ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2012/06(01)-0058-09
DOI: 10.3778/j.issn.1673-9418.2012.01.004

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-51616056

面向虚拟机环境的 Cache 动态划分算法设计与实现*

李家文+, 沈 立
国防科学技术大学 计算机学院, 长沙 410073

Design and Implementation of Dynamic Cache Partitioning in Virtualization Environment*

LI Jiawen+, SHEN Li
School of Computer Science, National University of Defense Technology, Changsha 410073, China
+ Corresponding author: E-mail: jwenl2008@126.com

LI Jiawen, SHEN Li. Design and implementation of dynamic cache partitioning in virtualization environment. Journal of Frontiers of Computer Science and Technology, 2012, 6(1): 58–66.

Abstract: In order to improve cache isolation performance and system performance in virtualization environment, this paper designs and implements a dynamic cache partitioning algorithm. It uses page coloring strategy to implement cache partitioning, in which private color pages are allocated to virtual machine (VM) and cache capacity can be adjusted on demand. The paper implements the method in Xen virtualization environment. Experimental results show that the method can boost system performance for applications running concurrently on different VMs with low overhead.

Key words: dynamic cache partitioning; performance isolation; virtual machine

摘 要：为改善虚拟化系统的 cache 隔离性, 提高系统的整体性能, 面向虚拟化环境设计并实现了一种 cache 动态划分算法。该算法采用页面着色的思想, 通过为虚拟机分配私有颜色页面来实现 cache 的划分, 同时能够根据虚拟机的 cache 需求为其动态调整 cache 容量。在 Xen 虚拟环境中实现了该算法。实验结果表明, 该算法可以在较低开销的情况下, 显著提高多虚拟机上并发程序的全局性能。

关键词：动态 cache 划分; 性能隔离; 虚拟机

*The National Natural Science Foundation of China under Grant No. 60813041 (国家自然科学基金); the National Grand Basic Research 973 Program of China under Grant No. 2007CB310901 (国家重点基础研究发展规划(973)).
Received 2011-07, Accepted 2011-09.

文献标识码：A 中图分类号：TP316

1 引言

虚拟化环境下, 资源管理一直是研究者关注的热点, 对于传统对称处理器(symmetric multiprocessors, SMP), 由于 cache 为单核所私有, 使得虚拟机之间 cache 隔离未引起研究者的重视。近年来, 随着片上多核处理器(chip of multiprocessors, CMP)的迅猛发展, 多处理器之间共享 cache 所导致的竞争问题已经引起了人们的广泛关注[1−2]。对于该问题, 研究者多采用静态或动态共享 cache 划分的方法, 来隔离程序之间的影响。

总结前人研究可以发现, 对共享 cache 的划分方法可以分为以下三种：(1) 修改 LRU(least recently used)替换策略; (2) cache 路替换策略[3]; (3) 页面着色策略。修改 LRU 替换策略的优点是具有 cache 块的粒度, 但对于多组相连的 cache, 实现代价非常高。cache 路替换策略(column caching)通过限制每个处理器可以访问的 cache 路来达到划分的目的, 但相对来说改变分区也会带来较高的代价。前两种方法都需要特殊的硬件支持[4−7]。页面着色算法[8]可以由纯软件实现, 通过限制从虚拟地址到物理地址的映射, 来约束每个处理器能够访问的 cache 组, 但它实现的粒度较大。表 1 是三种方法优缺点的比较。

Table 1 Comparison of cache partitioning strategies

表 1 cache 划分策略比较

比较项目	修改 LRU 替换策略	cache 路替换策略	页面着色策略
粒度	cache 块	cache 路	cache 组
实现方式	硬件辅助	硬件辅助	纯软件
可扩展性	差	差	好
可移植性	差	差	好

Suh 等人[9]针对当前应用广泛的组相连 cache, 提出了一种通用的动态 cache 划分机制, 分别使用修改 LRU 替换策略和 cache 路替换策略, 在模拟器中进行实验, 结果表明与标准的 LRU 替换策略相比, 可以明显地提高程序的 IPC(inter-process communication)。

