软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(10):2206−2219 [doi: 10.13328/j.cnki.jos.004681]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

Mbalancer:虚拟机内存资源动态预测与调配*

王志钢[1], 汪小林[1], 靳辛欣[1], 王振林[2], 罗英伟[1]

[1](北京大学 信息科学技术学院,北京 100871)

[2](Department of Computer Science and Technology, Michigan Technological University, Michigan 49931-1295, USA)

通讯作者: 汪小林, E-mail: wxl@pku.edu.cn

摘 要: 在现代数据中心,虚拟化技术在资源管理、服务器整合、提高资源利用率等方面发挥了巨大的作用,已成为云计算架构中关键的抽象层次和重要的支撑性技术.在虚拟化环境中,如果要保证高资源利用率和系统性能,必须有一个高效的内存管理方法,使得虚拟机的物理内存大小能够满足应用程序不断变化的内存需求.因此,如何在单机以及数据中心内进行内存资源的动态调控,就成为了一个关键性问题.实现了一个低开销、高精确度的内存工作集跟踪机制,进而进行相应的本地或者全局的内存调控.采用了多种动态内存调控技术:气球技术能够在单机内有效地为各个虚拟机动态调节内存;远程缓存技术可在物理机之间进行内存调度;虚拟机迁移可将虚拟机负载在多个物理主机间进行均衡.深入分析了以上各种方案的优缺点,并根据内存超载的情况有针对性地设计了相应的调控策略,实验数据表明:所提出的预测式的内存资源管理方法能够对内存资源进行在线监控和动态调配,并有效地提高了数据中心的内存资源利用率,降低了数据中心能耗.

关键词: 虚拟机;数据中心;工作集跟踪;内存管理;性能

中图法分类号: TP316

中文引用格式: 王志钢,汪小林,靳辛欣,王振林,罗英伟.Mbalancer:虚拟机内存资源动态预测与调配.软件学报,2014,25(10):2206−2219. http://www.jos.org.cn/1000-9825/4681.htm

英文引用格式: Wang ZG, Wang XL, Jin XX, Wang ZL, Luo YW. MBalancer: Predictive dynamic memory balancing for virtual machines. Ruan Jian Xue Bao/Journal of Software, 2014,25(10):2206−2219 (in Chinese). http://www.jos.org.cn/1000-9825/4681.htm

MBalancer: Predictive Dynamic Memory Balancing for Virtual Machines

WANG Zhi-Gang[1], WANG Xiao-Lin[1], JIN Xin-Xin[1], WANG Zhen-Lin[2], LUO Ying-Wei[1]

[1](Department of Computer Science and Technology, Peking University, Beijing 100871, China)

[2](Department of Computer Science and Technology, Michigan Technological University, Michigan 49931-1295, USA)

Corresponding author: WANG Xiao-Lin, E-mail: wxl@pku.edu.cn

Abstract: Virtualization technology intends to deliver flexibility, consolidation, and high resource utilization to data centers. High resource utilization as well as high performance promised by virtualization largely depends on effective and efficient physical memory resource management scheme where memory allocation can adjust to dynamic memory demands of applications. This paper presents a predictive memory resource management scheme that combines memory resource monitoring and balancing to improve the resource utilization of a virtualized data center. A design is provided for a new low-overhead working set size tracing mechanism without loss of prediction accuracy. With accurate prediction, the presented scheme further resorts to either local or global memory balancing when the predicted trend of memory demand of a virtual machine exceeds its current allocation. Multiple mechanisms are employed: Ballooning can dynamically adjust memory allocation within a single host, remote cache enables a host to take the idle memory of another host as its network cache, and virtual machine migration moves virtual machines across multiple physical servers. The strength and weakness of each

* 基金项目: 国家自然科学基金(61232008, 61170055, 61272158, 61328201); 国家高技术研究发展计划(863)(2012AA010905); 高等学校博士学科点专项科研基金(20110001110101); 深圳市生物、互联网、新能源、新材料产业发展专项资金(JC201104210107A)

收稿时间: 2014-02-28; 修改时间: 2014-07-07; 定稿时间: 2014-07-31

mechanism and design selection policy for memory balancing according to memory pressure are also discussed. Experimental results show that the global memory balancing achieves a significant center-wide speedup and energy conservation.

Key words: virtual machine; data center; working set size tracing; memory management; performance

随着软硬件技术的高速发展,高性能、低开销、易维护的计算促进了现代数据中心的快速发展.如何在保证程序性能的前提下,对数据中心分布式的计算资源高效地加以使用,成为了一个富有挑战性的问题.由于虚拟化技术在服务器整合、硬件资源分配、节能等方面具有灵活、易操作的特点,近年来在数据中心以及云计算架构中得到了越来越广泛的应用.

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

本文首先提出了虚拟数据中心的动态内存调控机制:(1) 低开销的内存工作集在线预测机制;(2) 合理、高效的多机内存动态调控机制.并在预测和调控机制上,进一步提出了在数据中心内进行内存调控的策略.

本文第 1 节介绍相关工作背景,并进一步阐释本文与相关工作的差别与意义.第 2 节介绍虚拟机内存工作集的在线预测.第 3 节介绍单机和多机环境下的内存资源动态调配.第 4 节是重要实验的数据分析.第 5 节是对全文工作进行总结和对进一步工作的展望.

1 研究基础和相关工作

应用程序的活动内存工作集(working set size,简称 WSS)指的是在最近一段时间窗口内被访问的内存页面的集合,它是应用程序需求内存的量度.一般来说,当应用程序的工作集大小超过了系统的可用物理内存时,内存管理器就会给磁盘换出一部分页面,程序的性能因此也受到影响.内存工作集的准确预测,能够指导我们给系统分配合适的内存大小,从而达到最优的性能.

目前,已有很多基于内存工作集的研究工作.VMware ESX 采用的方法是采样,在单位间隔内监控一组随机的内存页面,一段时间后统计出这组页面的利用率作为整个物理内存的利用率.但是程序性能和分配内存大小并不是呈线性关系的,所以该方法不能预测当把这些不活动的页面回收后,对程序性能会产生多大的影响.Geiger[4]通过监控磁盘 I/O 来推断当前内存压力并计算所超载的内存大小,但是这只能预测当内存超载时的应用程序的工作集大小,无法对回收多余的内存做出决策.

气球技术[1]是虚拟机系统中所特有的虚拟存储技术,可以从其他虚拟机窃取一些未使用的机器内存页面,给急需内存的虚拟机使用,进而实现对内存的动态调整.为了实现内存的窃取,VMM 需要在 GuestOS 的内核中安装一个用于窃取内存的模块,称作 Balloon Driver.本文的前期工作之一[5]利用了气球技术来动态均衡运行在单个物理主机上的各个虚拟机之间的物理内存,它实现了一个基于 VMM 的内存预测器,先预测出 VM 的 WSS,然后重新通过气球驱动的接口为各个 VM 重新分配可用内存.MEB 的不足之处在于,它只是一个本地的调控机制.一旦所有虚拟机的内存需求超过物理主机上的可用物理内存,VM 只能将页面换出来缓解内存压力.

为了使物理主机能够利用其他主机上的内存资源,Comer 等人[6]提出了“远程内存”模型,使在不同的物理机之间进行内存调度成为可能.网络传输速度远远高于磁盘 I/O 的读写速度,因而远程缓存可以显著降低磁盘

I/O 所带来的性能下降.同时,虚拟机迁移[7]可以将一台虚拟机从一台物理主机迁移到另外一台物理主机上去,也可以实现物理机之间的负载均衡.

