 该过程的粒度可静态设定。

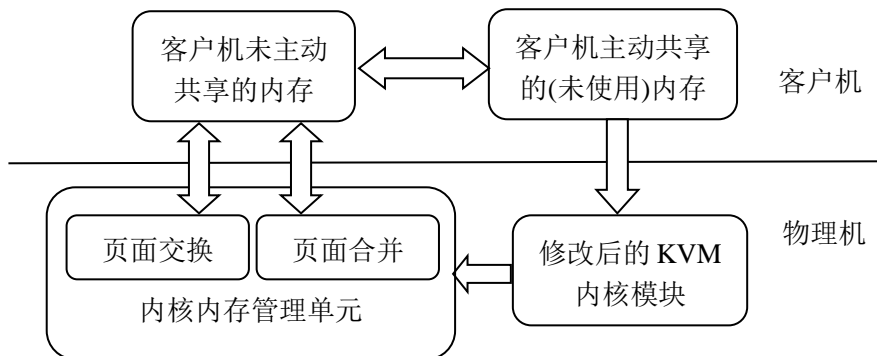


图 2.5 客户机主动共享内存管理基本框架

对于客户机中内存状态转换的记录信息有如下控制机制, 从而保证客户机与物理机切换时记录信息与实际内存状态的一致性, 避免出现错误释放的情况:

(1) 在客户机物理内存释放后在相应标志位作记录；

(2) 在客户机物理内存被使用前若相应标志位已有记录，则将该记录清除。

此外，为了获得更好的可用性，除了兼容现有的其他内存超量使用策略外，只要原系统支持，本文设计还可兼容客户机内存热插拔技术和嵌套虚拟机技术。

2.4 对比 KVM 其他内存超量使用策略及建模

2.4.1 与页面交换策略的比较

页面交换是物理机操作系统本身自带的一种内存管理策略，将每个虚拟机实例视为一个普通进程统一调度管理，无须对虚拟层代码作任何改动。当物理机有内存压力时，不论是否为正在使用的内存，都仅依据 LRU 算法作页面交换，而 IMR 策略无须访问磁盘，只对客户机中未使用部分的内存进行释放。虽然未使用的内存存在 LRU 算法中一般因较长时间不被访问先被交换出，但刚被释放成为未使用内存的部分也可能被优先保留，降低了 LRU 算法的效率。

表 2.2 所示为二者的对比，IMR 策略可消除部分磁盘操作，帮助页面交换策略减少计算量。

表 2.2 对比页面交换策略

	物理机操作	客户机读写缺页
页面交换	先写磁盘，然后释放物理内存	读磁盘，建立页表
IMR	释放物理内存	建立页表

对比页面交换策略建模：

设释放物理页面的时间开销为 $T_{freemem}$ ，将内存页面清零的时间为 T_{wmem} ，缺页读取磁盘的时间开销为 T_{rdisk} ，将内存页面写入磁盘的时间开销为 T_{wdisk} ，在超量使用内存时产生新空闲内存页面的平均额外时间代价记为 T_e ，对于未使用部分内存页面，采用页面交换策略和 IMR 策略的额外耗时分别记为 $T_{e(swap)}$ 和 $T_{e(IMR)}$ ，被交换的内存页面发生读写时的缺页创建页表时间为 $T_{e(paging)}$ ，则：

$$T_{e(swap)} = T_{wdisk} + T_{freemem} + T_{rdisk} + T_{e(paging)} \quad (2.3)$$

$$T_{e(IMR)} = T_{wmem} + T_{freemem} \quad (2.4)$$

公式(2.3)中 T_{rdisk} 和 $T_{e(paging)}$ 是原客户机内存页面被交换到磁盘的后续代价，公式(2.4)中不存在此情况。鉴于磁盘的读写速度差异不大，可将磁盘的访问用时统一表示为 T_{disk} ，则 $T_{e(swap)}$ 可记为：

$$T_{e(swap)} = 2T_{disk} + T_{freemem} + T_{e(paging)} \quad (2.5)$$

由于客户机读取未使用内存的情况几乎没有意义,出现的概率可以忽略不计,因此对客户机未使用的内存主要考虑写缺页的情况。关于磁盘访问、内存页清零和释放物理内存用时的差异可参考第四章的测试数据,进而得出:即使对要释放的内存执行清零操作, $T_{e(swap)}$ 也一定远大于 $T_{e(IMR)}$ 。

2.4.2 与 KSM 策略的比较

KSM 策略是物理机通过合并客户机中相同内存页面回收重复内存的行为,只须在物理机上开启配置 KSM 选项,就有一个名为 ksm 的守护进程开启,不断扫描检测内存页面重复情况。当物理机有内存压力时,不论在客户机中内存是否正在使用, KSM 线程均对其进行扫描并作合并操作,合并后若再有写入操作须重新申请占用内存。相比 IMR 策略, KSM 所需计算量较大,处理速度较慢。表 2.3 所示为二者的对比,在处理无效页面时可减少 KSM 策略的计算量。

表 2.3 对比 KSM 策略

	KSM	IMR
计算资源消耗	较大	极小
处理速度	较慢	极快
处理无效重复页面	可消除重复页面占用, 但须保留副本	清除所有无效内存页 面,无须保留副本
处理有效重复页面	同上	无操作
处理无效不重复页面	仍须扫描计算,但无法 减少内存占用	清除所有无效内存页 面
处理有效不重复页面	同上	无操作

对比 KSM 策略建模:

因为 IMR 和 KSM 策略侧重的情况不同,前者回收客户机中未使用的页面,后者回收客户机中重复的页面,这两种策略仅在处理无效且有重复的页面时才具有可比性。

为尽量达到相同的内存回收量,假设客户机中有若干内容完全相同的未使用内存页面,则本方案和 KSM 策略最终的内存回收量几乎相等。本方案的额外耗时仍然为 $T_{e(IMR)}$, KSM 策略合并内存的额外耗时记为 $T_{e(ksm)}$,该值除包含基本的释放内存时间 $T_{freemem}$ 外,还要包括合并内存的时间 T_{merge} ,以及以概率 P_{cow} ($0 \leq P_{cow} \leq 1$) 发生的写时复制操作作用时 T_{cow} ,即:

$$T_{e(ksm)} = T_{freemem} + T_{merge} + P_{cow}T_{cow} \quad (2.6)$$

其中 T_{merge} 至少为毫秒级,其余两项只须确定非负即可。由此可得即使在最

有利于 KSM 策略的冗余内存环境下, $T_{e(ksm)}$ 同样远大于 $T_{e(IMR)}$ 的结论。测试数据详见第四章。

2.4.3 与内存膨胀策略的比较

内存膨胀策略须在物理机和客户机上分别编写虚拟层代码和客户机驱动。不论物理机是否有内存压力, 用户均可手动指定客户机释放一定量的内存。

表 2.4 对比内存膨胀策略

	内存膨胀	IMR
灵活性	手动指定客户机释放	自动判断客户机内无效页面
精确性	可优先释放未使用的内存和缓存, 但若需释放内存值过大会导致客户机发生页面交换	仅释放未使用部分内存, 不消除客户机中的缓存, 除非客户机自行释放缓存

表 2.4 为二者的对比, 内存膨胀策略的灵活性和精确性有限, 甚至可能出现副作用, 虽然相比 IMR 策略处理客户机缓存的效果更好, 但由于内存膨胀策略因客户机内存实际需求变化各异^[37], 虚拟机监控器无法判定客户机操作系统适合释放内存的量, 该策略须手动执行。尽管业界也曾尝试过动态控制内存气球的方案, 但仍然是只是阶段性的执行内存膨胀的过程^[38], 此二者逻辑完全不同, 所以可比性不强。

2.5 本章小结

本章首先针对 KVM 平台内存资源的管理方式作了详细的分析, 研究了客户机操作系统的内存在物理机上的影响, 并针对现有的内存管理策略无法高效合理地处理客户机中未使用的内存提出了多种设计方案的假设和论证, 找到了一个可行的思路, 提出了本文的核心设计框架并对其原理进行了分析, 最后对一些现有的内存超量使用策略作了分析, 并对比了本设计方案, 同时在这个对比过程中提出了基于微观内存页面处理的数学模型, 论证了本设计方案的改进效果。

