云计算平台中多虚拟机内存协同优化策略研究

张伟哲 张宏莉 张 迪 程 涛
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 虚拟化技术为云计算基础设施资源的动态部署、安全隔离提供了重要保证.从过度占用内存的虚拟机中回收内存,提供给内存紧缺的虚拟机使用,优化多虚拟机间的内存分布是内存虚拟化中的挑战性问题.文中引入了自发调节和全局调节协作的多虚拟机内存管理架构.通过定义内存资源充裕状态和内存资源紧缺状态,提出了自发调节和全局调节之间协作的算法.处于内存资源充裕状态时,各虚拟机可采用自发调节策略平衡内存资源.处于内存资源紧缺状态时,基于空闲内存价格的全局调节策略平衡内存资源.实验结果表明,针对计算密集型与存储密集型负载,该框架与算法均能很好地提高服务能力,同时具有较高的可扩展性与较低的性能惩罚.

关键词 云计算;虚拟化;内存协同优化;自发调节算法;全局调节算法
中图法分类号 TP314 **DOI 号**: 10.3724/SP.J.1016.2011.02265

Memory Cooperation Optimization Strategies of Multiple Virtual Machines in Cloud Computing Environment

ZHANG Wei-Zhe ZHANG Hong-Li ZHANG Di CHENG Tao
(*School of Computer Science and Engineering, Harbin Institute of Technology, Harbin* 150001)

Abstract The virtualization technology has provided the important guarantee for the dynamic deployment and security isolation of the infrastructure as a service (IaaS) in the cloud computing. It is a challenging problem to optimize, across time, the distribution of RAM among a maximal set of virtual machines by reclaiming memory from ones that have an excess of memory and providing it to others that need more memory. This paper proposes a memory optimization scheme for multiple virtual machines with the cooperation of spontaneous adjustment and global adjustment algorithms. In the memory resources abundant status and scarce status, various virtual machines may adopt the spontaneous adjustment strategy and the overall adjustment strategy to balance memory resources, respectively. The experimental results indicate that in view of the computation intensity and the memory intensity load, this frame and the algorithm can sharpen the serviceability well, simultaneously have the high extendibility and the low performance penalty.

Keywords cloud computing; virtualization; memory collaboration optimization; self-adjustment; global-adjustment

1 引 言

云计算作为一种创新的计算模式,近年来日益受到学术界和业界的重视.有别于以个人计算机为中心的传统计算模式,云计算通常以互联网为中心构建多个大规模数据中心,为用户提供资源按需租用的服务模式[1].根据服务层次不同,当前云计算服

收稿日期:2011-06-13;最终修改稿收到日期:2011-10-29.本课题得到国家自然科学基金(61173145)、国家"九七三"重点基础研究发展规划项目基金(2011CB302605)和国家"八六三"高技术研究发展计划项目基金(2010AA012504,2011AA010705)资助.**张伟哲**,男,1976 年生,博士,副教授,研究方向为网络计算、并行与分布式系统.E-mail: wzzhang@hit.edu.cn.**张宏莉**,女,1973 年生,博士,教授,研究领域为网络安全、网络计算.**张 迪**,男,1988 年生,本科生,研究方向为网络计算、并行与分布式系统.**程 涛**,男,1987 年生,硕士研究生,研究方向为网络计算、并行与分布式系统.

务模式可以分为以下 3 种[2]：(1) 基础设施即服务(IaaS). 外包用于支持操作的设备，用户按照自己的意志运行操作系统和应用软件等程序；(2) 平台即服务(PaaS). 通过网络提供操作系统和相关服务，用户采用提供商支持的编程语言与工具编写服务；(3) 软件即服务(SaaS). 应用程序由供应商或服务供应商托管，用户通过各种客户端设备的瘦客户界面使用这些服务.

基础设施服务(IaaS)的核心是系统虚拟化技术. 将某一个或多个数据中心的计算与存储资源虚拟化，形成一个高效灵活的资源池，可以帮助降低基础设施的成本、延缓数据中心扩建的时间，提升应对快速变化的业务需求能力[3]. 虚拟化技术包括对内存、CPU 和 I/O 设备等资源的虚拟化. 高效的 CPU 和 I/O 设备分时复用已经得到广泛的研究，但分时共享内存较难实现. 因此，多虚拟机物理内存分时共享正逐渐成为系统虚拟化的瓶颈，也成为当前虚拟机资源管理的研究热点.

当多个虚拟机部署在同一台物理主机上时，需要在虚拟机之间分配物理内存. 如果静态分配，即在虚拟机运行过程中所占用物理内存大小不发生变化，则物理主机上所能并发执行的虚拟机数目受到物理内存大小的限制. 此外，不同虚拟机中运行的上层服务对内存的需求各不相同且动态变化，静态分配必然会造成内存资源分配不合理，影响虚拟机执行效率. 文献[4-20]设计并实现了各种虚拟机动态内存平衡机制，然而当前研究存在如下问题：(1) Xen[4]、VMware[5] 和 KVM[6] 等虽然提供了气球驱动、页面交换和内存共享等机制来动态调整虚拟机的内存，但缺乏从全局角度进行多虚拟机内存协同管理的系统架构；(2) 多虚拟机间缺乏内存协同分配策略，当虚拟机内存充裕时系统应当从哪些虚拟机回收内存、回收多少内存、回收的内存优先分配给哪些虚拟机使用，这些仍然是开放的问题.

针对上述问题，本文首先提出了自发调节与全局调节协作的多虚拟机内存管理体系结构. 根据多虚拟机系统中所有虚拟机的内存状态信息，通过定义内存资源充裕状态和内存资源紧缺状态，依据多虚拟机系统所处的场景执行不同的内存调节策略. 其次，提出了自发调节和全局调节之间协作的算法. 自发调节在内存充裕状态生效，通过操作系统本身的统计信息获取内存使用信息，利用 Xen 提供的气球驱动机制调整分配给不同虚拟机的内存资源. 针对内存紧缺状态，提出了基于空闲内存价格的全局调节策略，结合客户操作系统的空闲内存值和空闲交换空间等信息，确定每个客户操作系统的空闲内存价格，通过平衡价格的方法达到内存资源的平衡.

本文第 2 节简要介绍与本文相关的研究工作，并分析与我们研究工作的异同；第 3 节阐述多虚拟机内存管理系统的体系结构；第 4 节重点介绍多虚拟机内存动态平衡算法，详细给出自发调节、全局调节和两者协作的算法；第 5 节给出实验平台设置和实验结果；最后，对全文进行总结并对未来工作给出展望.

2 相关工作

虚拟机内存优化的目的是通过掌握当前虚拟机内存使用情况以及预测虚拟机未来负载变化，从过度占用内存的虚拟机中回收内存，以提供给内存紧缺的虚拟机使用或用来启动新的虚拟机，在不导致服务性能严重下降的前提下，优化多虚拟机间的内存分布. 当前的内存优化策略主要分为页面复用技术、内存动态调整机制和多虚拟机内存平衡技术 3 类.

(1) 页面复用技术主要包括页面交换(host swapping)和页面共享(page sharing)技术. Cellular Disco 系统[7] 首先提出了将虚拟机的部分物理内存页面与宿主操作系统(host OS) 的交换磁盘分区交换，允许虚拟机使用超过实际机器内存大小的内存空间. Waldspurger[9] 与 Sugerman 等人[8] 分别在 VMWare ESX Server 和 VMWare Workstation 中实现了基于交换的虚拟存储技术. 文献[9]进一步提出了基于页面内容比较的虚拟机间内存共享技术，通过一致性 Hash 等方法，识别同一虚拟机或不同虚拟机间存储同样内容的页面，达到节约内存的目的. Gupta 等人[10] 通过将页面共享技术与页面压缩、页面补丁技术相融合，大幅度地提高了虚拟机内存资源的利用率.