Overdriver[8]采用了远程缓存和虚拟机迁移两种技术进行虚拟化环境下的多机内存调控,根据负载时间的长短选择调控方法,汲取了以上两种方式的优点.但是 Overdriver 存在着两个明显的不足:(1) 每台物理主机上预留出一部分内存空间作为远程缓存或迁移使用,而未考虑对本机的 GuestOS 所产生的影响,造成了内存的浪费;(2) VM 的内存分配仍然是传统的静态分配方法,不能在本地主机内部进行有效的调节.

2 动态内存预测(dynamic memory prediction,简称 DMP)

建立应用程序的物理内存大小和其性能之间的数学模型对于内存管理是十分重要的,而 WSS 是程序访存行为的直接体现,是度量程序内存需求的重要指标.MEB 通过用 VMM 截获虚拟机对物理页面的访问请求,建立了内存访问直方图(least recently used,简称 LRU),可直观地观察到程序的局部性特征[9]以及失效率.

2.1 基于LRU的低开销DMP

在 x86 机器上,页表项(page table entry,简称 PTE)的第 2 位指定了访问权限为用户态或系统态[10],我们把指定页面的属性变为系统态,用户程序读写页面的所有请求都会产生次要页面失效陷入到 VMM 中,从而实现了应用程序访存的捕捉.我们使用页面保护\陷入机制来捕捉虚拟机上应用程序的访问内存行为,并以此构造内存访问失效率曲线(miss ratio curve,简称 MRC).但是这种方法带来的系统开销很大,因此我们使用了多种优化措施,在保证跟踪 WSS 精确度的前提下大大降低了系统开销.之后,采用线性拟合的方法滤去噪声,得到 WSS 的预测值.

一般常用的组织地址标签的数据结构是双向链表,我们可以将基于 LRU 的 WSS 的总代价表示为 $O(I\times(H+U))$,其中,I 是页面陷入的次数,H 是处理页面失效的时间,U 是更新 LRU 直方图的时间.为了减小系统开销,采用 3 种方法进行优化.这 3 种方法分别是:(1) 动态热页集;(2) 基于 AVL 树的 LRU 链表;(3) 间断式 LRU 监控.其中,方法(1)和方法(3)的目的是减小 I,方法(2)是为了减小 U.图 1 显示了优化后的 WSS 跟踪系统.

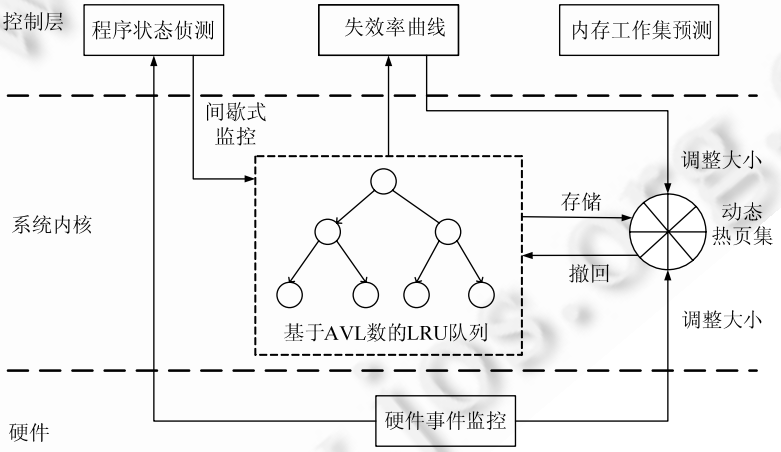


Fig.1 Optimized WSS tracing system

图 1 优化后的 WSS 跟踪系统

2.1.1 动态热页集(dynamic hot set,简称 DHS)

WSS 跟踪的系统开销和陷入次数是成正比的,如果让每一个页面访问都陷入 VMM 中去,那么代价无疑是相当昂贵的.为此,我们采取了“热页集”的概念,即,将所有物理页面划分为两个集合:热页集 H 和冷页集 C.H 用来记录最近经常被访问的页面,H 之外其余的页面都是冷的.虚拟机可以正常访问热页,只有对冷页面的访问会

陷入到 VMM.这样,热页集越大,可能产生的陷入次数就越少.

初始状态时,我们将所有页面都放在 C 中:当 GuestOS 新建一个页表时,将所有一级页表的 PTE 的访问权限设置为系统态,接下来,任何一个对冷页面的访问都会陷入 VMM 中去;VMM 的页面失效处理程序能够从上下文获知陷入的虚拟地址和其所属的页表地址,然后,只需要 $O(1)$的时间定位到对应的 PTE 和机器地址;接着,将 PTE 的用户访问权限恢复,这个页面就划归到 H 中.以后,对这个热页的所有访问请求都不会再陷入 VMM 中,因此避免了产生多余的开销.

对于局部性良好的程序,固定的热页集能够达到有效控制系统开销的目的.而对于局部性差的程序,仍然会有相当大的一部分页面访问落到冷页集中.为了改善这种情况,我们引入了动态热页集机制,从 MRC 的形态中得到程序局部性好坏的信息,并进而在程序局部性差时扩大热页集容量.图 2 显示了 SPEC CPU2006 的一组程序 sjeng,omnetpp,astar 和 xalanbcmk 在使用不同热页集大小时 WSS 跟踪的性能.

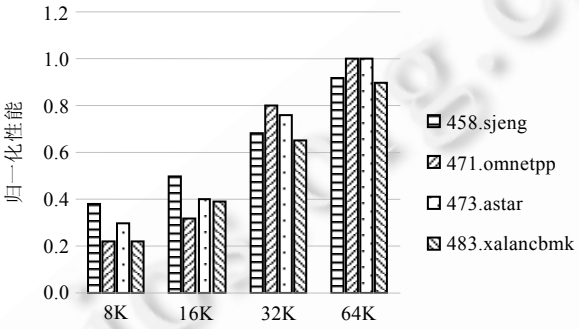


Fig.2 System overhead under different hot set size

图 2 不同热页集大小产生的系统开销

就精确度而言,对于大多数程序下热页集的大小远远小于 WSS,可以保证 VMM 能够捕捉到足够数量的页面陷入来构造 LRU 直方图.但是如果热页集设置得太大,则会导致太多的页面访问都落到热页集中,VMM 无法获得足够的虚拟机访问页面信息,构造出的 MRC 的形态可能会非常平缓,如果按照事先设定好的失效率阈值估计出内存需求,则结果会远远超过实际值.因此,需要根据程序运行的状态动态设置热页集的大小.

为了得到热页集大小对跟踪误差的影响,我们使用二进制代码分析工具 PIN[11]跟踪了无热页集时的访存地址,并模拟了设定热页集之后的 LRU 跟踪.为了避免热页集太大所造成的极端情况,需要知道落入热页集的内存访问的比例.如果大部分的内存访问都是针对热页的,则可以认为 WSS 的大小约等于热页集的大小.结合 DTLB Miss 的采样数据,可以用来推断对热页的访问和实际发生的总访存之间的差距.但是,由于 DTLB 的容量非常小,我们可以用 DTLB Miss 的个数作为总的访存次数的近似值.设 DTLB Miss 的数量为 T_m,陷入 VMM 的访存次数为 I,观察到,一般情况下,$T_m \leqslant I \times 10^2$.当 $T_m >> I$ 时,表明有相当大的一部分访存行为未被 VMM 捕捉到.这时,我们就会减少热页集大小,或者只简单地把当前的热页集作为 WSS.