第三章 客户机空闲内存页面标记与回收

3.1 修改客户机操作系统的方式

由于涉及到对客户机物理内存页面的管理问题，本设计必须修改客户机操作系统内核，从而记录客户机操作系统中物理内存页面的使用情况，以便外部的虚拟机监控器进行处理。又因为内存管理代码对性能要求极高，虚拟层必须以直接读写内存的高效方式尽可能地减少额外开销。

本文设计了两种修改方案的数据结构，第一种是直接修改客户机操作系统内核，以平铺的方式记录物理内存块的使用信息，即以连续的比特位表示客户机上被管理的内存块是否须要被释放，称之为平铺记录法；第二种是以修改客户机系统内核或在客户机上安装驱动的形式，将客户机操作系统内核中关于空闲物理内存页面的链表信息传达到虚拟机监控器上进行处理，称之为链表记录法。

3.1.1 平铺记录法

平铺记录法的记录方式很简单，以二进制的 0 或 1 代表某客户机物理内存页块是否须要被释放，每个页块的大小为 2 的整数次幂字节，可根据需要决定基本页块的大小，大块可用多个比特位表示。考虑到访问权限的因素，该信息记录必须存储在客户机中，并且解决与虚拟层对该信息的读写同步问题。

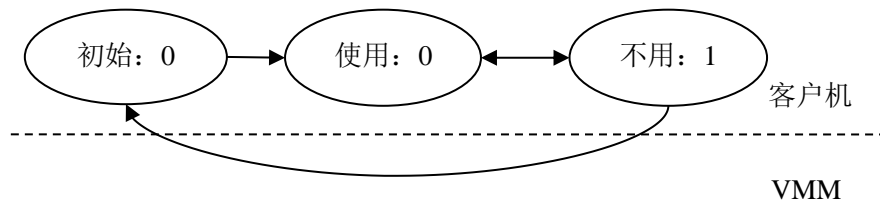


图 3.1 记录信息状态转换图

信息记录数据的同步机制的状态转换如图 3.1 所示，其中，从使用状态到不用状态是在客户机中释放该内存页块后置 1，并保证置位过程直接写入内存，不被客户机操作系统的虚拟处理器缓存，从不用状态到初始状态的过程须先锁定要释放内存的区域，以免与其他内存管理策略产生冲突。

这两种信息记录法的差异在于平铺记录法的效率略高,仅涉及比特位的简单读写,但其可管理客户机内存的最大值随控制粒度的提高而减少,而差量链表记录法因只记录须要释放的客户机内存块,可管理客户机内存量取决于客户机中同时可被释放的未使用内存量。即平铺记录法中比特位为 1 的数量较少时,同等情况下差量链表记录法对应的差量链表结点数量较少;平铺记录法中相邻比特位为 1 且符合 Linux 伙伴系统要求的情况较多时,同等情况下,差量链表记录法可在较高幂级的链表中合并记录一些相邻块的信息。

不论以哪种方法记录,从访问权限和信息读写速度要求的角度看,客户机空闲页块记录信息必须位于客户机中。然而,这些信息记录同样会占用一定的内存空间,在具体实现时还必须考虑这些内存占用和客户机操作系统启动时的内存页块占用情况。

对于平铺记录法,因为有内存页块大小的设定,所记录内容虽然十分紧凑,内存占用量也不大,但会影响该记录信息所在整个内存页块在虚拟层上表现为占用状态,导致一些邻近的客户机空闲物理内存页面无法被处理。为尽量避免信息记录对本设计的影响,这里本文的解决方案是在客户机 Linux 操作系统的 DMA 区(Direct Memory Access,直接内存存取,物理内存地址 0~16MB)处申请用于记录信息的内存,并且不再对该区域作任何检测,一方面是因为该处内存的线性地址到物理地址的转换十分简单,另一方面是因为该区域内存往往为各种驱动程序所用,需要较高的访问效率^[39]。在客户机操作系统的启动阶段内存管理部分初始化之后,可根据需要申请物理内存页面,每个比特位代表客户机上一块连续物理内存,该内存块大小最大可为 4MB(32 位 Linux 操作系统上),也可以是 2MB、1MB、512KB 等若干次减半后的页块(但不低于 4KB 的最小内存页面大小),但粒度越细,对应比特位的数量越多,处理这些比特位的存储和计算量越大。

对于链表记录法的差量链表记录方式,信息记录的方法和平铺记录法基本相同,但相比之下它的优势在于以下几点:第一,虚拟层可更高效地仅对差量链表中的结点页块进行处理,避免了平铺记录法中不论记录数值代表是否可以进行释放操作都进行读取判别。第二,由于是以差量链表的形式存储,这种方法不必像平铺记录法那样计算从客户机物理地址到对应比特位位置的转换。第三,差量链表的记录形式可很容易地表达客户机物理内存页面可被释放的优先级。

当然,不论哪种方案都会导致客户机再次使用内存时的请求时间增加,但本文的设计主要针对超量使用内存的环境,可在物理机操作系统出现内存压力时启动本策略。

3.1.4 空闲内存页块释放优先级及其影响

平铺记录法可通过仅处理大块物理内存页块的方式避免客户机操作系统对小块物理内存页面的请求和释放的抖动造成额外开销,这种方法很难控制不同空闲内存页块可被释放的优先级,仅依靠隐式的扫描顺序决定位置较靠前的优先释放,而差量链表记录法可进一步根据物理机内存压力的情况决定从客户机中有序释放未使用但占用物理机内存的页块,其内存页块的还原优先级的控制粒度更细,如图 3.3 所示,虚线方向代表优先级的从高到低。

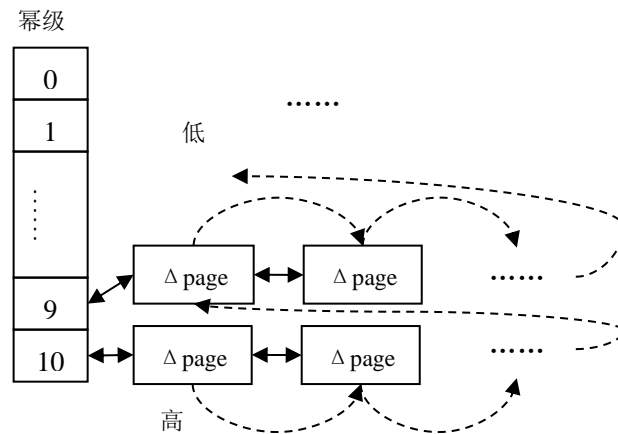


图 3.3 差量链表记录法内存页块释放优先级

因为客户机的 Linux 操作系统本身会在某些情况下以不同的方向在空闲内存双向链表中插入刚被释放的物理内存页块,例如在检测到所在更高一层的内存伙伴页块为空闲时,会将当前被释放的页块插入到所在等级的空闲内存双向链表的尾部,尽可能地延迟该空闲块的再次被使用,从而增加该空闲块被合并成更大内存页块的可能性。这样就可以直接利用客户机 Linux 操作系统现有的伙伴系统代码,避免在虚拟机监控器中建立伙伴系统^[40]。

3.2 具体实现与算法描述

本文实验环境详见 4.1 节。以下修改对象均指切换至 Linux 源码根目录后的下级目录与文件。

3.2.1 修改客户机操作系统内核

在客户机操作系统启动阶段,添加信息记录块本身所需内存的开辟代码,在客户机内存 DMA 区申请,禁止被客户机内存管理单元交换到硬盘上,特别须要

注意的是不能缺少 `volatile` 关键字，否则将会出现写入该信息记录块时被缓存，然后切换到物理机上物理机无法读取写入的记录。关于自定义的 `set_mem` 函数将在算法描述中作详细介绍。表 3.1 为客户机操作系统内核代码主要的修改。

表 3.1 客户机主要代码修改

文件: init/main.c	域: 全局
<code>volatile void *mem_block;</code>	
文件: init/main.c	域: <code>kernel_init()</code>
<code>mem_block=(volatile void __force*)get_zeroed_page(GFP_DMA __GFP_NO_KSWAPD);</code>	
文件: mm/page_alloc.c	域: 全局
<code>extern volatile void *mem_block;</code>	
文件: mm/page_alloc.c	域: <code>__free_one_page()</code>
<code>set_mem(page,order,1);</code>	
文件: mm/page_alloc.c	域: <code>__rmqueue_smallest()</code> 、 <code>__rmqueue_fallback()</code>
<code>set_mem(page,current_order,0);</code>	
文件: mm/page_alloc.c	域: <code>split_free_page()</code> 、 <code>__offline_isolated_pages()</code>
<code>set_mem(page,order,0);</code>	