针对目前大多数研究都是在模拟器上实现的情况, Lin 等人[10]首次在真实的操作系统上实现了一个基于页面着色的动态 cache 划分策略。相比于模拟环境中的实现, 在真实系统中的实现不但可以更好地发现存在的问题, 还可以提供一个真实可靠的方法来评估多核下 cache 划分策略。

对于虚拟化环境, Jin 等人[11]在 Xen 中实现了一个基于页面着色的静态 cache 划分策略, 其具有以下特点：

(1) 通过对内存页面进行着色处理, 不同的虚拟机只能访问分配给它的颜色对应的页面, 在隔离内存的基础上很自然地实现了 cache 的划分, 可以有效实现 cache 资源的性能隔离。

(2) 作为静态方法, 在虚拟机启动前已经设置好对应的颜色, 虚拟机启动后自动初始化所分配的页面, 以后无须再进行页面分配, 开销几乎为零。

静态 cache 划分方法在拥有以上优点的同时, 也存在一些难以克服的局限性：

(1) 多虚拟机并发执行时, 无法动态地调整每个虚拟机的 cache 大小, 只能够凭经验在启动之初对 cache 进行划分。

(2) 程序执行过程中, 其运行特征(如 L2 cache 失效率)并非一成不变。如图 1 所示, 程序运行特征的这种动态变化无法被静态划分算法所捕获, 从而也无法充分利用这种动态变化所可能带来的收益。

动态 cache 划分方法可以在运行时捕获程序的 L2 cache 失效率, 通过合理调整虚拟机之间 cache 比例, 最大限度地降低整个系统的 cache 失效率, 从而在实现良好隔离性的同时获得整体性能的提升。

本文实现了一种虚拟环境下 cache 动态划分算法。该算法可以有效地捕获程序对 cache 资源的需求, 在保证良好的资源性能隔离的同时, 获得了较好的性能提升。本文研究工作的贡献在于：

(1) 实现了虚拟化环境下 cache 的动态划分。

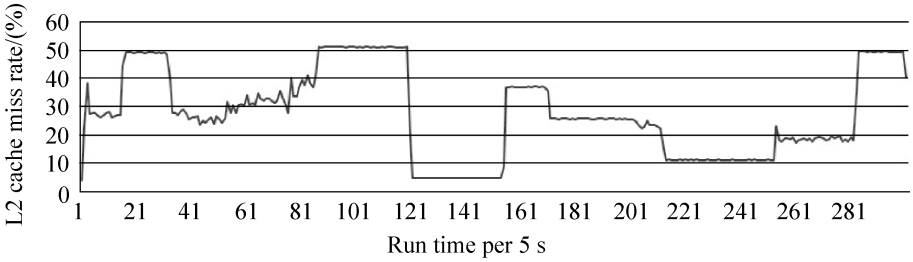

Fig.1 L2 cache miss rate of astar in 1 MB L2 cache

图 **1** **astar** 在 **1 MB L2 cache** 下的失效率

(2) 可以实时地捕获程序对 cache 的需求，从而对 cache 大小进行动态调整，获得良好的性能收益。

(3) 所实现的动态划分算法开销较小，无须修改操作系统，可以应用于多种虚拟机环境。

本文组织结构如下：第 2 章阐述了虚拟机环境下 cache 动态划分算法的设计与实现；第 3 章是实验与结果分析；第 4 章对全文进行了总结。

2 动态 cache 划分策略

在虚拟化环境下实现 cache 的动态划分，可以采用页面着色的方法，根据虚拟机对 cache 的需求动态调整其拥有的内存页面。本文实现了一种基于页面着色的动态划分算法。它在虚拟机启动时预先分配一定的 cache 资源，在运行过程中，通过捕获程序 L2 cache 失效率的变化，采用自适应策略对各虚拟机的 cache 资源进行合理调整，达到降低系统 cache 失效率，提高系统整体性能的目的。该策略包括两部分：动态调整虚拟机的颜色数；根据虚拟机颜色的变化调整内存页面。下面首先介绍 Xen 的内存管理。