2.1.2 基于 AVL 树的 LRU 直方图(AVL-based LRU,简称 ABL)

一般常用的组织地址标签的数据结构是双向链表,其维护 LRU 直方图的时间代价为 $O(N)$.若程序局部性良好,被命中的页面的标签往往离链表头很近;但是若程序内存需求很大,同时局部性又很差,则双向链表的搜索代价就非常大.如对于 milk 程序,搜索本身所带来的系统开销竟然达到了 238%.为了解决这个问题,我们将传统的双向链表数据结构改变为 AVL 树,把时间代价降低到 $O(\mathrm{Log}N)$,从而有效降低了构造 MRC 过程中的存储代价.

2.1.3 间断式 LRU 监控(intermittent LRU monitoring,简称 ILM)

程序的访存行为呈现出周期性的特征,我们发现:内存需求的变化与一些与访存相关的硬件事件(如 DLTB Miss,L2 Miss 等)是成正比的,而这些硬件事件的数据都能够通过特殊寄存器直接读取到,监控的代价很小.这个

特性启发我们:可以在程序处于一个稳定的周期时暂时关闭 LRU 监控,而在它进入到一个新的周期时通过特殊的硬件事件再次开启 LRU 监控.

在相关的 CPU 硬件事件中,DTLB Miss 和访存操作是最直接相关的.除此之外,L1 Access,L2 Miss 也是能直接影响访存行为的事件.图 3 给出 SPEC CPU2006 的一组程序的 WSS 与 DTLB Miss,L1 Access,L2 Miss 的相关性.由于一些处理器是支持同时对多个 CPU 事件进行监控的,可以设计不同的策略来选择检查周期变化的事件.激进的策略将尽可能地减小 LRU 监控时间,当所有的监控事件都发生了大幅改变时,才认为发生了周期交替然后唤醒 LRU 监控;相反地,保守的策略将尽可能地增长 LRU 监控时间,因此它对于事件的变化更为敏感,只要我们观察到其中任何一个被监控的事件有大幅度波动,就把它当作一个新的周期.另外,还可以采用相对温和的选举策略,以多数 CPU 事件的变化来决定周期变化.

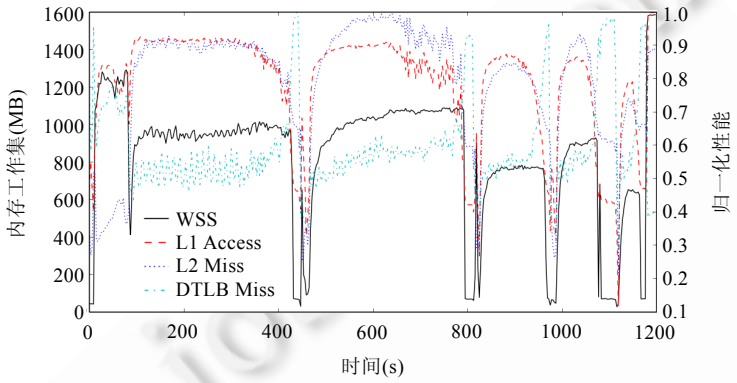


Fig.3 Consistency between WSS and hardware events

图 3 WSS 和硬件事件的一致性

2.1.4 周期检测

之前关于程序周期的研究都依赖于复杂的信号处理技术,如傅里叶变换、小波分析等[9].尽管这些技术能够在离线状态有效地滤除噪音,识别出周期变换,但分析的代价太大,因此不适合在程序动态执行中使用.本文采用了一种简单但有效的算法来检测内存需求的周期性变化.由于硬件计数器采样的原始数据有随机噪声,我们首先采用滑动平均滤波(moving avarge filter)对原始数据进行去噪处理.

设第 i 个采样间隔的滤波函数为 $f(i)=(v_i+v_{i-1}+\ldots+v_{i-k+1})/k$,其中,$v_i$ 是第 i 个采样间隔内得到的监控数据,k 是滑动窗口宽度,有如下公式:

$$\begin{cases} f_{mean}=mean(\{f(x)\mid x\in(j-k,j)\}) \\ err_r=f(j)/f_{mean} \\ err_a=\mid f(j)-f_{mean}\mid \end{cases}.$$

如果滤波器还没有填满采样窗口,说明还没有足够多的信息来判断新的周期,LRU 监控在此之前必须启动.当采样间隔足够多后,设 v_j 为当前间隔的采样值,err_r,err_a 分别是当前的滤波值与滑动窗口的平均滤波值之间的相对误差和绝对误差.对于一个给定的阈值 T,如果 err_r=[1−T,1+T],我们认为输入信号处于平稳的状态;否则,认为发现了一个新的周期,并将当前窗口内所有的数据清空,并重新对新周期进行采样分析.下面来讨论如何设定合适的阈值 T.

- 固定阈值的周期识别

在监测 WSS 是否稳定时,如果要保持较高的精度,可以设置一个较紧的阈值,如 0.05.另外,err_a 也可以用作指导 WSS 跟踪.例如,如果 WSS 跟踪是为了在 MB 的级别上进行内存适配,那么只要 err_a<1MB,就仍然可以认为 WSS 是稳定的,无论是否 err_r>T.对于过滤硬件计数器的噪声,如果设置的阈值过小,就有可能将数据的随机变化当成周期变化,从而不必要地开启 LRU 监控;相反地,若阈值设置得过大,滤波器就会忽略 WSS 变化,导致

LRU跟踪结果不准确.实验数据显示:对于一个特定的硬件事件,不同程序和不同的周期T值的变化都是很大的.对于固定阈值法,我们只能通过实验找到一个经验值,既能降低平均开销,又能保持精度.图4显示了两个周期检测器是如何协同工作的:一个是基于WSS跟踪的检测器,当检测到周期稳定时关掉LRU监控;另一个是基于硬件计数的检测器,当LRU停止监控时,用来识别新的周期并唤醒WSS跟踪.