3.2.2 修改物理机 KVM 内核模块

物理机 KVM 内核模块的修改中不再须要添加 `volatile` 关键字，但须要加上“`__user`”以表示所访问空间已不在内核空间。在客户机切换到物理机 KVM 模块时加锁保护释放过程，客户机中的信息记录块决定其可释放物理内存区间，自定义的 `range_kfree` 函数中特别须注意刷新快表等操作。表 3.2 为物理机 KVM 内核模块代码主要的修改。

表 3.2 物理机 KVM 内核模块主要代码修改

文件: include/linux/kvm_host.h	域: struct kvm
<code>u8 __user *p_mem;</code>	
文件: virt/kvm/kvm_main.c	域: <code>kvm_create_vm()</code>
<code>kvm->p_mem = NULL;</code>	
文件: arch/x86/kvm/vmx.c	域: <code>vmx_handle_exit()</code>
<code>vcpu->kvm->p_mem=(u8 __user *)gfn_to_hva(vcpu->kvm,0xc00);//定位信息记录块首地址</code> <code>down_write(&vcpu->kvm->mm->mmap_sem);//加锁保护</code> <code>range_kfree(vcpu->kvm->mm->mmap,start,len);//释放客户机在物理机上对应的内存</code> <code>up_write(&vcpu->kvm->mm->mmap_sem);//解锁以便被内核内存管理单元访问</code>	

3.2.3 相关算法描述

上述客户机操作系统内核中关于空闲物理内存页块的标记函数 `set_mem` 对应平铺记录法和差量链表记录法有两种不同的实现。下面就这两种方式的算法细节作详细描述。

平铺记录法的特点是客户机上的信息记录速度极快，影响效率的关键在于物理机上读取比特位的环节。在客户机中作记录时，比特位的设定取决于客户机的内存伙伴系统，可一次性置位连续多个比特位，若客户机最小内存管理块为 512KB，1 个字节可以存储最大的 4MB 内存页块，但在物理机读取时，可一次读出 32 位或 64 位数据。为提高效率，一般选择物理机为 64 位操作系统，则一次读取的比特位对应 32MB 客户机物理内存。为减少底层代码中栈的切换频率，应尽可能统一地释放对应客户机连续物理内存，即在 64 位区间内和相邻区间之间查找连续的 1。该算法伪代码如下所示：

```
物理机信息处理优化算法：
start ← end ← flag ← 0 //初始化
while start,end < total_length //设定总内存范围
    for i ← 0 to 63 //64 位区间内遍历
        if flag = 1
            end ← find next 0 //flag 为 1 时寻找下一个 0 位
        else
            start ← find next 1 //以下一个 1 位为起点
            end ← find next 0 //寻找下一个 0 位
        if i < 63
            record(start,end) //找到定位区间
            flag ← 0
        else //连续的 1 位超过 64 位区间范围
            flag ← 1
    get next 64 bits
if flag = 1
    record(start,end) //最后一次定位，尾部不超过总内存范围
```

差量链表记录法的特点是在物理机上读取记录时直接以链表的形式获得全部须要释放的内存块，效率很高。但是仍然因为执行权限的因素，不能在物理机上直接执行客户机内部的普通链表的释放代码，所以只能以静态链表的方式达到在物理机上删除客户机中信息结点的目的。为了避免对可用空闲链表结点的遍历过程，这里须另设一个可用空闲结点链表。信息记录方面并不涉及任何一个独立的算法，而是建立在客户机操作系统内核本身的算法之上，可做到根据客户机自身的判定决定被释放内存页面的优先级，例如客户机的空闲内存块链表的尾部插

入一块刚被释放的内存块，在差量链表中同样从尾部插入代表该块信息的结点。对于该方法中物理机上控制的释放优先级的伪代码描述如下所示：

```

客户机内存块释放优先级算法：
for order ← max_order to min_order
    while list[order] not empty
        if host under memory pressure
            free first  $2^{\text{order}}$  Bytes in list[order]

```

3.3 内存碎片的影响

上述两种解决方案均存在一个比较困难的问题——内存碎片^[41]。内存碎片是各种操作系统内存管理部分不可避免的一个问题，它对内存使用的性能可能会产生较大的影响。Linux 操作系统中主要以分配物理内存页面位置方面的优化达到尽可能减缓其不利影响的目的。从数据结构的角度分析，Linux 很早就开始使用伙伴系统的方法管理物理内存，伙伴系统是将大块物理内存不断以平分的方式分裂出小块内存的一种策略。这种管理机制的最小页块为 4KB，其目的是尽可能的减少内存的外部碎片，但由于内存页面使用后的释放时机不可控，外碎片依然存在，从 2.6.24 版本开始的 Linux 内核对此作了改进，它根据页框的移动性对空闲物理内存页块链表进行了划分管理，设定了不可移动页、可回收页、可移动页等迁移类型。其中不可移动页对内存的外碎片影响最大，这些内存页面往往是像内核的核心分配那样的会持久占用型内存，在申请时直接获得物理内存，而非像可移动页面(主要是用户空间的内存页面)那样的存在从线性地址到物理地址的转换层，以便内核认为在必要时自动减缓外部碎片的碎片化程度。

内存的外部碎片是无法避免的，对于本文的设计而言，会影响到最后的还原效果，对于这个问题，两种方案都有相应的对策。在平铺记录法中可以通过减小内存块单元的大小的方式使控制的粒度更细，从而进一步增加可被释放内存的量。在差量链表记录法中的办法是按优先级从高到低释放相应的内存页块，当其中一个幂级的差量内存页块处理完后，进入下一幂级的差量链表继续处理。当然，这两种方式在处理小块可释放内存页块时都会因为小块数量太多而增加很多计算量，再加上 Linux 中小块物理内存的申请使用量极高，经测试发现，大部分为最小的 4KB 物理内存页块，根据第四章中表 4.6 的测试数据，设定控制块的大小为 1MB 或 0.5MB 为宜。

3.4 本章小结

本章首先针对 KVM 平台内存资源的管理方式作了详细的分析, 研究了客户机操作系统的内存在物理机上的影响, 并针对现有的内存管理策略无法高效合理地处理客户机中未使用的内存提出了多种设计方案的假设和论证, 找到了一个可行的思路, 然后对一些现有的内存超量使用策略作了分析, 并对比了本设计方案, 同时在这个对比过程中提出了基于微观内存页面处理的数学模型, 论证了本设计方案的改进效果。最后介绍了本设计的两种具体实现以及其中的数据结构、处理优先级和应对潜在的内存外部碎片的方案。

第四章 性能测试与分析

为了测试本设计对比非超量使用内存环境的性能损耗,以及量化上述对比页面交换和 KSM 策略模型中的变量,本文设计了以下实验,文中本设计方案仍简写为 IMR。

4.1 实验环境

表 4.1 所示为本实验的测试环境。环境一和环境二代表影子页表和扩展页表(EPT)这两种不同的虚拟机内存页表策略。环境二主要用于测试扩展页表方式下缺页创建页表的时间开销,并以此说明不论哪种页表机制对本设计都不会有太大影响。环境二中 2GB 内存的配置用于宏观内存使用实验。

表 4.1 实验环境

	环境一(影子页表)	环境二(EPT)
CPU	Intel E8400	Intel i5 3550
内存	2GB DDR3-1066	16GB DDR3-1600
硬盘	320GB 7200rpm (缓存 16MB)	1TB 7200rpm (缓存 64MB)
客户机内存	1GB	1GB/2GB
KVM 版本	0.14.1	1.0
物理机操作系统	Ubuntu11.10 (内核 3.5.3)	Linux Mint 13 (内核 3.4.2)
客户机操作系统	Ubuntu11.10 (内核 3.6.1)	Debian 6 (内核 3.4.2)