(2) 内存动态调整机制主要包括气球驱动技术(balloon driver)和热插拔技术(virtual hot Plug). 气球驱动技术是内存动态调整的主流技术. VMware 首先提出了气球驱动机制[9]，通过回收某些虚拟机中未使用的内存页面，满足内存需求剧增的虚拟机请求. 虚拟机管理器 Xen 和 KVM 分别在各自系统中提供了对气球驱动机制的支持[11-13]. 虚拟热插拔在虚拟机内存波动时，通过对操作系统中内存管理接口的欺骗，伪装成运行时系统物理内存得到了增

加或缩减，从而使虚拟机地址空间具备了伸缩能力. 文献[14]详述了逻辑层虚拟热插拔的工作原理，此后文献[15]进一步剖析热插拔和气球驱动两种方法的优缺点.

(3) 多虚拟机内存平衡技术可以分为单台物理机和多台物理机间的内存平衡. 文献[16]提出了基于黑盒与灰盒的单台物理机内部多虚拟机内存平衡方法，Magenheimer[17]提出了基于 Xen 气球驱动机制的 self-balloon 策略，Zhao 等人[18]提出了基于缺页率曲线的虚拟机内存预测方法及内存分配策略. 文献[19]提出了虚拟机的动态内存映射模型(DMM)，将半虚拟化、影子页表和硬件辅助虚拟化等与虚拟存储、内存共享等融合起来. 文献[20]通过引入双层地址空间映射机制，构建了跨越多个物理机的虚拟机内存优化框架.

上述研究工作中页面复用技术[7-10]和内存动态调整机制[9-16]提供了内存动态分配的底层支持机制，但并没有解决内存在各虚拟机间何时分配、如何分配等问题，而文献[16-20]缺乏从全局角度进行多虚拟机内存协同管理系统架构和协同调度策略. 本文将对其进一步深入研究，提出新的高效解决方法.

3　多虚拟机内存动态管理体系结构

多虚拟机内存管理以多个虚拟机内存资源的动态自适应为目的，利用内存资源的使用情况等信息，决定给不同的虚拟机分配不同大小的内存，从而达到物理机器内存的最大有效使用和多个虚拟机内存资源均衡分配. 因此，多虚拟机内存管理系统应具备以下功能：(1) 监测物理机内存资源的使用情况，间歇性地获取正在使用的物理内存的大小和空闲内存大小，供内存调节决策使用；(2) 监测每个虚拟机中客户操作系统内存资源的使用情况，间歇性地的获取客户操作系统的最大可分配内存值、最小可分配内存值、正在使用中的内存值、空闲内存值、空闲交换空间值和系统所有应用提交给客户操作系统的内存值等内存信息；(3) 在内存资源充裕状态下，多虚拟机内存管理系统能够根据每个客户操作系统上的应用提交给系统的内存值，来调节客户操作系统的内存值；(4) 在内存紧缺状态下，能够从全局角度根据每个客户操作系统内存的使用情况和物理内存的情况，平衡每个客户操作系统的内存值. Domain0 相对于其他客户操作系统优先级较高，因此在首先保证 Domain0 内存使用情况下，再执行对其他客户操作系统的内存平衡策略. 另外，在进行内存调节的过程中，需要考虑内存调节引起的内存和计算能力的开销.

多虚拟机内存动态管理系统体系结构如图 1 所示，主要由内存信息采集(Information Collection，IC)、管理策略(Policy-Management，PM)、调节机制(Regualtor Module，RM) 3 个部分组成. 内存信息采集模块包括虚拟机监视模块(VMC)和物理机监视模块(PMC). 管理策略包含自发调节(Self-adjustment)和全局调节策略(Global-adjustment).

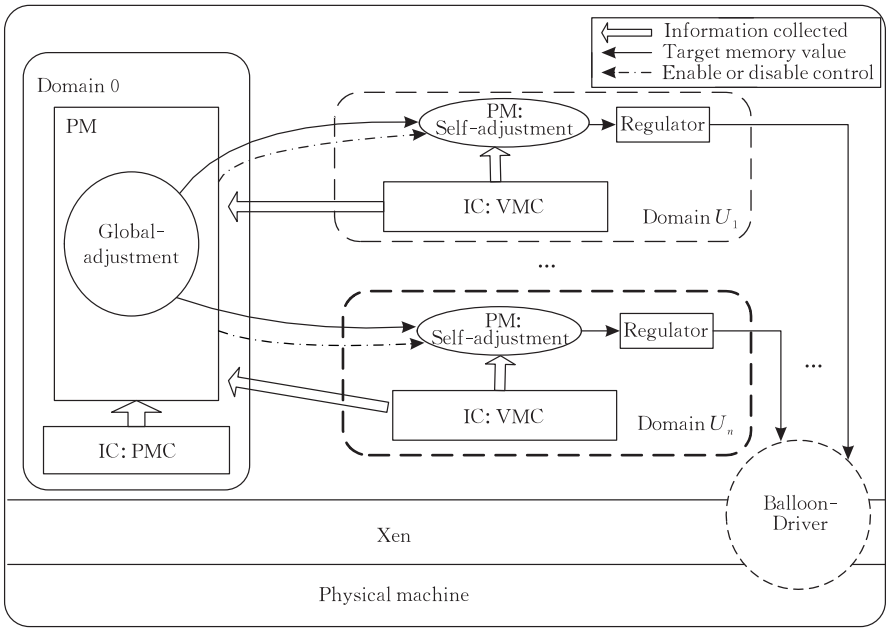


图 1　多虚拟机内存动态管理系统体系结构

内存信息采集模块负责内存使用信息的获取，供管理策略部分决策使用，是整个多虚拟机内存管理系统的基础.内存监视模块周期性地获取内存信息，并把内存信息发送给调节管理模块.内存监视模块分为虚拟机内存监视模块和物理机内存监视模块.虚拟机内存监视模块不仅可以获取分配给虚拟机的物理内存的使用情况，还可以获得每个虚拟机的交换空间的使用情况；物理机内存监视模块，可以获得物理机的总内存值和空闲内存值以及运行的所有客户操作系统的最大内存信息等.

管理策略模块是多虚拟机内存管理系统的核心，负责整个系统调节策略的制定和决策.根据按需分配的调节原则，在内存资源充裕的情况下，给每个客户操作系统分配充足的内存，调节工作由自发调节模块完成.在内存资源紧缺状态下，各个客户操作系统自发地调节已经无法满足按需分配原则，此时调节工作则由全局调节模块完成.全局调节模块决定从哪些客户操作系统中收回内存，向哪些客户操作系统分配内存，并计算出可回收和分配的内存大小，将给出的每个 Domain$U$ 的最佳内存值下发给实际调节部分.详细内容将在第 4 节介绍.

调节机制负责具体的资源调节工作，本文采用 Xen 中的 Balloon-Driver 机制作为内存调节机制，使用 XenStore 完成各部分间的数据交互.内存信息采集模块中的虚拟机监视模块(VMC)在 Domain$U$ 中通过轻量级 Daemon 实现，物理机监视模块(PMC)在 Domain0 中实现.全局调节模块(Global-adjustment)负责全局的调节，运行在 Domain0 中；自发调节模块(Self-adjustment)负责各个 Domain$U$ 内存资源的调节，运行在 Domain$U$ 中.

4 多虚拟机内存动态平衡算法

本节首先给出自发调节与全局调节的定义，而后阐述了两者协同工作算法，最后分别详细描述了自发调节和全局调节算法.