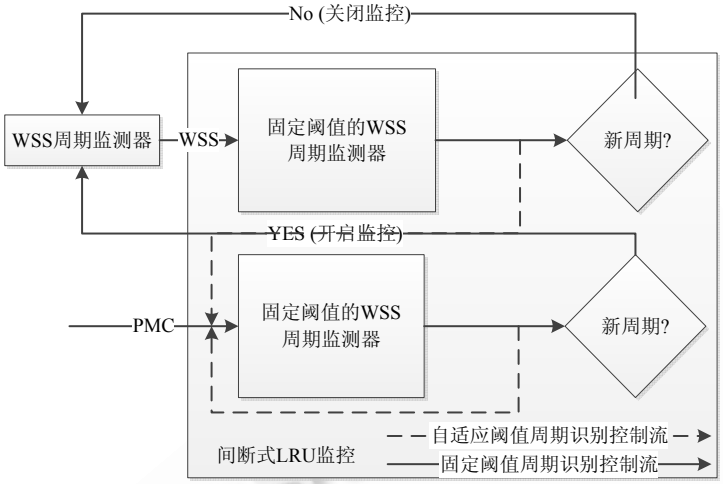


Fig.4 Phase detecting control flow

图4 周期识别控制流图

- 自适应阈值的周期识别

虽然固定的阈值能够减少平均开销,但是由于不同的程序访存特性差别很大,它不能使特定程序性能达到最优.为此,我们进一步提出了自适应阈值的周期识别方法,在程序动态执行的过程中,通过调节T的大小来改善性能.这种方法的思想是:将WSS跟踪的状态反馈到基于硬件计数器的周期检测器,形成一个闭环控制系统,如图4所示.开始时,两个周期检测器同时工作且阈值设置相同,如果两个检测器的结果一致,则T无需作任何改变.如果WSS检测器得到的结果是稳定的,但PMC检测器做出了不一致的判断($err_r>T$),说明当前设置的T值太紧,我们将T放松到err_r,那么T增大后,下次PMC检测器可能会发现系统进入了稳定的状态,从而关掉WSS跟踪;相反地,如果当前内存需求不稳定,但是PMC检测器的结果是稳定的($err_r<T$),说明T设置得过宽,我们就将它缩小到err_r.

只要WSS跟踪处于开启状态,T的调整取决于是否与WSS跟踪的周期状态一致.但是,当内存跟踪被关掉时,就无法获知WSS的信息了.为了解决这个问题,我们使用“检查点”的方法:每过k个采样周期后,唤醒WSS跟踪器,让它工作一段时间以决定是否需要调整T,如果不需要调整,则继续关闭WSS跟踪,直到下一个检查点或检测到一个新的周期.在理想情况下,不到检查点的WSS跟踪就应该处于停止工作状态,即,开启时间占总时间的比例为1/k.我们可以按照这个理想状态的期望动态地调整k的值,以减少开销.如果在上一个检查点没有作任何调整,则可以适当增加k的值;一旦在某个检查点作了调整,就立刻将k恢复到原始的默认值.对于一个周期比较长的程序,这种调节方法能够进一步缩短LRU开启的时间.在具体实现中,k设置为10个采样周期,增加的步长是5.

2.2 在线DMP

基于LRU的跟踪法估算出的WSS不会超过当前虚拟机所有的可用内存.除此之外,我们还需要知道:当物理内存不够时虚拟机所需要的额外内存的大小.我们知道:当虚拟机内存超载时,它就会通过页面交换利用swap分区读写数据.大多数操作系统中都能直接将swap分区的读写次数输出给用户.因此,我们在每个虚拟机上运

行一个后台程序用来定期收集虚拟机 swapin 的值并传递给内存预测器.如果没有发生 swapin,那么预测器只需要根据 MRC 得到 WSS 的大小即可;如果 swapin 的值超过了限度,预测器会根据当前物理内存的实际大小和 swapin 的值得出一个估计值,作为期望的 WSS 大小.

由于当内存超载时内存预测依赖的输入是当前分配内存和 swap 的使用数据,与 LRU 跟踪的输出没有直接关系,这一点启示我们:可以在检测内存超载的时候关闭 LRU 监控,从而进一步降低跟踪开销.

在 SPEC CPU2006 的测试结果表明:传统的 WSS 跟踪法所引起的平均系统开销高达 187%;而经过我们一系列的优化后,平均系统开销只有 4%.内存预测工作为后续的动态内存调配工作奠定了稳定的基础:它不仅能探测出各个虚拟机真实的内存需求,而且如此低的系统开销不会影响到内存调配期望所带来的性能提升.

3 内存动态调配

根据内存工作集在线预测的结果,我们可以在单个物理主机内部实时地调整每台虚拟机的物理内存分配,然后进一步采用远程缓存机制和虚拟机在线迁移机制将本地内存调配扩展到数据中心,以提高整体系统的内存利用率.

3.1 全局内存调配

我们设计了一个全局内存调配器(global balancer,简称 GB)来协调数据中心的各个服务器的内存需求以及供给,每台服务器上的内存预测器能够即时感知本地的内存需求.全局调配器会定期轮询每台服务器,当它发现某台主机出现内存超载的情况时,就会找到另外一台有足够空闲内存的主机作为迁移目的机器并引发迁移,从而使数据中心的内存资源得到最大化的使用.

在内存预测模型的基础上,我们可以采用多种策略来进行内存调控,图 5 显示了全局内存动态调控的原理.在实现调控策略时,我们采取的原则是“先本地,后远程”,即,尽可能地利用本地物理主机的内存和气球技术满足虚拟机的内存需求,当本地物理内存超载时,再借助于数据中心内其他物理主机来缓解内存压力.在多机环境中,远程内存交换和迁移是调控内存分配、均衡负载的常用技术,二者各有千秋.远程内存交换的代价较小,而且启动时间短,因而适用于短期的内存超载;相反地,虚拟机迁移的代价较大,因此适用于持续的内存超载.

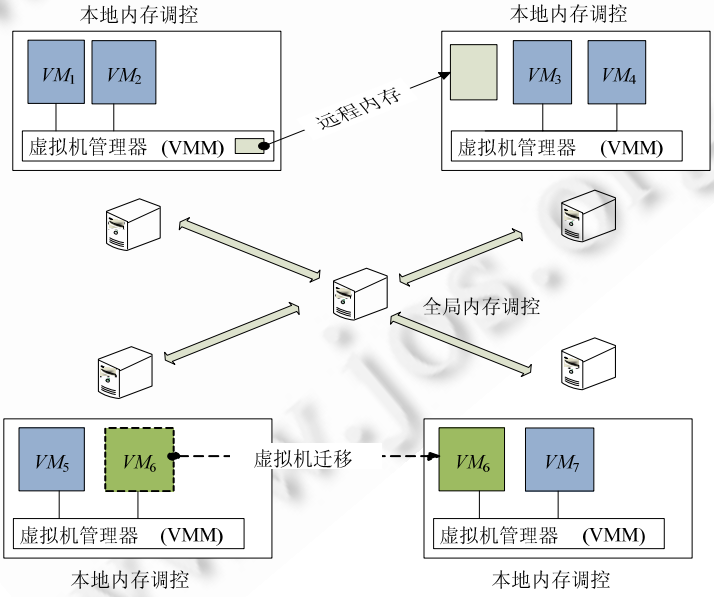


Fig.5 Gloabal memory balancing framework

图 5 全局内存调控框架图

3.2 单机内存资源动态调配

当多台虚拟机竞争内存资源时,我们可以使用内存跟踪器计算出每个虚拟机的内存需求.内存管理器将尝试找到一种调配策略,使得虚拟机整体性能达到最优,虚拟机内存的重分配是通过 ballooning 机制来完成的.内存调配器的关键是调配算法的设计,本文采用了类似文献[5]中的调配算法,在考虑算法设计时遵循了以下原则:

- 性能:调整后的内存分配使得虚拟机产生的总的页面交换次数最少;
- 公平性:内存调配过程中不应有虚拟机被“饿死”;
- QoS:保证虚拟机能提供最低标准的服务;
- 可扩展性:当物理主机上运行的虚拟机个数增加时算法依然具有普适性.