4.2 实验结果及分析

实验一：各种应用程序匿名内存用量分析

测试目标：1、各常见应用程序未加载文档时匿名内存用量和总内存用量；2、匿名内存存在计算和数据库类标准测试中的使用特点；3、分析 IMR 的适用场合。

实验二：与非超量使用内存环境的对比

测试目标：1、缺页建立页表的额外时间消耗；2、已在内存中的数据读写操

作的额外时间消耗。

实验三：与页面交换和 KSM 策略的对比

测试目标：1、磁盘的读写时间；2、页面交换对物理机 CPU 阻塞情况的影响；3、KSM 内存页面合并时间。

实验四：IMR 策略效果分析

测试目标：1、客户机内核中标记所需额外耗时；2、KVM 内核中回收内存页面的平均耗时；3、物理机硬件内存本身的写入速度；4、本策略释放内存的时间和内存外碎片的影响。

实验五：IMR 策略下宏观内存使用测试

测试目标：1、在多个虚拟机启动和动态内存申请释放时，物理机内存和交换区空间用量；2、IMR 提升整体效率的分析。

(注：所有测试结果均为单个内存页面的平均用时)

4.2.1 各应用程序匿名内存用量分析

表 4.2 中选择了三种不同源程序语言(C/C++/java)编写的开源软件,虽然各种程序语言库的内存管理的具体实现各异,但底层对应匿名内存页面占比较大的特征是一致的。表中各应用程序均未加载任何文档。

表 4.2 各种应用程序匿名映射内存情况

	Gedit	Firefox	LibreOffice Write
匿名虚拟内存(KB)	30856	192312	81336
总虚拟内存(KB)	72864	275120	208728

从表中不难看出各种应用程序均有较大的匿名虚拟内存占比, Gedit、Firefox 和 LibreOffice Write 在未加载文档或网页时匿名虚拟内存占比分别为 42.35%、69.90%和 38.97%。

文档经 mmap 系统调用加载到内存并不涉及匿名内存的使用。匿名内存的使用主要用于计算类问题,例如对一段由 malloc 函数开辟的匿名虚拟内存空间写入大量数据进行数值计算,但这些数据在计算完成后没有继续存在的意义,须要用 free 函数进行释放,而 KVM 客户机中的这类行为就是本文的研究重点。

为进一步理解匿名内存使用的特点,下面将使用 sysbench 测试工具作说明。图 4.1 以如下命令执行 CPU 测试。

```
sysbench --test=cpu --cpu-max-prime=10000 run
```

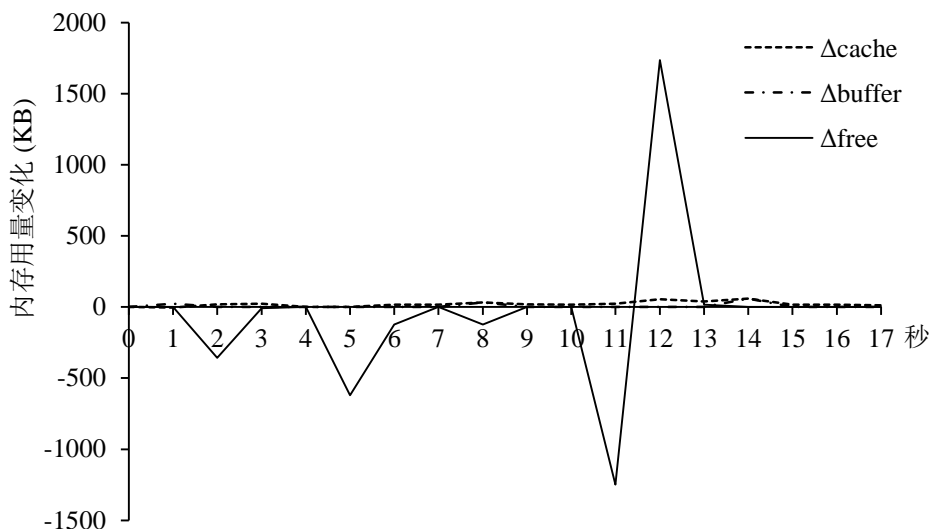


图 4.1 计算环境中各种内存使用情况

实际的计算过程从第 4 秒开始, 到第 12 秒结束, 由于匿名内存的使用, free 类内存减少($\Delta free$ 在 0 下方), 而 cache(缓存)和 buffer(缓冲区)几乎没有发生变化。测试结束后, 因为匿名内存被释放, free 值迅速增加。然而, 其他测试, 例如数据库方面的测试, 主要为文件存取等涉及文件系统缓存或缓冲区的操作, 匿名内存使用的比例不那么明显。

表 4.3 是以下测试命令的部分记录。此记录表明, 三种类型的内存均存在一定比例的使用情况。

```
sysbench --test=oltp --mysql-db=test --mysql-user=root --mysql-password=000000
--oltp-table-size=10000 run
```

表 4.3 数据库 OLTP 测试中的内存使用情况

时间顺序(s)	Free(KB)	Buffer(KB)	Cache(KB)
n	85980	678804	893680
n+1	91660	680480	895604
n+2	95956	678868	892136
n+3	103100	679396	884992

4.2.2 对比非超量使用内存环境

在客户机中向一段线性地址写入 10^8 个整型随机数(4×10^8 字节), 然后用 C++ 标准模版库中的 sort 函数对其排序, 分别记录写随机数(测试缺页建立页表)和排序(测试已在内存中的数据操作)的耗时。上述操作再重复执行若干次取平均值作为非首次执行结果。

实验结果如表 4.4 所示, 随机数写入项在回收与不回收未使用内存情况下测

试结果数值之差除以缺页的次数可得平均每次缺页创建页表的额外用时。

表 4.4 各种客户机内存页面操作时间

	影子页表			EPT		
	首次缺页	非首次缺页		首次缺页	非首次缺页	
		无 IMR	IMR		无 IMR	IMR
随机数写入(秒)	2.7	2.1	2.6	1.55	1.39	1.54
缺页创建页表(微秒)	6.1	0	6.1	1.6	0	1.6
排序计算(秒)	24.56	24.57		21.47	21.48	

影子页表和 EPT 这两种页表机制的效率差距较大，但不论如何，在缺页创建页表时二者都还只是微秒级的操作，在本设计与其他内存超量使用策略对比时影响有限，故后面的微观对比测试一般仅选择性能较差的环境一作实验环境。另外从排序计算的时间可见，对于已在内存中的数据的数据的读写操作基本不受未使用内存是否被回收的影响。

4.2.3 对比页面交换策略

在客户机系统内核中嵌入代码，重新编译并加载内核，计算缺页从磁盘读取先前交换出去的内存页面的阻塞时间，多次测量取平均值。

在环境一和环境二中，由于磁盘性能的差异，测得的平均缺页读取磁盘时间(T_{disk})分别为 6 毫秒和 2 毫秒(这里的差异主要来源于硬盘本身所带缓存的区别)。相比之下，IMR 策略应被认为是在高内存压力环境下有效提高内存效率的方式。

若未使用内存不被回收，客户机非首次申请该处内存不会重新创建页表。若将未使用内存回收，则非首次缺页创建页表耗时与首次时很接近，但物理机缺页访问磁盘的概率将减少，在高内存压力下回收未使用内存策略的整体表现更好。

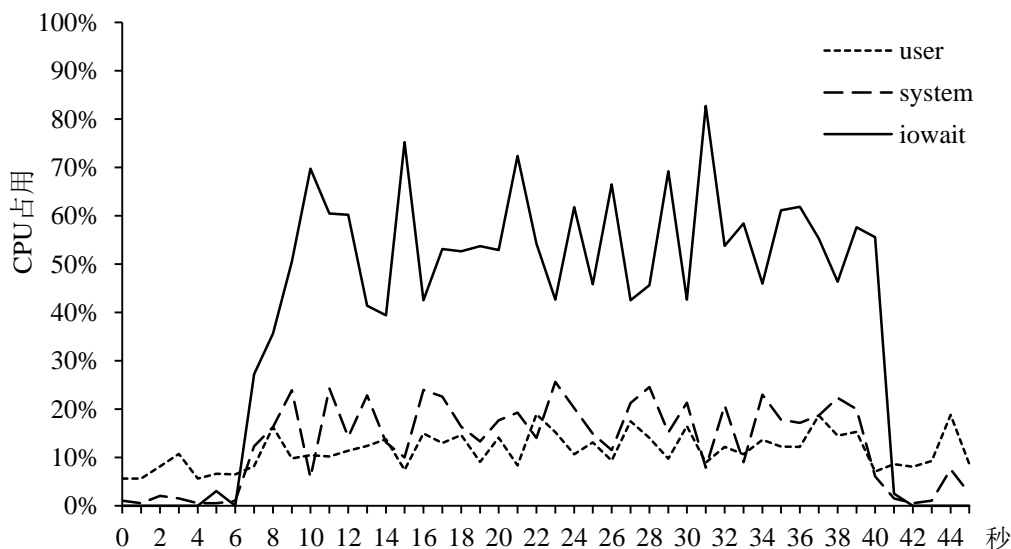


图 4.2 页面交换对 CPU 使用的影响

为了进一步说明该问题，本文对 Linux 操作系统页面交换情况下的 CPU 阻塞情况作了记录。如图 4.2 所示，在物理机出现内存压力时，处理器因页面交换过程引发 I/O 阻塞，且该项占比极高，一般持续时间较长，所以说尽可能地减少页面交换可有效提高计算密度。