4.1 自发调节与全局调节

为了方便讨论，表 1 列出了虚拟机内存资源信息的形式化定义.在对自发调节和全局调节进行定义前，首先形式化地定义内存资源的状态.设 T_h 为物理机所有可用的物理内存的大小；F_h 为物理机上空闲内存的大小.集合 V 是所有客户操作系统的集合.对于每个虚拟机 $VM_i \in V$ 在启动时设定了一个最大内存值 H_i 和一个最小内存值 L_i.对每个虚拟机上运行的客户操作系统都有一个对未来的内存需求提交的预测值 C_i，受虚拟机最大和最小内存值限定，调整地提交内存值 $MC_i = \min(\max(L_i, C_i), H_i)$.设 N_i 为当前时刻分配给每个虚拟机的内存值，则 $\sum(MC_i - N_i)$ 为所有启动了自发调节服务的虚拟机需要的内存值的总和，该值为正值表示需要使用剩余的空闲物理内存，负值表示需要释放虚拟机多余的内存资源.

表 1 虚拟机内存信息符号表

符号	含义
H_i	最大内存
L_i	最小内存
C_i	提交内存值
MC_i	根据最大和最小内存值调整后提交的内存值
N_i	当前分配给虚拟机的内存值
Nt_i	下一时刻虚拟机目标内存值(不包含内存“红利”)
T_i	虚拟机最终目标内存值(包含“红利”)
F_i	当前虚拟机空闲内存值
A_i	处于活动状态的内存，等于 $N_i - F_i$
SF_i	空闲交换空间大小
ST_i	交换空间总大小

定义 1. 内存资源充裕状态.当 $F_h \geq \sum(MC_i - N_i)$ 时，物理机空闲内存资源可以满足所有虚拟机提交内存的需要，称此状态为内存资源充裕状态.

定义 2. 内存资源紧缺状态.当 $F_h < \sum(MC_i - N_i)$ 时，空闲的物理内存资源无法满足所有虚拟机提交内存的需要，称此状态为内存资源紧缺状态.

定义 3. 多虚拟机内存自发调节.若系统处于内存充裕状态，各虚拟机可根据客户操作系统提交的内存值自行调节，此种内存调节方式称为自发调节.

定义 4. 多虚拟机内存全局调节.当系统处于内存紧缺状态下，物理机内存资源已经无法满足多个虚拟机提交内存的需求了，出现内存资源需求竞争.此时，需要从多虚拟机全局的角度来考虑平衡调节策略，此种内存调节方式称为全局调节.

4.2 自发调节与全局调节的协同算法

自发调节和全局调节的区别在于自发调节对于每个虚拟机无需知道其它虚拟机的内存使用信息，只根据自身信息就可以决定分配多少内存资源.而全局调节需要掌握每个受控的客户操作系统的内存使用信息，从全局的角度执行平衡调节策略决定分配给每个虚拟机多少内存资源.全局调节依赖自发调节发送的虚拟机内存信息，自发调节受控于全局调节.

自发调节和全局调节之间的协作算法 1 如下所示，其算法的复杂性取决于从其它 Domain$U$ 中回收

内存资源和执行多个 Domain*U* 的全局平衡调节策略.

算法 1.　自发调节与主动调节协同算法.

```
输入：间隔时间 time
输出：无
Cooperate (time) {
  While (1) {
    调节 Domain0 内存；
    全局调节标志设置为假；
  If (空闲内存资源不够 Domain0 使用) {
    禁用自发调节；
    从其它 DomainU 中回收内存资源；
    全局调节标记设置为真；}
  If(全局调节标记为真 ‖ 内存资源处于紧缺状态) {
    禁用自发调节；
    执行多个 DomainU 的全局平衡调节策略；
  } else {
    启用自发调节；}
  Sleep(time)；
  }
}
```

首先需要考虑的是不同 Domain 之间的优先级关系. Domain0 主要负责 VM 的创建、管理、配置等工作，并完成设备驱动，相对于其它 Domain*U* 更加重要. 因此无论是在内存资源充裕状态，还是在内存资源紧缺状态下，分配给 Domain0 的调节内存值应始终为 Domain0 客户操作系统的提交内存值. 满足 Domain0 需求的前提下，再在各个 Domain*U* 之间进行全局调节. 内存资源充裕状态下，所有客户操作系统内存调节由自发调节机制完成；在全局调节模式被触发后，拥有自发调节服务的客户操作系统的自发调节服务将被禁用.

其次要考虑全局调节的触发时机. Domain0 优先级高于其它 Domain*U*，在无法满足 Domain0 内存资源需求时，需要从其它 Domain*U* 中回收内存，区别于只是在相同优先级的 Domain*U* 之间进行内存资源平衡；然而，通常情况下，并不在 Domain0 中运行计算或者服务程序，Domain0 的内存只是在有新的虚拟机启动或者有虚拟机关闭的时候才出现内存使用的明显变化. 因此，当能够满足 Domain0 内存资源需要后的一段时间内都不会出现第一种情况，即不需要从其它 Domain*U* 中回收内存资源. 这两种情况下系统都是处于内存资源紧缺状态. 综上，可以确定以下两种情况会触发全局调节模式：

(1) Domain0 无法满足提交内存需要时，需要从其它客户操作系统回收内存；

(2) 能够满足 Domain0 的需要，但内存资源处于紧缺状态.

4.3　自发调节策略

自发调节和全局调节模式在同一时间只能有一个处于启用状态，判定哪种模式处于启用状态和如何执行最终的内存调节是自发调节策略重点. 从 /proc 文件系统的 meminfo 中可以得到客户操作系统的内存信息，包括提交内存值 committed_AS；利用 Xenstore 作为客户操作系统和 Domain0 之间内存信息和控制信息传递的媒介；利用气球驱动机制作为调节客户操作系统内存的基础. 自发调节算法如算法 2 所示，算法复杂度为 $O(1)$.

算法 2.　自发调节算法.

```
子程序：调节到目标内存
/* type==1 为自发调节，其它非负整数都为全局调
   节目标内存值 */
输入：调节类型；
输出：无
Adjust_to_target (int type ) {
  If (type==1) {
    计算自发调节模式目标内存；
    目标内存值为自发调节内存；
  } else {
    目标内存值为全局调节内存值；}
  获取当前内存值；
  If (current >target) {
    根据上调比率计算目标内存；
  } else if (current <target) {
    根据下调比率计算目标内存；}
  调节当前内存到目标内存；
  }
自发调节：
输入：间隔时间 time
输出：无
Self (time) {
  通知全局调节自发调节启动；
  While (true) {
    根据配置和全局控制，
    判断自发调节状态；
  If (自发调节启用) {
    Adjust_to_target (1)；
  } else {
    获取全局目标内存 global；
    Adjust_to_target (global)；
  }
  向全局发送内存等状态信息；
  Sleep (time)；
  }
}
```

在自发调节的触发过程中需要注意以下问题：(1)自发调节启用需要通知主控模块.自发调节的启动时间的不确定性要求自发调节在启动后需要通知主控模块.若自发调节在启用后没有通知主控模块，则主控模块在进行调节时并不知道 Domain$U$ 自身已经作出了调节，会引发内存资源竞争；(2)自发调节是否启用开始时由自己控制，后受控于主控模块.自发调节是否启用在开始时由自身的配置文件决定，一旦全局调节改变了配置的状态，自发调节是否启用的状态只受控于全局调节.(3)对于客户操作系统来说，自发调节和全局调节同时只能有一个处于启用状态.也即对于每一个客户操作系统来说同一时间只受控于一种调节机制，不会出现调节冲突.