假设 P 是物理主机上所有可用内存,V 是所有运行的虚拟机的集合,调配后,虚拟机 VM_i 的内存为 T_i.预测器根据虚拟机 VM_i 的 MRC 和 swap 利用率得出的内存需求为 WSS_i.为了保证 QoS,分配给 $VM_i \in V$ 的最小内存为 L_i.设 $E_i=\max(L_i,WSS_i)$为 VM_i 的内存期望值.当 $P \geqslant \sum E_i$ 时,可用物理内存能够满足所有的 VM 的内存需求.剩余的 $(P-\sum E_i)$ 可以作为 bonus 灵活地分配给各个 VM.在我们的设计中,是把 bonus 按照 E_i 的比例分配给各个 VM,即,$T_i = bonus \times \left(E_i / \sum E_i\right)$. 而当 $P < \sum E_i$ 时,说明至少有一台虚拟机的内存需求没有得到满足.假设所有的虚拟机具有相同的优先级,我们的目标是要减少系统总的页面失效次数.设虚拟机 VM_i 当前的失效率曲线函数为 $MRC_i(x)$,访问内存次数为 NR_i.那么,当它的内存分配值为 m 时,会产生 $Miss_i(m)=MRC_i(m)\times NR_i$ 个页面失效.因此,我们的目的是为了找到一组$\{T_i\}$,使得 $\sum_{i \in V} Miss_i(T_i)$ 的值最小.

我们用近似贪心的算法来求最优值.虽然 ballooning 可以在页面的粒度上调整内存大小,但是这样的话找到最优值的时间复杂度就是 $O(M^{|V|})$,其中,M 是属于 VM 的最大物理页面个数.显然,这对于在线算法来说是不现实的.因此,我们采用一种较为简单的近似算法,将搜索的步长从一个页面扩大到 S.下面以调整两台虚拟机 VM_1,VM_2 的内存为例,来说明它是如何工作的:

1. 将 VM_1 的内存减少 S,将 VM_2 的内存增大 S;
2. 计算调整后的页面失效数 C,重复步骤 1,直至求出 C 的最小值 C_1;
3. 再将 VM_1 的内存增大 S,将 VM_2 的内存减小 S;
4. 计算调整后的页面失效数 C,重复步骤 3,直至求出 C 的最小值 C_2;
5. 选出 $\min(C_1,C_2)$,并将产生最小 M 的虚拟机各自的分配值作为调整后的最终结果.

当有超过两台虚拟机工作时,可递归地调用如上算法.

我们在 Xen 上实现了如图 6 所示的单机内存调配器:使用 WSS 跟踪器对每个 VM 进行跟踪,实时获取 WSS;运行在 Domain0 上的调控器能监控到各个 VM 的内存需求,并通过气球驱动对各个 VM 的内存按需进行分配.

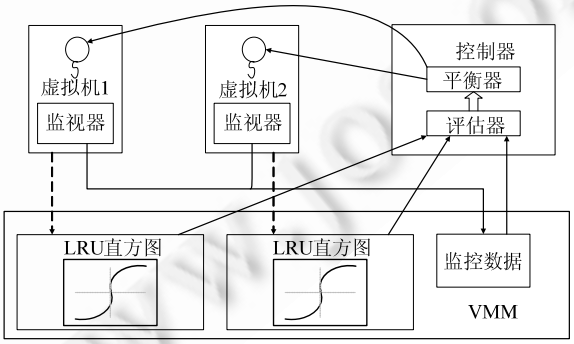


Fig.6 Local memory balacing

图 6 单机内存资源调配

3.3 基于远程缓存的多机内存资源调配

现代数据中心的服务器都是千兆以太网甚至更快的网络介质互联,而网络的传输延迟要比磁盘访问延迟低好几个数量级,这就为我们提供了一种有效缓解虚拟机页面抖动的途径:远程磁盘缓存,其工作原理如图 7 所示.我们可以在本地内存不足时,通过网络来使用 cache 服务器上的远程内存.每个物理节点上都嵌入了远程内存驱动,一旦系统要发生 swap,远程内存驱动就把要发生 swap 的页面通过网络存到远程 cache 服务器内远程缓存中;而一旦系统需要被缓存过的页面,则先去远程缓存中查询,如果命中,则从远程取回相应的页面.

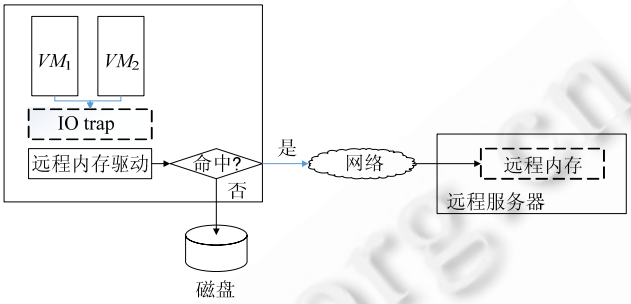


Fig.7 Principle of remote memory

图 7 远程内存工作原理

在远程内存交换模型中,远程内存缓冲区相当于磁盘缓冲和磁盘驱动层之间的二级缓冲.如何协调本地磁盘缓冲与远程缓冲区的操作以达到最佳的使用效率,是我们在设计中需要考虑的又一个重要问题.我们采取“排除型缓存”:低层的缓存只存储被高层缓存淘汰的数据.这种策略可使额外的数据拷贝开销和无效的内存访问开销尽可能地得到减少.当第 1 次从磁盘读出新的磁盘块时,VMM 只会把磁盘块的内容写入磁盘缓冲区,而当这块数据要被 GuestOS 淘汰时才将它写入远程缓存.文献[12−14]都表明,排除型缓存具有更高的利用率.

在排除型的缓存布局中,只有当 GuestOS 回收页面时才会分配新的缓存块.根据局部性原则,刚刚被 GuestOS 换出的页面极有可能在最近的将来又被访问.因此,我们在 VMM 中维护一个 LRU 链表,用来暂存由 GuestOS 淘汰下来的页面,每从 GuestOS 接收到一个新的被淘汰的页面,就把它插入 LRU 链表的头部.这样,VMM 的 LRU 链表就和 GuestOS 自己管理的 LRU 链表形成一个统一的 LRU 结构,活动频繁的页面总是处于链表的头部.

当本地内存调配器发现内存超载时,VMM 会首先将 LRU 链表头部的页面传送到远程.同时,内存服务器也按照 LRU 的方式组织缓存块.当新分配了一个缓存块时,服务器将它插入 LRU 链表头部;当缓存容量不够时,服务器首先选择淘汰 LRU 链表尾部的页面.这种 LRU 的方式能够提高缓存的命中率和利用率.由于一台服务器上的物理内存是固定的,因而远程缓存的容量和内存服务器本身的内存需求是呈互补关系的.随着虚拟机内存需求的变化,内存服务器也需要调整缓存的容量.如果需要扩大缓存,直接将空闲页面添加到缓存池中即可;如果需要缩小缓存,则从 LRU 链表的尾部删除掉多余的页面并释放.

远程内存是内存动态平衡的关键性技术,其提供了一种轻量级的机器间内存共享机制.这是因为,在程序的运行周期内,其内存需求是动态变化的.如果一个物理节点内多个虚拟机的内存需求可以互补,则可以通过本地内存调控在虚拟机之间进行内存调控.但是多个虚拟机同时发生内存不足的应用场景是有可能发生的,这样,本地内存调控已经不能满足所有虚拟机的内存需求,从而导致部分虚拟机发生 swap,引起整体的性能下降.