4.2.4 对比 KSM 策略

在环境一客户机中写入大量重复或随机数据，然后释放相应的匿名内存页面，观测对比 KSM 线程和 IMR 策略的处理效果。其中重复数据和随机数据均为 4×10^8 字节(约 381.5MB)。

为确保 KSM 的执行基本不受客户机上测试程序外的内存数据影响，在客户机上先申请 4×10^8 字节线性地址空间，重复写入字符 ‘a’ 填满此地址空间后退出此测试程序以制造大量完全重复的内存页面，然后等待 KSM 线程启动合并操作，当原客户机操作系统和刚写入的字符 ‘a’ 都被处理至客户机进程内存用量稳定，合并过程结束。然后再在客户机上启动新的测试程序，再申请 4×10^8 字节的线性地址空间，为避免前一次合并的影响，重复写入另一字符 ‘b’，测试物理机上 KSM 线程合并重复数据的效果；或启动另一写随机数的测试程序，测试物理机对随机数据的处理效果。此外，本测试过程未引起物理机的页面交换操作。

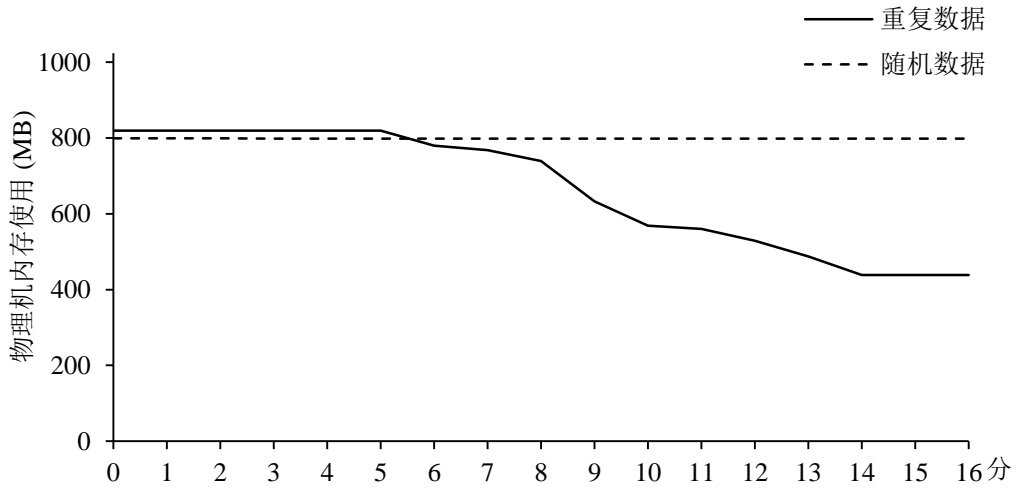


图 4.3 KSM 处理重复或随机数据的效果

由图 4.3 所示，在数据大量重复的最佳情况下，KSM 共合并约 380.9MB 数据，耗时 14 分钟，平均每页合并时间(T_{merge})为 8.6 毫秒，之后客户机内存用量基本不变，而随机数据环境下始终几乎没有数据被合并。虽然在较新版本的 KSM 中内存页面合并的速度有了十分明显的提升^[42]，特别是对于这种数据完全重复的最佳情况，整个合并过程一旦开始，同样的测试过程，结果显示只需若干秒即可，但是一方面检测到重复的数据仍然需要一些时间且平均到每个被合并页面的

时间仍然远大于 IMR 策略，另一方面这种极端的数据重复程度一般情况下不会出现，所以说，KSM 策略仍然存在相当的局限性。

KSM 对完全重复的数据和随机数据的处理分别体现了最好和最坏的处理效果。对于完全重复的数据，此测试案例合并了 99.85% 的内存，而随机数据的测试案例中仅 0.07% 的内存被合并。而不论是重复数据还是随机数据，KSM 线程都要长时间消耗处理器资源。

4.2.5 IMR 策略微观测试

在环境一中针对完全重复数据和随机数据作测试，重复数据量和随机数据量仍然均为 4×10^8 字节。设定最大连续内存管理块为 4MB。

本文中描述的平铺记录法和差量链表记录法在标记时间复杂度上均为 $O(1)$ ，因为前者可直接计算出特点内存页面对应的标志位置，后者采用静态链表的形式管理差量结点信息和剩余可用记录空间。二者在时间上略有差异，在客户机上，平铺记录法每次标记(一个页块)的额外耗时为 8.7 纳秒，链表记录法每次标记的额外耗时平均为 9.4 纳秒，差别微乎其微；在物理机上，由于 CPU 高速缓存的加速，即使平铺记录法须逐位判定是否进行释放，二者在读取标记时的差别也不大，所以这里的测试结果统一给出。测试结果如表 4.5 所示。

表 4.5 IMR 策略回收未使用内存测试结果

	重复数据	随机数据
起始内存用量(MB)	865	882
最终内存用量(MB)	525	530
耗时(秒)	0.049	0.051

重复数据和随机数据的作为未使用内存回收时分别回收了 89% 和 92% 的原有内存释放量，差别不大，但相比 KSM 策略的计算时间，在除处理使用中的冗余数据外的情况下 IMR 策略优势十分明显。 $T_{e(IMR)}$ 由表 4.5 中的耗时项的值除以总共处理的页数，结果为 0.56 微秒，而环境一的内存写入速度经 sysbench 软件测定为 7218.89MB/s，即平均每页(4KB)写入用时(T_{wmem})为 0.54 微秒。因此， $T_{freemem}$ 约为 0.02 微秒。

本方案因受 Linux 内核内存碎片影响不能完全回收客户机中释放的所有内存。若将内存管理块的大小设为 0.5MB，经测试发现可回收超过 99% 的可回收内存，内存碎片的影响被明显削弱。另一方面这些暂未回收的内存碎片可在下次被客户机操作系统使用时继续使用，一定程度上避免了客户机再次向物理机申请

小块物理内存的开销，满足客户机操作系统对小块物理内存的频繁需求。不同最大连续物理内存管理块在回收可释放客户机物理内存时的效果如表 4.6 所示，测试内容为 4×10^8 字节随机数据的客户机匿名内存页面回收。

表 4.6 内存管理块大小对回收效果的影响

内存管理块	4MB	2MB	1MB	0.5MB
回收前(MB)	882	879	876	887
回收后(MB)	530	511	501	509
回收百分比	92.3%	96.5%	98.3%	99.1%

4.2.6 IMR 策略宏观测试

为更好理解 IMR 对减少整体额外开销的贡献，本文设计了以下实验说明其效果。

- (1) 依次启动环境二配置(EPT 虚拟页表, 2GB 内存)的客户机至登陆界面;
- (2) 依次在每个客户机中启动动态内存测试程序, 该程序将申请 1GB 的虚拟内存空间, 然后在其中填满随机数, 并立即释放该内存。

从图 4.4 和图 4.5 可得, 开启 IMR 时客户机在系统启动阶段少消耗约 25%(未启用 IMR 时, 单个客户机消耗约 120MB 内存; 开启 IMR 后, 系统启动过程仅需约 90MB 内存)的内存。而动态内存的测试中, IMR 策略可回收几乎所有被释放的匿名内存。由于 KSM 合并页面的影响, 测试中的内存增长不一定是完全线性的。