4.4 全局调节策略

在处于内存紧缺状态时，自发调节策略已不能满足需要.需要从全局视角对有限的内存资源重新动态分配.Waldspurger 等人[9]提出了按股份分配的思想：在按比例分配方案中，客户拥有的分配资源的权利称为股份，客户根据拥有的股份按比例地分配与回收系统资源.当内存资源紧缺时，系统从不完全使用其内存的客户中收回内存资源.空闲内存税表示从一个客户中收回空闲页面的最大比重.对于一个股份为 S 的客户，当前分配的内存页为 P，其中有 f 部分处于活动状态，每页股份率 ρ 为 $\rho=S/P(f+k(1-f))$.其中，$k=1/(1-\tau)$ 为空闲页开销，$\tau(0\leq\tau<1)$ 为空闲内存税.空闲内存税 τ 指定了可以回收多少空闲内存资源.当 $\tau=0$ 时，方法是纯粹的按比例分配方法，当 $\tau\approx1$ 时，所有的空闲内存资源都将被收回.然而该方法没有考虑交换空间对空闲内存税的影响，更重要的是该方法并没有提出定量计算多虚拟机间动态内存分配的策略.因此，本节提出基于空闲内存税的全局调节策略.

我们规定每个客户操作系统拥有的股份是相同的，设定每个客户操作系统的 $S=1,P=N_i$，处于活动部分的内存比率 $f=A_i/N_i$，可得该客户操作系统的每页股份率 ρ_i 为 $\rho_i=1/(A_i+k(N_i-A_i))=(1-\tau)/(N_i-\tau\times A_i)$.其中，$N_i$ 为该客户操作系统的当前内存值，A_i 为该客户操作系统处于活动状态的内存值，且 $A_i=N_i-S_i$.内存处于紧缺的客户操作系统会使用系统的交换空间，且使用率不同.考虑加入交换空间后每页股份率 ρ_i 如式(1)所示，ε 为交换空间使用率加权比，控制交换空间的使用情况在每页股份使用率评价中的比重.

$$\rho_i=(1-\tau)+\varepsilon\times(1-SF_i/ST_i)/N_i-\tau\times A_i \tag{1}$$

设 Nt_i 为下一时刻分配给客户操作系统的内存值，在物理机处于内存资源紧缺状态时，平衡多虚拟机的内存资源为平衡空闲内存的价格，即求下一时刻分配给客户操作系统多少内存资源可以保证多个虚拟机的空闲内存价格相等.由此可得方程组如式(2)所示：

$$\begin{cases}\dfrac{(1-\tau)+\varepsilon\times\left(1-\dfrac{SF_1}{ST_1}\right)}{Nt_1-\tau\times A_1}=\dfrac{(1-\tau)+\varepsilon\times\left(1-\dfrac{SF_2}{ST_2}\right)}{Nt_2-\tau\times A_2};\\ \dfrac{(1-\tau)+\varepsilon\times\left(1-\dfrac{SF_1}{ST_1}\right)}{Nt_1-\tau\times A_1}=\dfrac{(1-\tau)+\varepsilon\times\left(1-\dfrac{SF_3}{ST_3}\right)}{Nt_3-\tau\times A_3};\\ \cdots\\ \dfrac{(1-\tau)+\varepsilon\times\left(1-\dfrac{SF_1}{ST_1}\right)}{Nt_1-\tau\times A_1}=\dfrac{(1-\tau)+\varepsilon\times\left(1-\dfrac{SF_n}{ST_n}\right)}{Nt_n-\tau\times A_n};\\ \sum N_i=Nt_1+Nt_2+\cdots+Nt_n\end{cases} \tag{2}$$

设

$$\left((1-\tau)+\varepsilon\times\left(1-\frac{SF_i}{ST_i}\right)\right)\Big/\left((1-\tau)+\varepsilon\times\left(1-\frac{SF_1}{ST_1}\right)\right)=b_i,$$

其中 $i=2,3,\cdots,n$.

化简方程组(2)得

$$\begin{cases}b_2\times Nt_1-Nt_2=b_i\times\tau\times A_1-\tau\times A_2;\\ b_3\times Nt_1-Nt_2=b_i\times\tau\times A_1-\tau\times A_2;\\ \cdots\\ b_n\times Nt_1-Nt_2=b_i\times\tau\times A_1-\tau\times A_2;\\ Nt_1+Nt_2+\cdots+Nt_n=\sum N_i\end{cases} \tag{3}$$

由此，可得方程组的系数矩阵 $\boldsymbol{A}$(如式(4)所示)，$\boldsymbol{A}$ 是 $n\times n$ 的.

$$\boldsymbol{A}=\begin{pmatrix}b_2&-1&0&\cdots&0\\ b_3&0&-1&\cdots&0\\ \vdots&\vdots&\vdots&\ddots&\vdots\\ b_n&0&0&\cdots&-1\\ 1&1&1&1&1\end{pmatrix} \tag{4}$$

易求的系数矩阵 $\boldsymbol{A}$ 的秩为 n.又因为一个满秩方阵和其增广矩阵同秩.若对于 $\boldsymbol{Ax}=\boldsymbol{b}$ 线性方程组，矩阵($\boldsymbol{A}$)的秩与($\boldsymbol{A}$ $\boldsymbol{b}$)的秩相等，并且等于满秩，则方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 有唯一解.

全局调节算法如算法 3 所示.根据该算法可求得每个客户操作系统下一时刻分配的内存资源值 Nt_i.F_h 为物理机上空闲内存的值，可求虚拟机目标内存值 $T_i=Nt_i+F_h\times Nt_i\Big/\sum Nt_i$，其中 T_i 就是需

要求的每个客户操作系统的最终目标内存. 构造线性方程组的算法复杂度为 $O(n)$，求解线性方程组的复杂度为 $O(n^2)$，因此最好情况下算法复杂度为 $O(n^2)$，最坏情况下复杂度为 $O(n^3)$，其中 n 为受控的虚拟机个数.

算法 3.　平衡空闲内存价格的全局调节算法.

```
输入：当前内存值之和，所有虚拟机内存信息
输出：每个虚拟机的目标内存值
Balance_mem (total_cur) {
If (只有一个客户操作系统) {
  根据最大和最小内存，
  调整目标内存值；
} else {
  构造线性方程组；
  解线性方程组；
    N=0；
  While(未全遍历本次计算受控客户系统) {
    If(目标结果需要调整) {
      根据最大和最小内存，调整目标内存；
      total_cur－＝调整结果；
      N＋＋；}}
    If (N>0)
      Balance_mem (total_cur)；}}
```

5　实验结果及性能分析

本节实验分为 5 部分：第 1 部分通过实验测定自发调节与全局调节算法中空闲内存、空闲内存税率和交换空间使用率等关键参数；第 2 部分验证自发调节与全局调节算法的有效性；第 3 部分采用 Dacapo 和 SPEC CPU 2000 等标准测试程序，进一步验证多虚拟机内存管理系统的性能；第 4 部分和第 5 部分分析多虚拟机内存管理系统的开销与可扩展性.

实验环境采用 Intel 服务器和 Xen 3.0. 服务器为 8 核(2 个 Intel Xeon E5506 2.13GHz Quad-Core processors)，16 GB 双通道 1333 MHz 内存. 客户机系统为 linux2.6.16，为了避免 CPU 竞争，实验中给每个虚拟机分配一个 CPU 核心. 客户机上的守护进程使用 shell 编写. 全局调节使用 C 语言实现，调节每秒运行一次. 每个客户机的初始内存值为 214 MB，基线数据通过在没有内存信息收集和内存调节的环境下测得.