传统的做法是在合适的时候使用虚拟机迁移进行负载均衡,但其不足之处是显而易见的:一是虚拟机迁移需要迁移一个虚拟机的运行状态和全部内存数据,迁移的过程中不仅不能缓解内存紧张的状况,而且会进一步占用系统资源;二是在不确定内存超载时间时就启动虚拟机迁移,很可能导致虚拟机的来回迁移,额外占用了大量系统资源,导致全局性能严重下降.

远程内存则可以有效地解决这个问题.一旦某物理节点发生内存不足现象,它可以立刻向有内存空闲的机

器申请远程内存空间,并把即将发生 swap 的页面通过网络发送到远程缓存池;在需要该页面的时候再通过网络取回,从而避免了 swap 的发生.其优点是启动快、代价小,在基本不增加系统开销的前提下,填补了本地内存调配和虚拟机迁移之间的调控机制的空缺.

4 性能测试

为了测试本文提出的内存预测系统在虚拟化环境下的性能,我们使用开源的虚拟机监控器 Xen[15]实现了整个内存调度系统.Xen 支持两种虚拟化方式:全虚拟化和半虚拟化.我们的实现基于的是后者.测试环境见表 1.

Table 1 Experiment settings

表 1 实验环境配置

项目	配置
处理器	2.8GHz Intel Core I5 9300
内存	12G 1066MHz DDR3
网卡	1000M bps
VMM	Xen4.2.1
Domain0	Linux 3.10.12 x86_64
DomainU	Linux 3.10.12 x86_64 (PVM)

4.1 WSS性能测试

为了测试 3 种优化方法对 WSS 跟踪的系统开销的影响,我们首先测试了 SPEC CPU 2006 和 Dacapo[16]程序组在没有任何 WSS 跟踪情况下的程序性能作为基准性能;然后,比较了使用不同方法的 WSS 跟踪的性能:基于链表的 LRU 直方图(L)、DHS(D)、ABL(A)以及固定和非固定阈值的间歇式监控($I(f)$和$I(a)$).在本实验中,为了保证所有子程序的性能,我们为每个虚拟机都分配了 4G 内存.

从表 2 可以看出:对于整个 SPEC 2006 程序组,使用双向链表所导致的平均系统开销有 173%;而单独使用 DHS 或 ABL 时,系统开销减少到 39%和 43%;同时使用 DHS 和 ABL 能够进一步将系统开销减小到 16%;当 WSS 较小或程序局部性非常好时,ABL 和 DHS 的优势并不突出;但是对 WSS 很大或局部性很差的程序,优化所带来的提高就显而易见了.

Table 2 Performance of WSS tracing system

表 2 WSS 跟踪的性能优化

程序	程序执行时间						LRU 开启率	
	L	D	A	D+A	D+A+$I(f)$	D+A+$I(a)$	$I(f)$	$I(a)$
400.perlbench	1.54	1.28	1.07	1.05	1.00	1.02	0.22	0.11
401.bzip2	1.03	1.04	1.01	1.02	1.01	1.01	0.76	0.14
403.gcc	1.96	1.17	1.37	1.08	1.02	1.03	0.87	0.11
410.bwaves	3.30	2.81	1.30	1.33	1.19	1.02	0.56	0.15
416.gamess	1.01	1.01	1.01	1.01	1.00	1.00	0.18	0.09
429.mcf	59.16	9.07	3.21	1.75	1.41	1.04	0.72	0.37
433.milc	13.08	8.45	3.35	2.74	2.46	1.05	0.52	0.11
434.zeusmp	2.49	1.33	1.22	1.14	1.06	1.06	0.13	0.21
435.gromacs	1.01	1.01	1.00	1.00	1.00	0.99	0.13	0.10
436.cactusADM	1.20	1.12	1.08	1.09	1.00	1.02	0.14	0.15
437.leslie3d	2.68	1.01	1.37	1.00	1.00	1.00	0.13	0.10
444.namd	1.02	1.01	1.01	1.01	1.00	1.00	0.15	0.11
445.gobmk	1.07	1.02	1.02	1.02	1.01	1.01	0.38	0.10
447.dealII	1.34	1.09	1.17	1.03	1.02	1.01	0.87	0.11
450.soplex	3.16	1.27	1.43	1.13	1.01	1.10	0.49	0.21
453.povray	1.01	1.01	1.01	1.01	1.00	1.00	0.26	0.09
454.calculix	1.03	1.01	1.01	1.01	1.00	1.00	0.10	0.12
456.hmmer	1.01	1.01	1.01	1.01	1.00	1.01	0.25	0.09
458.sjeng	9.74	1.13	1.79	1.06	1.01	1.01	0.31	0.10
459.GemsFDTD	6.79	4.17	3.28	2.75	0.99	0.99	0.15	0.10
462.libquantum	7.89	1.02	1.29	1.02	1.00	1.01	0.15	0.10
464.h264ref	1.04	1.02	1.02	1.01	1.01	1.00	0.15	0.14

Table 2 Performance of WSS tracing system (contiuned)

表 2 WSS 跟踪的性能优化(续)

程序	程序执行时间						LRU 开启率	
	L	D	A	D+A	D+A+*I*(*f*)	D+A+*I*(*a*)	*I*(*f*)	*I*(*a*)
465.tonto	1.01	1.00	1.01	1.00	1.00	1.01	0.13	0.09
470.lbm	4.31	2.34	1.71	1.65	1.01	1.00	0.17	0.10
471.omnetpp	41.13	1.07	4.60	1.05	1.00	1.04	0.26	0.23
473.astar	15.77	1.02	2.92	1.01	1.01	1.00	0.51	0.13
481.wrf	1.18	1.12	1.12	1.13	1.00	1.00	0.13	0.09
482.sphinx3	1.03	1.01	1.02	1.02	1.00	1.00	0.14	0.09
483.xalancbmk	7.81	1.33	2.28	1.05	1.02	1.00	0.57	0.09
Mean(SPEC2006)	2.73	1.39	1.43	1.16	1.06	1.02	0.26	0.12
Dacapo	1.28	1.07	1.14	1.07	1.04	1.01	0.82	0.20

4.2 单机动态内存调配对虚拟机性能的影响

我们设计了一系列具体的实验来验证单机动态内存调配对虚拟机性能的影响.本实验的目的是:(1) 验证本地内存调配的有效性;(2) 验证优化后的 WSS 跟踪机制对内存调配的性能所产生的影响.

我们在 Xen 上启动了两台虚拟机.在第 1 个实验中,初始化配置给每台虚拟机以 250M 的内存,也就是说它们总的可用内存不超过 500M.其中,一台虚拟机运行 Dacapo,另一台虚拟机运行 SEPC CPU 2000 中的 186.crafty.首先,我们测试了不用内存调控的情况作为基准数据;然后,再加上 WSS 跟踪的机制测试了不同的内存调控效果.

测试结果如图 8 所示,纵轴显示的是标准化之后的性能,Overall 代表两个程序性能的几何平均.L 代表的是基于双向链表的 WSS 跟踪,A+D 是使用 ABL 和 DHS 的 WSS 跟踪.另外两种场景是在 ABL+DHS 的基础上使用 IMT 的 WSS 跟踪.*I*(*f*)使用的是固定阈值 0.2,*I*(*a*)使用自适应阈值.