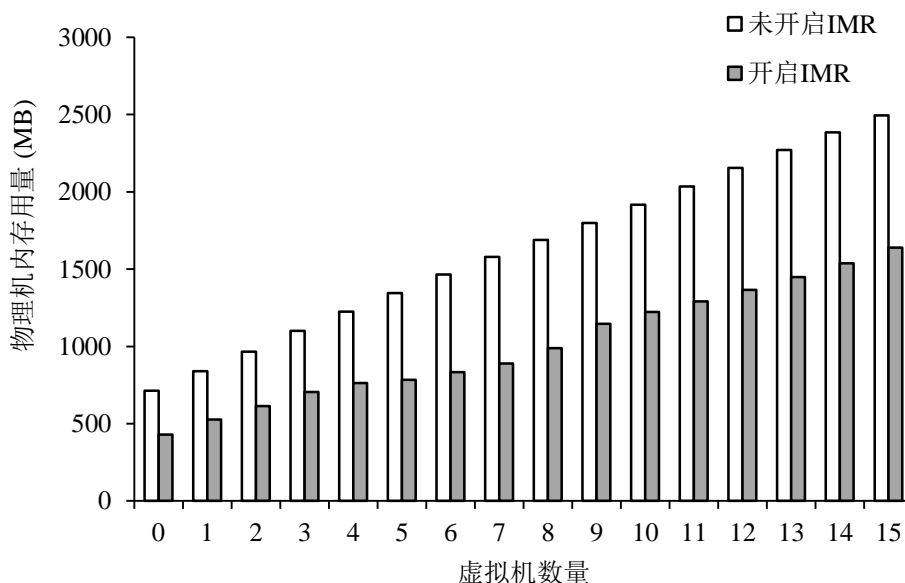


图 4.4 系统启动时内存使用情况

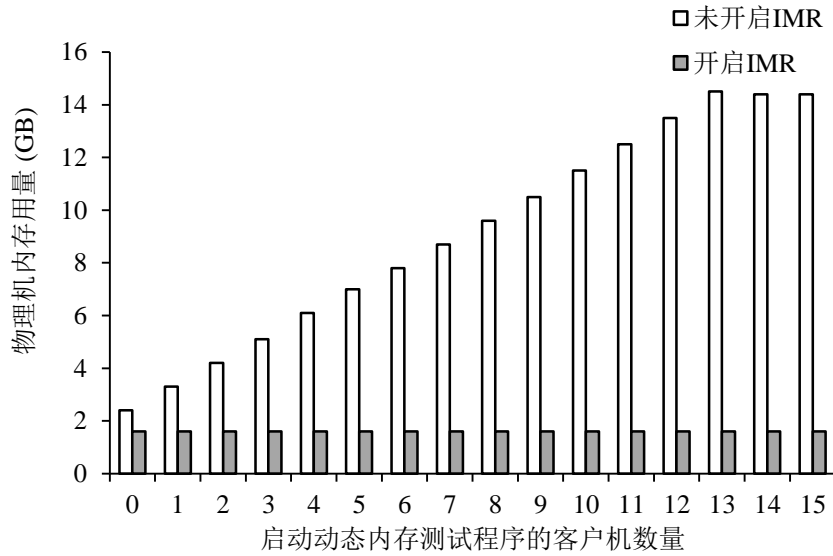


图 4.5 动态内存使用实验

未开启 IMR 环境中，最后两次客户机中内存使用的申请导致前面两个客户机被虚拟机监视器自动强制关闭。物理机内存用量的增长和客户机中内存申请量的差额由交换机制填补，此实验过程中交换区空间的使用情况如图 4.6 所示，开启 IMR 后，因远未达到物理机内存压力位，该项交换区空间用量一直为 0。

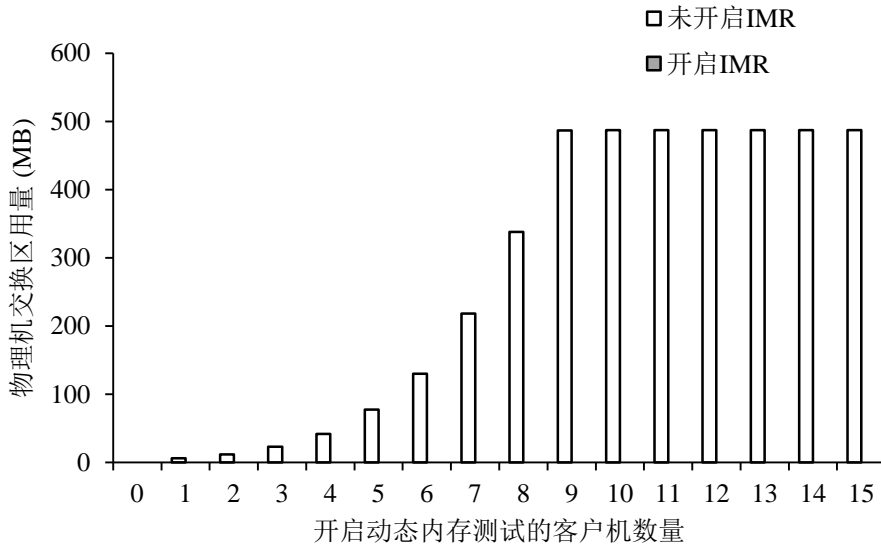


图 4.6 动态内存测试中物理机交换区使用情况

4.3 本章小结

本章简单介绍了本设计的实验软硬件平台，设定了两个主要的测试方向，即与非超量使用内存平台的对比和与其他超量使用内存的策略的对比，设计并实现了严格的测试，宏观上以物理机 CPU 的 I/O 阻塞时间百分比得出了避免页面交

换的重要性，并作了客户机启动和动态内存使用这两个测试案例，微观上测试了两种页表机制的效率差异，缺页读取磁盘的平均时间，最佳状态下 **KSM** 的平均页面合并时间，本设计的平均每页释放时间，以及减小内存管理块对内存外碎片的削弱情况。

第五章 总结与展望

5.1 工作总结

虚拟化技术使服务器系统的计算、存储、网络的管理方式发生了巨大的变革，它实现了硬件资源的软件化管理，一定程度上解放了人们对硬件平台的管理，集中了硬件计算资源的使用，避免了大量低负载服务器的出现，有效提高了计算密度，降低了整体能耗和空间需求。在单台物理机上扩展出多个操作系统的方式提供了硬件资源灵活分割的方式，极大的便利了用户使用、管理和扩展或收缩计算规模。然而，由于硬件资源的规模不论多大，始终是有限的，分配给虚拟机使用的各种硬件资源必然须要统一调度，又因为一些虚拟机对某些硬件资源的利用率可能很低，在虚拟层调度过程中可能出现超量使用的情况。最典型的就是内存这种硬件资源，读写较少的内存页面在允许超量使用内存的虚拟化环境中往往会被交换到硬盘上临时存储，再次被访问时才被重新读入内存中，本文针对内存虚拟化进行了深入研究，主要工作如下：

第一，本文详细调研了 KVM 虚拟化平台的内存管理，针对 KVM 客户机中的匿名映射内存特点作了重点研究。

第二，提出了让客户机主动共享其中不再使用的物理内存页面，通过信息记录传递的方式告知物理机回收相应内存页块，达到尽可能减缓物理机内存压力的目的。

第三，通过实验测试的方法分析论证本策略的效果，重点对比了其他超量使用内存策略，微观模型中对比页面交换策略和最佳情况下的 KSM 策略的回收内存页面效率均提升了三个数量级，还分析了本策略相比内存膨胀策略的优势在于可动态判定可回收内存的量。

因为宏观的内存性能测试存在过多不可控的影响因素，本文主要采用了间接测试 CPU 的 I/O 阻塞时间百分比和对微观的内存页面活动建模并测定相应操作的时间消耗的评估形式，主要针对匿名内存设计了一些宏观分析实验，尽可能地排除不可控因素的干扰。

相比其他超量使用内存的方式，IMR 策略的操作所需计算时间最少，返回内存给宿主机的速度最快。但由于现代操作系统存在缓存的因素，IMR 策略的效果一定程度上受限于客户机操作系统处理缓存的策略，但对匿名映射页面的释放的处理效果十分理想。

Linux 操作系统默认情况下在程序退出后会缓存部分内存页面，若该程序再次运行，则可不必要再次从磁盘中加载这些已被缓存的文件，充分利用了空闲内存，提高了效率。但当系统中内存占用率很高时运行一个不在缓存中而内存消耗较大的程序，则须先按一定策略释放一些缓存页面。