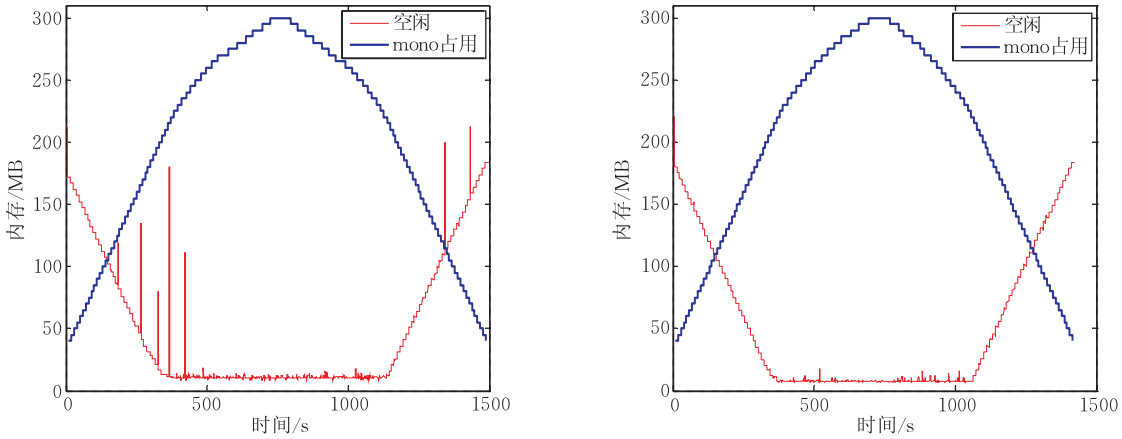
基准测试程序包括两个微内核基准测试程序(Dacapo[21]和 SPEC CPU 2000[22])和两个微内核程序(random 和 mono)[18]. random 运行时随机申请一个 r 大小的内存空间 $r\in[low,high]$，然后以确定的迭代次数随机地访问申请的内存空间，当迭代完成时，释放申请的内存空间并开始下一次重复操作. Mono 和 random 相似，但在每个阶段的访问中，mono 申请的内存空间的大小先从 low 到 $high$ 单调递增，然后再从 $high$ 到 low 单调递减. Dacapo 是一组 Java 测试标准集合，包括 10 个真实的计算机应用程序，其中一些程序计算需要很大的内存资源. 默认情况下，初始的 Java 堆空间大小为 50 MB，最大为 100 MB. SPEC CPU 2000 是标准性能评价公司(Standard Performance Evaluation Corporation)发布的测试标准. SPEC 发布 CPU 2000 是在广泛使用的硬件平台上，提供了对计算密集型性能进行对比的测试标准.

5.1　关键参数测定

空闲内存值. 多数情况下，通过读取客户操作系统中/proc 文件系统的 meminfo 信息即可获得当前空闲内存值. 然而，当物理机内存资源处于紧缺状态时，空闲内存会出现短时间剧烈颠簸现象. 在分配 256 MB 内存的客户操作系统 SLED10(SUSE Linux Enterprise Desktop10)中，运行 mono 标准测试程序，内存访问范围为 40 MB～300 MB. 如图 2(a)所示：采集到的空闲内存值出现多次瞬间剧烈波动. 由于全局调节算法在计算空闲内存价格时需要用到空闲内存值的大小，空闲内存价格异常颠簸最终导致全局调节算法分配给每个客户操作系统的内存值失配.

分析发现，剧烈波动的原因是物理机内存资源紧缺时，客户操作系统通过页框回收算法(Page Frame Reclaiming Algorithm，PFRA)回收内存页面. 因此采用多次采集方法确定客户操作系统空闲内存大小. 设定空闲内存波动发现基准 fmd(5 MB)，k 为收集了空闲内存值次数，$free_{k+1}$ 为当前时刻获取的空闲内存值的大小，$free_k$ 为前一时刻收集到的空闲内存值的大小. 若本次获取的系统空闲内存值满足 $free_{k+1}-\sum_{i=1}^{k} free_i/k>fmd$，则认为该次获取的内存值异常，取前 k 次获取的空闲内存值的平均值作为该时刻系统的空闲内存值，即 $free_{k+1}=\sum_{i=1}^{k} free_i/k$. 根据上述方法实验结果如图 2(b)所示：空闲内存波动降低，为内存动态调整提供了保障.

空闲内存税率. 启动两个客户操作系统，每个系统的最大内存值为 512 MB，分配的当前内存值为



(a) 页框回收算法的负面影响 (b) 多次采集获取平均值

图 2 基于 mono 标准测试程序确定空闲内存示意图

256 MB. 首先在两个客户操作系统中配置了不启用自发调节方式的自发调节服务,在 Domain0 中启用只按照基于空闲内存价格的调节策略进行计算,不作实际调节工作. 在其中一台客户操作系统上运行 mono 标准测试程序,另一台机器上不运行任何程序. 并且设置内存空闲税率 τ 为 0,随时间 τ 值逐渐递增. 实验结果如图 3 所示:当空闲内存税率递增至 0.75 时,调节程序会根据内存使用情况多分配 40% 的内存(mono 访问 100 MB 时,两个客户操作系统中 VM1 内存增加了 40 MB,而 VM2 内存减少了 40 MB). 因此,确定内存空闲税率 $\tau=0.75$.

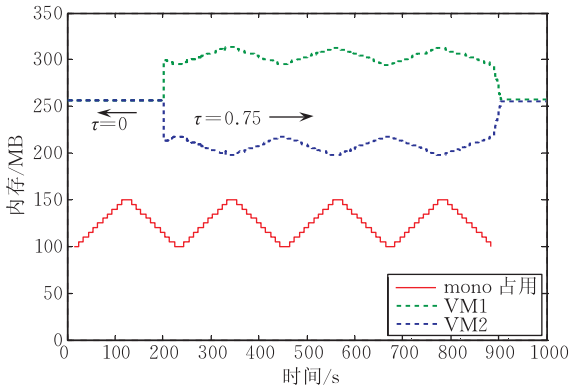


图 3 基于 mono 标准测试程序确定空闲内存税率示意图

交换空间使用率. 交换空间使用率 ε 体现交换空间的利用对内存分配的影响. 启动两个客户操作系统,在客户操作系统上启用自发调节服务,但自发调节服务的自发调节被禁用,只做内存信息收集和交互工作. 而且全局调节也只按照基于空闲内存价格的策略进行计算目标内存值,不执行调节. 在 VM1 中运行申请内存范围为[200,350] MB 的 mono 程序,VM2 上不运行程序. 实验结果如图 4 所示:当交换空间使用率加权比 $\varepsilon=0.1$ 时波动性相对 $\varepsilon=0.04$ 根据交换空间进行调节的范围显著. 在小的时间片内没有很大波动,对调节的影响可以忽略. 因此取交换空间使用率加权比 $\varepsilon=0.1$.

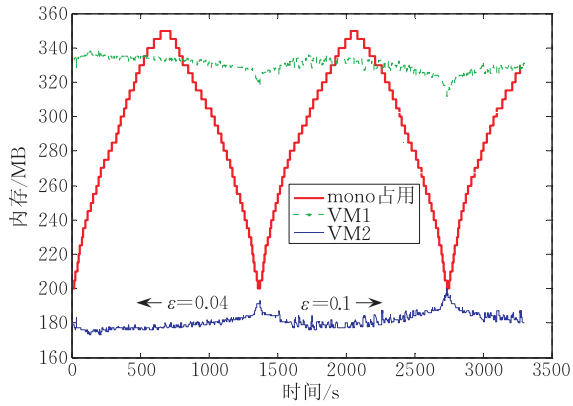
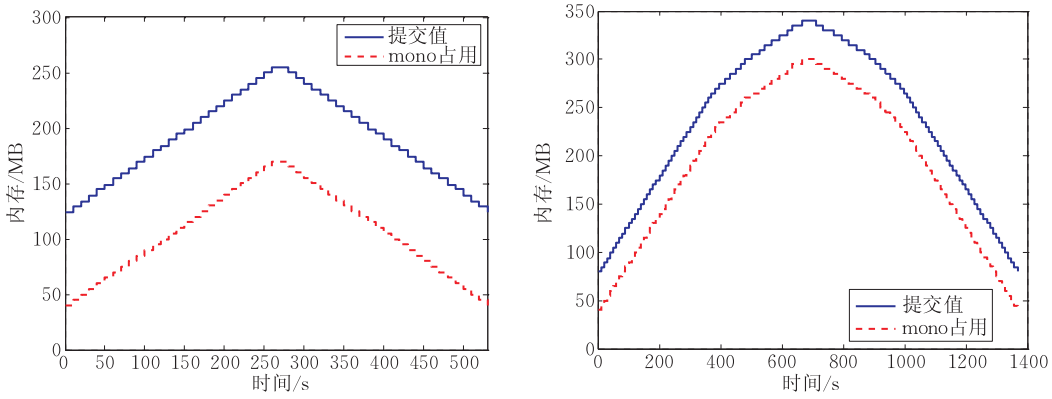