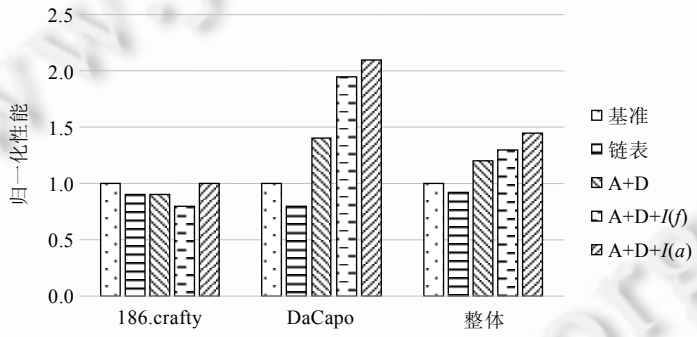


Fig.8 Local memory balancing between Dacapo and 186.crafty

图 8 Dacapo 和 186.crafty 的单机内存调控

图中的数据说明了两点:(1) 内存调控获得了显著的性能提升,Dacapo 中 eclipse 程序的工作集比较大,内存调控机制使运行 crafty 的虚拟机把多余的内存贡献给它,虽然 crafty 的性能略有损失,但是 Dacapo 的页面失效大为减少,从而整体上获得了 1.16 的加速比;(2) 各种针对 WSS 跟踪的优化减小了系统开销,从而进一步提高了性能,尤其是同时使用了 ABL,DHS 和自适应阈值的 IMT 之后,虚拟机整体性能获得了 1.44 倍的提高.

第 2 个实验也是在两台虚拟机上进行的,区别是使用了 WSS 更大的程序作为测试程序,如图 9 所示.初始化给每台虚拟机分配 700M 内存,其中一台运行 470.lbm,另外一台运行 433.milc.当没有使用 IMT 时,WSS 跟踪引起的开销使得 lbm 的性能下降了 10%,但 milc 由于得到更多的内存,性能提高了 2.96 倍;使用 IMT 之后,lbm 的性能损失只有 4%,milk 的性能也进一步提高到 3.06(*IMT*(*f*)),3.56(*IMT*(*a*)).对整个系统而言,最多可以获得 1.22 倍的加速比.

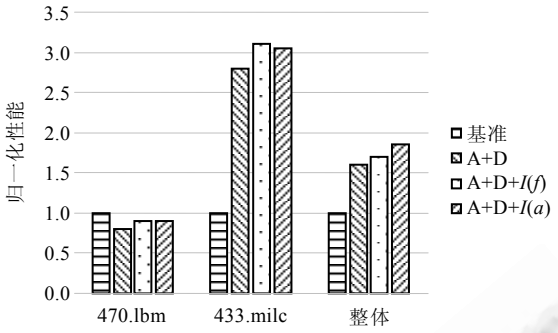


Fig.9 Local memory balancing between lbm and milc

图 9 lbm 和 milc 的单机内存调控

基于气球技术的单机内存调配并不能解决当所有的虚拟机都发生内存紧缺时的问题,一旦发生这种情况,虚拟机只能将页面换出到它的虚拟磁盘上去,而这将导致严重的性能下降.

4.3 多机远程缓存的性能测试

为了观察对比远程缓存和迁移对于不同时长的内存超载的调控影响,我们使用了两台服务器 H_1,H_2,并分别限制在其上运行的虚拟机可用的物理内存为 800MB 和 1 200MB.初始化时,我们在 H_1 上启动一台虚拟机运行 SPEC JBB,H_2 空闲作为内存服务器.按照 *A-B-A* 的模式变化,其中,*B* 代表的就是内存需求高峰.实际操作中,在 *A* 阶段运行了 3 个数据仓库,*B* 阶段运行了 12 个数据仓库.*A* 阶段运行的时间固定为 120s,通过改变 *B* 阶段运行的时间来模拟不同时长的内存超载的情况.

本实验中,我们给 H_2 的远程缓存固定分配了 300M.图 10 显示了不同内存突发时间所对应的 *B* 阶段的吞吐量相对于不使用远程缓存所对应的吞吐量的加速比.

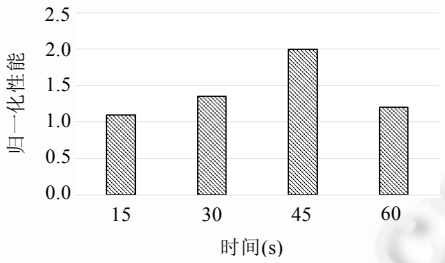


Fig.10 Throughput of JBB's phase B using remote memory

图 10 使用远程缓存的 JBB 的 B 阶段吞吐量

当内存超载时间只有 15s 时,使用远程缓存的性能提升并不明显,因为远程缓存从冷失效到命中需要一个预热的时间;当内存超载时间为 45s 时,远程缓存带来了 2 倍的性能加速比.与迁移的结果比较,我们可以得到如下启示:(1) 对于不同时长的内存突发需求,使用远程缓存不存在开销导致的性能下降问题,说明远程缓存是一个轻量级的调控机制,因此在策略设计上启动远程缓存可以比启动迁移更加激进;(2) 当内存压力继续保持时,由于大批量的页面传送、缓存块替换等开销增大,远程缓存所带来的性能加速比有所下降,此时应当选择迁移代替远程缓存来进行大幅度的内存调整.

接着,我们测试了远程缓存与本地内存调控相结合的系统性能.在 H_1 上启动两台虚拟机 VM_1 和 VM_2,每台分配 250M 内存.服务器 H_2 作为 H_1 的内存服务器.虚拟机 VM_1 按照正序运行 Dacapo,虚拟机 VM_2 按照逆序运行 Dacapo.我们分别统计了两台虚拟机的性能以及产生的缺页中断次数,标准化后的结果见表 3.表中,Baseline 代表不采用任何调控优化的情况,是基准数据;L 代表只使用本地(local)内存调控;R 代表只使用远程(remote)缓存;L+R 代表同时使用本地内存调控和远程缓存的情况;Overall 代表两台虚拟机测试结果的几何平均值.

只考察性能的提升,可以观察到:在只使用本地内存调控或者远程缓存的情况下,总体性能可以分别获得 2.48 和 1.65 倍的加速比;当同时结合二者时,总体性能进一步提高,获得了 3.57 倍的加速比.观察缺页次数的统计结果,可以看到:单机内存调控在降低由于内存不足导致的缺页上发挥了主要作用,而远程缓存能够提高 I/O 请求响应时间,所以二者的结合能够获得最满意的性能.尤其是 eclipse 占据了 Dacapo 运行时间的 1/2 以上,两台

虚拟机运行 eclipse 时有部分重叠,这部分重叠的区间无法用单机的内存调控来加以解决,只能依赖远程交换来缓解压力.

Table 3 Global memory balancing using remote memory

表 3 基于远程缓存的全局内存调控

虚拟机	Baseline		L		R		L+R	
	性能	缺页	性能	缺页	性能	缺页	性能	缺页
VM_1	1	1	2.38	0.36	1.72	0.75	3.57	0.37
VM_2	1	1	2.59	0.36	1.58	0.92	3.56	0.29
Overall	1	1	2.48	0.36	1.65	0.83	3.57	0.33

5 总结与展望

在单物理机上进行资源调配,可以在较大程度上改善本地的内存利用情况.但在虚拟化数据中心里,物理机之间的负载不够均衡,需要在全局范围内进行内存调配.