对于物理机或不超量使用内存的虚拟机来说，操作系统对退出程序的缓存是非常高效的管理策略，且已释放成未使用页面的内存不会导致其他虚拟机在需要用内存时出现内存不够用的情况。但如果虚拟机软件允许超量使用内存，由于宿主机对客户机中内存使用情况一无所知，误把未使用的页面当作正常使用的，在内存超量使用时就要额外处理这些未使用的内存页面，降低了效率。另外，对于客户机上只运行一次的程序和退出后再次启动时间间隔较长的程序所对应的内存，本应优先释放，但因为宿主机无法区分(在不改动客户机的前提下)，造成一些正常使用的页面可能先被宿主机置换。

对于以上问题，内存膨胀技术可以有效应对，但它须手动执行，不灵活、不精确，可能导致客户机内出现内存压力，从而引起不必要的客户机页面交换。

虽然各种超量使用内存的策略各有侧重，但 IMR 策略代价最小，速度最快，且有助于减少所有其他策略的计算量，应优先启用。

5.2 研究展望

本文在 KVM 虚拟化平台下对内存资源的超量使用管理策略进行了研究，提出了让内存资源在虚拟化环境中实现具有类似于时分复用特性的目标，通过修改客户机操作系统内核和物理机上 KVM 内核模块的方式，实现了将客户机不再使用的内存信息传达到物理机上，同时利用现有的页面交换和 KSM(KSM 合并相同内存页面的过程可变相实现一般操作系统加载程序时避免内存数据的重复性，只是效率上比较低)策略达到接近操作系统级虚拟化技术的内存管理特点，而各个虚拟机仍然可以拥有不同的操作系统内核。

未来的研究方向可以面向以下几个方向：

第一，进一步改进对内存外碎片的控制，或修改 Linux 内核本身，减少其本身碎片化的程度，或设法让客户机的内存外碎片管理与外部物理机操作系统内核协作，达到物理机上实现统一处理内存外碎片机制的目的。

第二，可考虑通过新增处理器硬件虚拟化指令的方式^[43]，类似于直接 I/O 访问的虚拟化技术那样，让虚拟机的内存访问直接绕开虚拟机监控器，从而提高虚拟化平台整体内存的使用效率。

致 谢

光阴荏苒，转眼间我的研究生学习生涯即将结束。在读研究生的这两年半时间里，实验室的各位老师和同学给我留下了难忘的记忆，离别之际，在此向所有关心、帮助、指导过我的老师和同学致以衷心的感谢和祝福。

首先，我要衷心感谢我的导师万健教授。万老师从我开学刚进入实验室开始，对我的学习和生活给予了极大的支持和鼓励。万老师治学态度严谨、学识渊博、工作热情饱满、性格平易近人，令我深深折服。特别珍惜与万老师见面交流探讨学术问题的机会，也感谢他对我批评指正，给我提出的许多宝贵意见，帮助我在学习和生活方面快速进步，在此谨向我的恩师由衷地表示感谢！

还要感谢蒋从锋副教授。在蒋老师的悉心指导下使我不仅学到了关于虚拟化技术的专业知识，更掌握了科学的研究方法、严谨的论文规范以及为人处事的道理，他使我的研究工作质量得到了迅速的提升。在硕士论文的选题、开题、撰写和修改过程中，蒋老师给予了我极大的关心、指导和帮助。在此，我向蒋从锋老师表示崇高的敬意和深切的祝福！

感谢实验室在我攻读硕士学位期间给我提供了良好的硬件环境，让我在学习和科研过程中能够拥有便利的实验设备。同样也感谢任祖杰老师、殷昱煜老师、张伟老师、张纪林老师以及实验室的其他老师对我的关心和教导，使我增长了不少知识和历练。他们不遗余力地指导和严格的科研管理，使我对研究课题有了更为深入的理解，对所遇到的难题有了更多的改进和完善的方法，对我完成学业和学位论文起到了十分重要的作用。特此论文完成之际，谨向我的恩师们表示由衷的感谢！

此外，还要感谢段良成、刘知俊、朱圣代、颜佳伟、刘恩益、陈恂、应俊、宋智波等同学，以及实验室的其他与我共同学习研究和生活的师兄姐妹们，让我感受到了学习和科研的乐趣。在此，我还要感谢徐向华教授在我研究生阶段前期对我的指导。

最后，特别感谢我的父母在方方面面给予我最大的理解、支持和信赖，你们对我一直以来的支持、鼓励让我有了攻克难关的坚定信心和强大动力，我一定不辜负你们殷切的期望。

参考文献

- [1] Huang W, Liu J, Abali B, et al. A case for high performance computing with virtual machines[C]. Proceedings of the 20th annual international conference on Supercomputing. ACM, 2006: 125-134.
- [2] Mendel Rosenblum , Tal Garfinkel, Virtual Machine Monitors: Current Technology and Future Trends, Computer, v.38 n.5, p.39-47, May 2005
- [3] Kantarci B, Foschini L, Corradi A, et al. Inter-and-intra data center VM-placement for energy-efficient large-Scale cloud systems[C]. Globecom Workshops, 2012 IEEE. IEEE, 2012: 708-713.
- [4] Bhardwaj S, Jain L, Jain S. Cloud computing: A study of infrastructure as a service (IAAS)[J]. International Journal of engineering and information Technology, 2010, 2(1): 60-63.
- [5] Tom á s L, Tordsson J. Improving cloud infrastructure utilization through overbooking[C]. Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference. ACM, 2013: 5.
- [6] Sotomayor B, Keahey K, Foster I. Overhead matters: A model for virtual resource management[C]. Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing. IEEE Computer Society, 2006: 5.
- [7] Buyya R, Abramson D, Giddy J, et al. Economic models for resource management and scheduling in grid computing[J]. Concurrency and computation: practice and experience, 2002, 14(13 - 15): 1507-1542.
- [8] Marshall P, Keahey K, Freeman T. Elastic site: Using clouds to elastically extend site resources[C]. Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. IEEE Computer Society, 2010: 43-52.
- [9] Corradi A, Fanelli M, Foschini L. VM consolidation: a real case based on OpenStack Cloud[J]. Future Generation Computer Systems, 2012.
- [10] Garfinkel T, Pfaff B, Chow J, et al. Terra: A virtual machine-based platform for trusted computing[C]. ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 193-206.
- [11] Habib I. Virtualization with kvm[J]. Linux Journal, 2008, 2008(166): 8.
- [12] Wang X L, Sun Y F, Luo Y W, et al. Dynamic memory paravirtualization transparent to guest OS[J]. Science in China Series F: Information Sciences, 2010, 53(1): 77-88.
- [13] 陈昊罡, 汪小林, 王振林, 张彬彬, 罗英伟, 李晓明. DMM: 虚拟机的动态映射模型. 中国科学: 信息科学, 2010, 40: 1543-1558.

- [14] Ke Y. Intel virtualization technology overview[J]. Intel System Software Division, 2009.
- [15] Virtualization A M D. AMD-V Nested Paging[J/OL]. White paper. <http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx>, 2008.
- [16] van Doorn L. Hardware virtualization trends[C]. VEE. 2006, 6: 45-45.
- [17] Bellard F. QEMU, a Fast and Portable Dynamic Translator[C]. USENIX Annual Technical Conference, FREENIX Track. 2005: 41-46.
- [18] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 164-177.
- [19] Abels T, Dhawan P, Chandrasekaran B. An overview of xen virtualization[J]. Dell Power Solutions, 2005 (8): 109-111.
- [20] Soltesz S, Pözl H, Fiuczynski M E, et al. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors[C]. ACM SIGOPS Operating Systems Review. ACM, 2007, 41(3): 275-287.
- [21] Kivity A, Kamay Y, Laor D, et al. kvm: the Linux virtual machine monitor[C]. Proceedings of the Linux Symposium. 2007, 1: 225-230.
- [22] Tang L, Mars J, Vachharajani N, et al. The impact of memory subsystem resource sharing on datacenter applications[C]. Computer Architecture (ISCA), 2011 38th Annual International Symposium on. IEEE, 2011: 283-294.
- [23] Russell R. virtio: towards a de-facto standard for virtual I/O devices[J]. ACM SIGOPS Operating Systems Review, 2008, 42(5): 95-103.
- [24] Kantarci B, Foschini L, Corradi A, et al. Design of energy - efficient cloud systems via network and resource virtualization[J]. International Journal of Network Management, 2013.
- [25] Heo J, Zhu X, Padala P, et al. Memory overbooking and dynamic control of Xen virtual machines in consolidated environments[C]. Integrated Network Management, 2009. IM'09. IFIP/IEEE International Symposium on. IEEE, 2009: 630-637.
- [26] Waldspurger C A. Memory resource management in VMware ESX server[J]. ACM SIGOPS Operating Systems Review, 2002, 36(SI): 181-194.
- [27] Arcangeli A, Eidus I, Wright C. Increasing memory density by using KSM[C]. Proceedings of the linux symposium. 2009: 19-28.
- [28] Schwidefsky M, Franke H, Mansell R, et al. Collaborative memory management in hosted linux environments[C]. Proceedings of the Linux Symposium. 2006, 2.
- [29] Kolyshkin K. Virtualization in linux[J]. White paper, OpenVZ, 2006.
- [30] Matthews J N, Hu W, Hapuarachchi M, et al. Quantifying the performance isolation properties of virtualization systems[C]. Proceedings of the 2007 workshop on Experimental