图 4 基于 mono 标准测试程序确定交换空间使用率加权比示意图

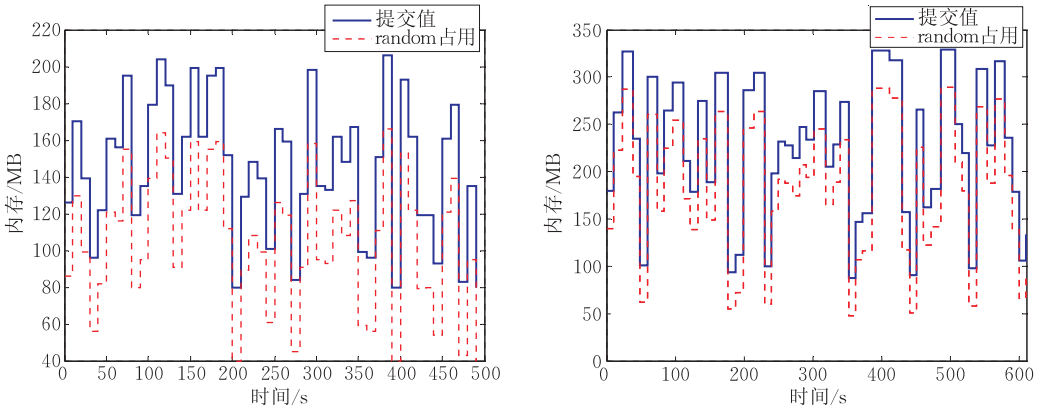
5.2 内存动态平衡算法有效性

自发调节有效性. 当内存资源处于充裕状态时,多虚拟机内存管理系统使用自发调节作为基本的调节策略,也即每个客户操作系统使用提交内存值作为调节的目标内存值. 在客户操作系统分别运行 mono 和 random 两个测试标准,申请内存范围为[40,170] MB 和[40,300] MB. 实验结果如图 5 和图 6 所示:mono 和 random 两个测试标准无论是在有足够物理内存情况下(图 5(a)、图 6(a)),还是使用了交换空间的测试结果(图 5(b)、图 6(b)),均表明系统提交内存能够体现程序占用内存资源的情况,使用提交内存值作为系统的目标内存值以及应用自发调节策略的有效性.



(a) mono (40 MB～170 MB)　　(b) mono (40 MB～300 MB)

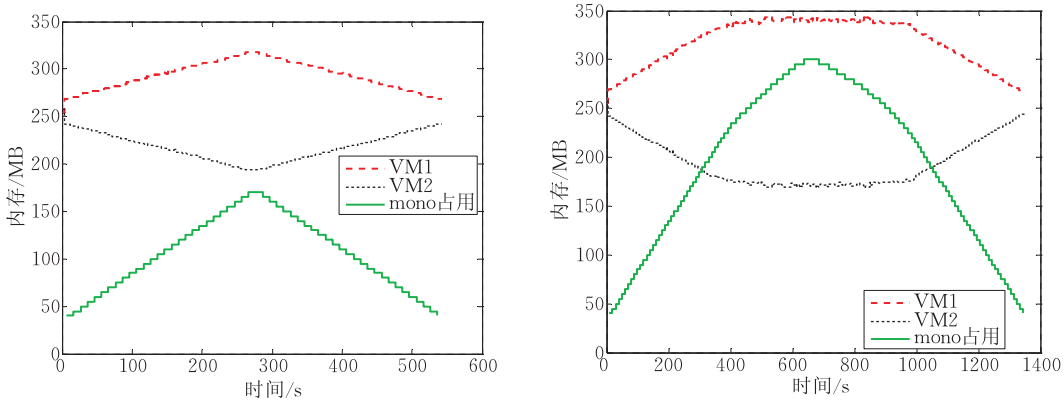
图 5　基于 mono 标准测试程序验证自发调节算法的有效性示意图



(a) random (40 MB～170 MB)　　(b) random (40 MB～300 MB)

图 6　基于 random 标准测试程序验证自发调节算法的有效性示意图

全局调节有效性. 采用 mono 和 random 两个标准测试程序基于空闲内存价格调节方式的有效性. 实验结果如图 7 和图 8 所示：因为每台客户操作系统分配的内存值为 256 MB，因此，mono 和 random 在申请访问内存在[40，170] MB 范围内时没有页面(或者很少)被换入到交换空间. 从图 7(a)和图 8(a)可以很明显地看出，基于空闲内存价格调节策略，两个客户操作系统的目标内存值和运行在客户操作系统上的测试标准能很好地吻合. 另外，因为进行空闲内存值调整的缘故，利用基于空闲内存价格进行内存调节是一个缓慢下降的过程. 图 7(b)和图 8(b)所示的 mono(40 MB～300 MB)和 random(40 MB～300 MB)的实验，因为客户操作系统物理内存只有 256 MB，测试标准访问内存空间必然导致页面换出操作，对于 mono(40 MB～300 MB)当交换空间使用量到一定值时，虽然分配给每个客户操作系统内存



(a) mono (40 MB～170 MB)　　(b) mono (40 MB～300 MB)

图 7　基于 mono 标准测试程序验证全局调节算法的有效性示意图

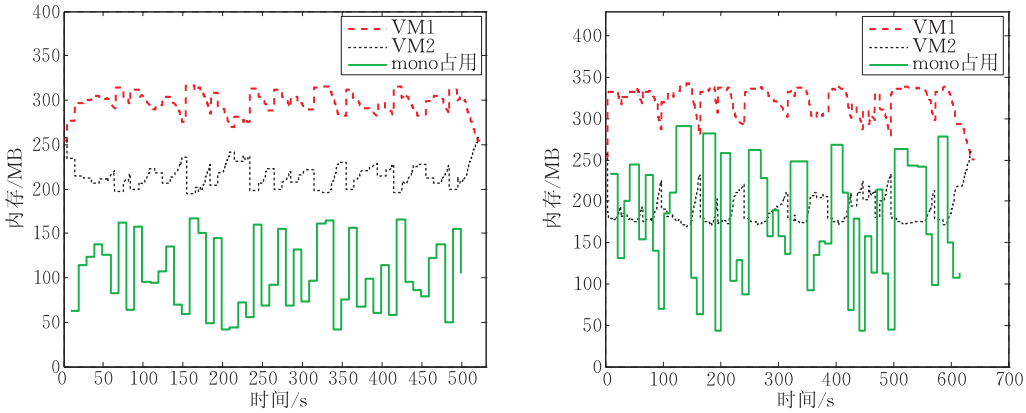


图 8 基于 random 标准测试程序验证全局调节算法的有效性示意图

值趋于稳定,但仍然发现 VM1 的目标内存值在变化. 对于random(40 MB～300 MB)则可以很容易地看出,基于空闲内存价格的方式调节能够按照操作系统上运行的测试程序合理调节客户操作系统的内存.