高效的虚拟化数据中心离不开有效的资源管理机制.本文着眼于内存资源的管理,并重点阐述了如何用动态内存资源调控解决数据中心内存资源利用率低下的问题.理想的内存资源调控应该使虚拟机的可用物理内存能够满足应用程序动态的内存需求,因此,首先需要一个精确的在线内存预测器.结合以前的工作,我们采取了一系列的优化措施,在不影响预测器精度的条件下尽可能降低了系统开销.我们利用处理器的硬件计数器来间断性地进行 WSS 监控,同时改善了 LRU 链表的数据结构,实现了一个精确的内存需求预测器,并将系统开销控制在相当小的范围内.另外,根据内存超载的时间长度,我们设计了两种多机内存资源调整策略:远程内存策略和虚拟机迁移策略.

首先,根据预测结果研究了远程内存策略.在该策略下,所有物理机在逻辑上都是对等的,内存富余的机器承担着内存服务器的角色,并且随着内存负载的变化动态调整角色.一旦发生内存超载的情况,就立刻启动远程内存策略,客户机通过网络使用远程内存服务器上的内存资源以减轻内存负载.

其次,对如何确定全局内存调配策略进行了探讨.在目前的应用场景中,我们不能预测需求持续时间,当虚拟机资源不足时,就开启远程内存.但是远程内存的性能还是无法与本地内存相比,如果内存持续超载,则应启用虚拟机迁移,并且这两个过程的目的主机应尽量保证相同,通过重用远程内存的成果来缩短迁移时间.而远程内存开启的时间则是我们研究的重点,找到一个合适的阈值,可以在很大程度上提升系统的内存效率.

现有的多种内存调控或负载均衡技术扩大了本地服务器可用的物理内存范围,允许在本地物理内存紧缺的情况下使用远程机器的空闲内存来缓解内存压力.虽然利用这些技术可以更灵活地进行多机间的内存调控、整合多机的资源、提高系统整体性能,但同时也不能忽略这些方法本身所产生的系统开销.虚拟化环境中,最常见的迁移比较适合解决预期持久的内存超载,否则,迁移本身的代价可能会高过迁移后获得更多内存所带来的性能提升.而远程缓存交换技术则比较适合暂时性的内存超载,能够以微不足道的代价及时响应内存抖动,从而带来整体的性能提升.

我们在 Xen 上实现了基于远程缓存和基于虚拟机迁移的全局内存调配器.实验结果表明:该内存调配机制可与现有的 ballooning 技术兼容,在本地内存调配器的协调下,大大提高了虚拟机群的整体性能.

本文所阐述的工作主要提供了进行内存动态调整的机制,下一步工作的重心则是对内存调整的策略进行更加深入的研究,针对不同的资源需求模式评估内存资源调整的代价和调整后对系统整体性能所带来的影响,据此来确定内存调配的机制和调整发生的时机,进而更精准地进行动态内存调配.

References:

[1] Goldberg RP. Survey of virtual machine research. Computer, 1974,7(6):34–45. [doi: 10.1109/MC.1974.6323581]

[2] Waldspurger CA. Memory resource management in VMware ESX server. ACM SIGOPS Operating Systems Review, 2002,36(SI): 181–194. [doi: 10.1145/844128.844146]

[3] Clark C, Fraser K, Hand S, Hansen JC, Jul E, Limpach C, Pratt I, Warfield A. Live migration of virtual machines. In: Proc. of the 2nd Conf. on Symp. on Networked Systems Design & Implementation, Vol.2. Boston: USENIX Association, 2005. 273−286.

[4] Jones ST, Arpaci-Dusseau AC, Arpaci-Dusseau RH. Geiger: Monitoring the buffer cache in a virtual machine environment. ACM SIGOPS Operating Systems Review, 2006,40(5):14−24. [doi: 10.1145/1168917.1168861]

[5] Zhao WM, Wang ZL. Dynamic memory balancing for virtual machines. ACM SIGOPS Operating Systems Review, 2009,43(3): 37−47. [doi: 10.1145/1618525.1618530]

[6] Comer D, Griffioen J. A new design for distributed systems: The remote memory model. In: Proc. of the USENIX Summer Conf. USENIX Association, 1990. 127−135.

[7] Wood T, Shenoy P, Ramakrishnan KK, Van der Merwe J. CloudNet: Dynamic pooling of cloud resources by live WAN migration of virtual machines. ACM SIGPLAN Notices, 2011,46(7):121−132. [doi: 10.1145/2007477.1952699]

[8] Williams D, Weatherspoon H, Jamjoom H, Liu YH. Overdriver: Handling memory overload in an oversubscribed cloud. ACM SIGPLAN Notices, 2011,46(7):205−216. [doi: 10.1145/2007477.1952709]

[9] Shen X, Zhong Y, Ding C. Locality phase prediction. ACM SIGOPS Operating Systems Review, 2004,38(5):165−176. [doi: 10.1145/1037949.1024414]

[10] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual, 2013.

[11] Luk CK, Cohn R, Muth R, Patil H, Klauser A, Lowney G, Wallace S, Reddi VJ, Hazelwood K. Pin: Building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005,40(6):190−200. [doi: 10.1145/1064978.1065034]

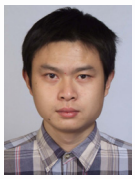
[12] Chen HG, Wang XL, Luo YW, Wang ZL, Wen X, Zhang BB, Sun YF. REMOCA: Hypervisor remote disk cache. In: Proc. of the 2009 IEEE Int'l Symp. on Parallel and Distributed Processing with Applications. IEEE, 2009. 161−169. [doi: 10.1109/ISPA.2009.27]

[13] Chen ZF, Zhou YY, Li K. Eviction-Based cache placement for storage caches. In: Proc. of the USENIX Annual Technical Conf. 2003. 269−281.

[14] Sherwood T, Perelman E, Calder B. Basic block distribution analysis to find periodic behavior and simulation points in applications. In: Proc. of the 2001 Int'l Conf. on Parallel Architectures and Compilation Techniques. IEEE, 2001. 3−14. [doi: 10.1109/PACT.2001.953283]

[15] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebaner R, Pratt I, Warfield A. Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, 2003,37(5):164−177. [doi: 10.1145/1165389.945462]

[16] Blackburn SM, Garner R, Hoffmann C, *et al*. The DaCapo benchmarks: Java benchmarking development and analysis. ACM SIGPLAN Notices, 2006,41(10):169−190. [doi: 10.1145/1167515.1167488]

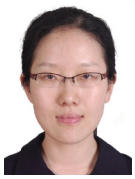


王志钢(1985－),男,江西吉安人,博士生,主要研究领域为系统虚拟化,云计算.
E-mail: pkuwzg@pku.edu.cn

王振林(1970－),男,博士,副教授,博士生导师,主要研究领域为云计算,高能效计算,编译技术.
E-mail: zlwang@mtu.edu

汪小林(1972－),男,博士,副教授,CCF 会员,主要研究领域为系统虚拟化,高能效计算.
E-mail: wxl@pku.edu.cn

罗英伟(1971－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为系统虚拟化,云计算,高能效计算.
E-mail: lyw@pku.edu.cn



靳辛欣(1987－),女,博士生,主要研究领域为云计算,高能效计算.
E-mail: xinxinjin89@gmail.com