- computer science. ACM, 2007: 6.
- [31] Magenheimer D, Mason C, McCracken D, et al. Transcendent memory and linux[C]. Proceedings of the Linux Symposium. 2009: 191-200.
- [32] Kukreja G, Singh S. Virtio based transcendent memory[C]. Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on. IEEE, 2010, 1: 723-727.
- [33] Kong Z, Xu C Z, Guo M. Mechanism design for stochastic virtual resource allocation in non-cooperative cloud systems[C]. Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011: 614-621.
- [34] 张彬彬, 汪小林, 杨亮, 赖荣凤, 王振林, 罗英伟, 李晓明. 修改客户机操作系统优化 KVM 虚拟机的 I/O 性能. 计算机学报, 2010, 33(12): 2312-2320.
- [35] Yoon J, Min S L, Cho Y. Buffer cache management: predicting the future from the past[C]. Parallel Architectures, Algorithms and Networks, 2002. I-SPAN'02. Proceedings. International Symposium on. IEEE, 2002: 92-97.
- [36] Gorman M. Understanding the Linux virtual memory manager[M]. Prentice Hall, 2004.
- [37] Niu Y, Yang C, Cheng X. Dynamic Memory Demand Estimating Based on the Guest Operating System Behaviors for Virtual Machines[C]. Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on. IEEE, 2011: 81-86.
- [38] Hines M R, Gopalan K. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning[C]. Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 2009: 51-60.
- [39] Corbet J, Rubini A, Kroah-Hartman G. Linux device drivers[M]. O'reilly, 2009.
- [40] Wood T, Tarasuk-Levin G, Shenoy P, et al. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers[C]. Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 2009: 31-40.
- [41] Gorman M. Understanding the Linux virtual memory manager[M]. Prentice Hall, 2004.
- [42] Miller K, Franz F, Groeninger T, et al. KSM++: Using I/O-based hints to make memory-deduplication scanners more efficient[C]. Proceedings of the ASPLOS Workshop on Runtime Environments, Systems, Layering and Virtualized Environments (RESolve'12). 2012.
- [43] Wang X, Zang J, Wang Z, et al. Selective hardware/software memory virtualization[J]. ACM SIGPLAN Notices, 2011, 46(7): 217-226.

附录

作者在读期间发表的学术论文及参加的科研项目

发表的学术论文

1. Jian Wan, Wei DU, Congfeng JIANG. Memory Collection and Allocation through Cooperative Sharing in Multi-tiered Virtualized Servers. International Journal of Grid and Distributed Computing(Accepted).

参加的科研项目

1. 国家自然科学基金青年基金项目：多处理器计算系统能耗和服务质量约束的任务调度策略研究(No.61003077)，2011-2013
2. 浙江省自然科学基金项目：多虚拟机环境下磁盘 I/O 调度策略研究(No.Y1101092)，2011-2013

杭州电子科技大学

硕士学位论文 详细摘要

题 目: KVM 客户机主动共享的内
存超量使用策略研究

研 究 生 杜炜

专 业 计算机软件与理论

指导教师 万健 教授

完成日期 2013 年 12 月

详细摘要

随着云计算产业的快速发展,云服务器集群中的计算资源的管理已成为人们关注的重点。相比于传统的服务器集群,云计算通过虚拟化等技术改进服务器的软硬件资源的管理和分配的方式,提高了硬件资源的整体利用率和服务的可靠性。然而,虚拟化技术在应用于服务器资源管理的过程中也遇到了一些挑战,其中的挑战之一就是虚拟机内存资源分配的合理性。合理高效的虚拟机内存资源分配管理是保障云计算环境整体效率的关键之一。内存作为计算和存储的中间环节在虚拟化平台中显得尤为重要,进一步优化虚拟机内存管理将是提高虚拟化平台整体资源利用率的关键。

然而,由于现有的 KVM 虚拟化平台中内存管理的方式无法在物理机上分辨客户机内部内存页面的具体状态,客户机内存页面仅以被访问的频率和内容上的重复性的形式被物理机所感知,可能出现一些不合理的情况,例如客户机中刚被释放的物理内存页面被物理机认为最近被访问的而不被页面交换策略优先处理;KSM 策略在合并重复内存页面时无法预判被合并的页面多久后被写入新数据,或是客户机中几乎没有重复的数据。在客户机中安装驱动的内存膨胀策略的优点是遵循客户机内部调度,但存在无法较自动精确的控制释放内存量的缺陷。

本文提出了新的超量使用内存策略,即以修改客户机系统内核的方式使客户机主动提出与物理机共享其中未使用部分内存,以修改物理机上 KVM 模块的方式回收客户机愿意共享部分的内存,从而排除此部分客户机内存对物理机内存调度算法的干扰。

本策略可充分利用现有 KVM 平台的各种策略模拟操作系统级虚拟化技术的内存管理特点,保留物理机操作系统的页面交换策略,让其感知客户机操作系统的内存访问频率;以 KSM 策略合并客户机中重复页面的方式模拟操作系统级虚拟化中的加载程序或文件的去重特点;以本设计回收客户机中未使用内存的方式模拟操作系统级虚拟化中客户机程序将未使用内存释放回物理机操作系统内核内存管理单元;以本设计关于客户机操作系统内核物理内存页块优先级的处理模拟操作系统级虚拟化中统一的伙伴页块管理。

在处理客户机内信息记录与实际物理内存块状态的语义一致性时,本文充分考虑了客户机与物理机之间可能随时切换的特点,并提出了具体的解决方案;在物理机 KVM 内核模块处理客户机中的信息记录时,本文设计很好的处理了与其他内存超量使用策略的兼容性问题,并对连续信息记录的处理作了优化。

对于信息记录的方式，文中提出了两种不同的方式。其中平铺记录法由于可以统一处理跨伙伴页块的连续记录，处理全部记录时的效率较高，更适合有较大突发内存需求的环境；而差量链表记录法在处理信息记录时的效率略低，但适合在总体内存用量波动较小的场合，仅在达到所设阈值时才按内存页块记录的优先级回收部分客户机内存，使物理机内存用量降至所设阈值之下，尽可能地避免客户机多次向物理机请求内存。

在与其他现有的内存超量使用策略对比时，本文提出了基于微观内存页面行为的模型，随后展开介绍了两种不同的设计方案和内存外部碎片对本设计的影响，最后通过 KVM 虚拟化平台下模拟内存占用的实验论证了本设计可取得良好的效果。宏观测试实验部分从系统整体的角度对比了开启与未开启 IMR 策略的情况下，客户机启动与动态内存使用场景下物理机内存使用，以及物理机交换区空间使用状态方面的差别。

本设计主要适用于匿名内存使用量波动较大的计算环境，文中选择了几个主流开源软件进行匿名内存用量的测试，验证了本设计的基础；在基于微观页面行为的测试中对比了页面交换和页面合并策略，本策略在处理客户机空闲物理内存块的速度上相比页面交换(swap)的平均代价提高了三个数量级，相比页面合并(KSM)的最佳情形的效率也能提高三个数量级。

此外，为了获得更好的可用性，除了兼容现有的其他内存超量使用策略外，只要原系统支持，本文设计还可兼容客户机内存热插拔技术和嵌套虚拟机技术。