5.3 全局内存调节算法性能

本节采用不同类型的负载组合着重验证全局内存调节算法的性能.

计算密集型加存储密集型负载. 主要测试基于空闲内存价格的调节策略的性能. 首先利用基本没有内存竞争的程序进行实验. 程序包括 Dacapo 的一套标准和 186.crafty.186.crafty 是一个处理密集型的程序,内存的负载很小. 在 VM1 上 186.crafty 先运行 10 次迭代,然后运行 Dacapo 的标准,同时在 VM2 上先运行 Dacapo,接着运行同样迭代次数的 186.crafty. 图 9 给出了实验调节后程序运行时间和基准时间的对比. 基准时间是程序在启动了自发调节服务(但不执行内存调节)的客户操作系统的运行时间,系统分配的内存始终为 214 MB. 从图中可以看出 eclipse 的运行时间降低最大为 20%,整个 Dacapo的运行时间降低了 13%.

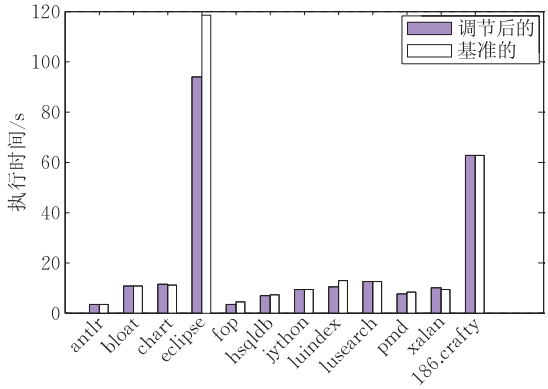


图 9 Dacapo+186.crafty 内存调节前后的性能对比

存储密集型加存储密集型负载. 在内存调节中,最具有挑战的情况就是当出现内存资源竞争时. 在 VM1 上运行 Dacapo,在 VM2 上以逆序的方式运行 Dacapo(表示为 Dacapo′). Xalan 和 eclipse 占用 300 MB 左右的内存,并且 eclipse 占了运行中的大部分时间. 当两个客户操作系统都运行 eclipse 时,出现内存竞争. 图 10 和图 11 分别给出了两个客户操作系统上 Dacapo 中各个程序运行时间和基准时间的对比. 可以看出 eclipse 的运行时间缩短的最大. 和基准运行时间相比,Dacapo 在两个客户操作系统上运行的总时间分别缩短了 36%和 32%.

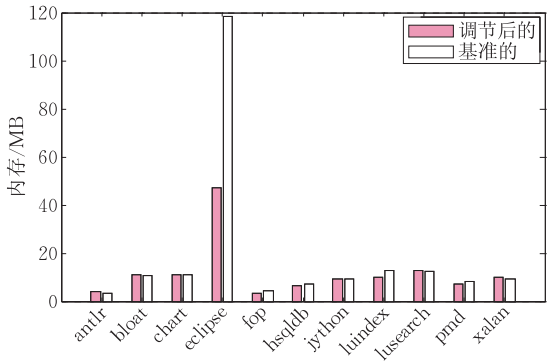


图 10 Dacapo+Dacapo′负载 VM1 中内存调节前后的性能对比

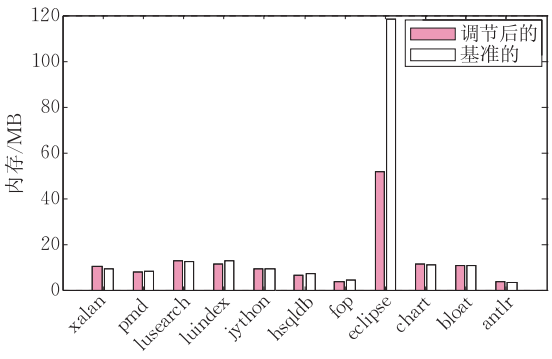


图 11 Dacapo+Dacapo′负载 VM2 中内存调节前后的性能对比

5.4　系统额外开销

额外开销包括存储开销和计算开销. 其中存储开销包括全局调节存储各个客户操作系统相关的信息、存储空闲内存和空闲交换空间的开销. 对于每一个客户操作系统的信息占用约 130 个字节，最多启动 100 个客户操作系统(Xen 3.0 虚拟机监视器至多支持启动 100 个左右的客户操作系统)，存储客户操作系统内存信息需要的内存大小为 12 KB，得出存储空闲内存和空闲交换空间需要的内存大小分别为 50 KB 和 40 KB. 自发调节不做信息存储，内存开销非常小. 相比内存开销，系统 CPU 开销较为明显. Domain0 启用全局调节的 CPU 开销增大了 2% 左右，各个 Domain*U* 在启动了自发调节服务后，CPU 的开销增大了 0.3%.

分别利用 Dacapo 和 SPEC INT 对比观察运行调节程序对客户操作系统性能的影响. 如图 12 和图 13 所示：运行了自发调节客户操作系统对 Dacapo 和 SPEC INT 中的程序均有不同程度的影响，其中对 fop 影响最小为 0.007%，对 eclipse 和 luindex 影响较大，分别为 7% 和 9%. 对 SPEC CPU 2000 中程序的影响为 0～2% 之间.

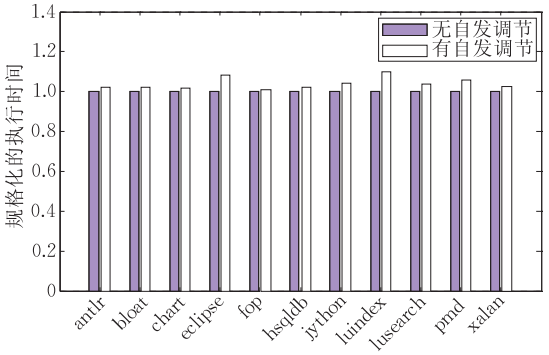


图 12　运行和不运行自发调节情况下 Dacapo 执行时间

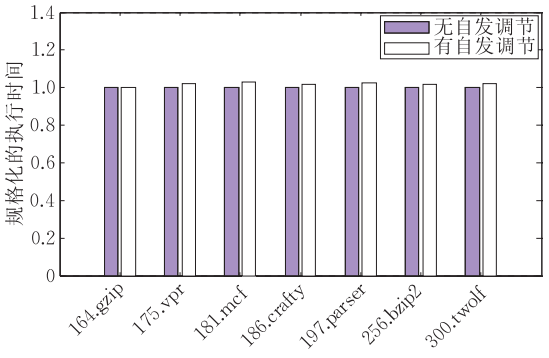


图 13　运行和不运行自发调节情况下 SPEC INT 执行时间

5.5　系统可扩展性

为体现基于空闲内存价格的方法的可扩展性，我们进行多台虚拟机不同类型负载的实验. 启动 3 台虚拟机，第 1 台虚拟机上运行 Dacapo 测试标准集，第 2 台虚拟机上逆序运行 Dacapo 测试标准集，第 3 台虚拟机上运行 SPEC CPU 2000 中的 186.crafty 循环 10 次. 实验结果如图 14 所示：Dacapo 标准集的运行时间分别在前两台虚拟机上分别降低了 17.1% 和 21.8%，186.crafty 运行时间增加了，因为在开始时间，内存资源被更多地分配给了第 1 台和第 2 台虚拟机使用，但从整体上来说系统的性能提高明显.

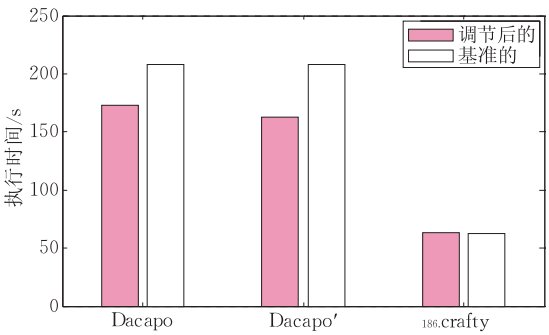


图 14　3 虚拟机混合负载性能对比

如图 15 所示，系统启动 4 个客户操作系统. VM1 上运行 Dacapo，在 VM2 上以逆序的方式运行 Dacapo(表示为 Dacapo′)，在 VM3 上运行 18crafty，在 VM4 上运行 make linux kernel 程序. 这 4 个 VM 的初始内存是 128 M，最大内存值均为 256 M. 与基准时间相比，可以看出在 Dacapo 和 dacapo 上系统性能都提升了 30% 以上. 在 make linux kernel 中提升了 10% 以上，在 186.crafty 测试集中有性能损失不大. 在内存不足的情况下，系统的性能总体提升达到 17%，由此看出在比较复杂的情况下，系统能够显著地提升性能.

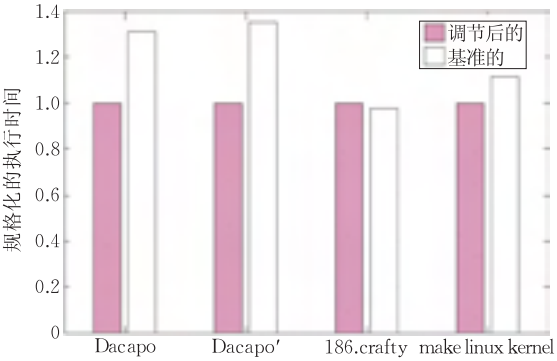


图 15　4 虚拟机混合型负载性能对比

6　结束语

本文提出了自发调节与全局调节协作的多虚拟

机内存管理系统.根据多虚拟机系统中所有虚拟机的内存状态信息,我们定义了内存资源充裕状态和内存资源紧缺状态,依据多虚拟机系统两种不同状态之间的转换执行不同的内存调节策略,提出了自发调节和全局调节之间协作的算法.自发调节是通过操作系统本身的统计信息获取内存使用信息,利用 Xen 提供的气球驱动机制调整分配给客户操作系统的内存资源.提出了基于空闲内存价格的全局调节策略,结合客户操作系统的空闲内存值和空闲交换空闲等信息,确定每个客户操作系统的空闲内存价格,通过平衡价格的方法达到内存资源的平衡.利用多个标准测试程序对基于 Xen 的多虚拟机内存管理系统进行了测试,验证了自发调节与全局调节的有效性.针对计算密集型与存储密集型的混合负载,标准测试程序性能提升达到 13%;针对存储密集型负载,标准测试程序性能提升达到 30%以上.同时,系统平均性能惩罚在 5%以下并具有良好的可扩展性.本文的方法对 IaaS、PaaS 和 SaaS 的服务均可平滑扩展,由于系统本身实现在虚拟机的应用层,针对 PaaS 和 SaaS 进行服务时,仅需掌握应用释放和申请内存情况,即可采用本文的自发调节与全局调节策略进行调节.

致 谢 在此,我们向对本文的工作给予支持和建议的审稿人表示由衷的感谢!

参 考 文 献

[1] Hayes B. Cloud computing. Communications of the ACM, 2008, 51(7): 9-11

[2] Wikipedia. Cloud computing. http://en.wikipedia.org/wiki/Cloud_computing

[3] Goldberg R P. Survey of virtual machine research. IEEE Computer, 1974, 7(6): 34-45

[4] VMware virtualization software for desktops, servers & virtual machines for public and private cloud solutions. http://www.vmware.com

[5] Xen. The powerful opensource industry standard for virtualization. http://www.xen.org

[6] KVM. Kernel based virtual machine. http://www.linux-kvm.org/page/Main_Page

[7] Govil K, Teodosiu D, Huang Y Q et al. Cellular Disco: Resource management using virtual clusters on shared-memory multiprocessors. ACM Transactions on Computer Systems, 2000, 18(3): 229-262

[8] Sugerman J, Venkitachalam G, Lim B H. Virtualizing I/O device on VMware Workstation's hosted virtual machine monitor//Proceedings of the 2001 USENIX Annual Technical Conference. Boston, Massachusetts, USA, 2001: 1-14

[9] Waldspurger C A. Memory resource management in VMware ESX Server//Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI). New York: ACM, 2002: 181-194

[10] Gupta D, Lee S, Vrable M, Savage S et al. Difference engine: Harnessing memory redundancy in virtual machines. Communications of the ACM, 2010, 53(10): 85-93

[11] Barham P, Dragovic B, Fraser K et al. Xen and the art of virtualization//Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York: ACM, 2003: 164-177

[12] Pratt I, Fraser K, Hand S et al. Xen 3.0 and the art of virtualization//Proceedings of the Linux Symposium 2005. Ottawa, Canada, 2005: 65-77

[13] Virtio. http://www.linux-kvm.org/page/Virtio

[14] Schopp J H, Hansen D, Kravetz M et al. Memory hotplug redux//Proceedings of the Ottawa Linux Symposium. Ottawa, Ontario, Canada, 2005: 152-174

[15] Schopp J H, Fraser K, Silbermann M J. Resizing memory with balloons and hotplug//Proceedings of the 2006 Ottawa Linux Symposium. Ottawa, Canada, 2006: 305-311

[16] Wood T, Shenoy P J, Venkataramani A, Yousif M S. Black-box and gray-box strategies for virtual machine migration//Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation. Cambridge, UK, 2007: 229-242

[17] Magenheimer D. Memory overcommit without the commitment//Proceedings of the Xen Submmit 2008. Boston, USA, 2008: 1-3

[18] Zhao W, Wang Z. Dynamic memory balancing for virtual machines//Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Washington, DC, USA, 2009: 37-48

[19] Chen H, Wang X, Wang Z, Zhang B et al. DMM: A dynamic memory mapping model for virtual machines. Science China Information Science, 2010, 53(6): 1097-1108

[20] Li Ya-Qiong, Song Ying, Huang Yong-Bing. A memory global optimization approach in virtualized cloud computing environments. Chinese Journal of Computers, 2011, 34(4): 684-693(in Chinese)
(李亚琼, 宋莹, 黄永兵. 一种面向虚拟化云计算平台的内存优化技术. 计算机学报, 2011, 34(4): 684-693)

[21] Blackburn S M, Garner R, Hoffman C et al. The DaCapo benchmarks: Java benchmarking development and analysis//Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. New York, NY, USA, 2006: 169-190

[22] SPEC CPU2000. http://www.spec.org/cpu2000

ZHANG Wei-Zhe, born in 1976, Ph. D., associate professor. His research interests include network computing, cluster computing, parallel and distributed system.

ZHANG Hong-Li, born in 1973, Ph. D., professor. Her research interests include network and information security, network measurement, network computing.

ZHANG Di, born in 1988, undergraduate. His research interests include network computing, parallel and distributed system.

CHENG Tao, born in 1987, M. S. candidate. His research interests include network computing, parallel and distributed system.

Background

The virtualization technology has provided the important guarantee for the dynamic deployment and security isolation of the infrastructure as a service (IaaS) in the cloud computing. Memory resource optimization is one of the important challenges in the cloud computing environment. The authors propose a memory optimization scheme for multiple virtual machines with the cooperation of spontaneous adjustment and global adjustment algorithms. The experimental results indicate that the frame and the algorithm can sharpen the serviceability well, simultaneously have the high extendibility and the low performance penalty. This work was supported in part by the project of National Natural Science Foundation of China (61173145), National Basic Research Program (973 Program) of China (2011CB302605) and National High Technology Research and Development Program (863 Program) of China (2010AA012504, 2011AA010705).