2.1 Xen 内存分配与虚拟地址转换

作为虚拟化系统中的关键技术，Xen 内存虚拟化具有举足轻重的地位。在实现动态 cache 划分时，主要涉及内存分配和虚拟地址转换两个方面。

2.1.1 内存分配

对于使用页面着色算法进行 cache 划分来说，需要在 Xen 内存管理模型中添加一个按页面颜色进行分配的内存分配器[11]，它将内存页面着色后挂载到不同的颜色链表中，当虚拟机进行内存请求时，按其拥有的颜色分配对应的页面。不同的虚拟机分配出去的页面由于颜色不同而映射到不同的 cache 组中，从而隔离了虚拟机使用的 cache 资源。在动态划分算法实现时，虚拟机的页面分配都是通过该分配器完成的。

2.1.2 虚拟地址转换

在 Linux 系统中，虚拟地址的转换是通过内存管理单元(memory management unit, MMU)的硬件电路完成的，其中由 MMU 的分页单元完成由线性地址到物理地址的转换。与操作系统中的“线性地址→物理地址”两级映射关系不同的是，Xen 虚拟系统中使用了“线性地址→伪物理地址→机器地址”三级映射关系。在硬件辅助虚拟化模式下，使用影子页表完成地址的转换，虚拟机操作系统对所拥有的页面进行访问都要受虚拟机监视器(virtual machine monitor, VMM)的监控，这也是本文动态算法进行页面替换的基础。

2.2 动态调整颜色数

使用图 2 所示算法来实现虚拟机颜色数的动态调整，算法假定同时运行两个虚拟机。

在算法中，设置了 5 个优先级队列，每隔一个时间段(实现时设为 5 s)，比较当前活动的各虚拟机的 L2 cache 失效率，将它们放置到不同队列中。分析可知，虚拟机在以下三种情况可能发生颜色数的改变：

(1) 虚拟机的创建和关闭。虚拟机创建时应该设定最初分配的颜色数，该算法中初始分配每个虚拟机 16 种颜色，对应分配 1 GB 内存。虚拟机关闭时只需简单地将其颜色全部释放即可。

```
Algorithm: Dynamic Color Adjusting for VMs
Initialize each VM with N colors;
While Programs are running
    Put Doms to different priority queues according to L2 cache miss rate;
    Count the sum of Doms cache miss rate;
    If the free color > 0
        Select the highest priority and no empty queue, allocate each Dom one color in it;
    Else
        Try to select a strategy according M (M = sum of cache miss rate now – sum of
do  cache miss prev, and M is initialized as 0)
        S0: when M >= 0
            If Dom1's colors > N
                Free Dom1 one color and allocate Dom2 one color;
        S1: when M < 0
            If Dom2's colors > N
                Free Dom2 one color and allocate Dom1 one color;
    When finished, shutdown Dom and free its colors.
```

Fig.2 Dynamic color adjusting algorithm

图 **2** 动态颜色调整算法

(2) 虚拟机运行中颜色分配。虚拟机启动后，还有一部分空闲颜色未分配，本阶段算法的目的就是合理地将其分配出去。首先判断当前是否有空闲颜色存在，若存在空闲颜色，则选取非空最高优先级队列，依次为队列中的每个虚拟机分配一种颜色；当没有空闲颜色时，进入下一步算法。

(3) 虚拟机运行中颜色调整。在空闲颜色全部被分配完毕后，尝试采用以下策略调整虚拟机的颜色数：回收 Dom1 一种颜色，分配 Dom2 一种颜色，运行一个时间段后，计算当前所有虚拟机总的 L2 cache 失效率，若此失效率小于先前的失效率，则说明此次分配策略是正确的，下一次仍然回收 Dom1 颜色，分配 Dom2 颜色；若当前失效率大于先前的失效率，则说明这次分配策略是错误的，下一次需要回收 Dom2 颜色，分配 Dom1 颜色。

2.3 页面的动态替换

依据虚拟机对 cache 的需求进行颜色的动态调整后，必须按颜色对虚拟机的页面进行动态替换，才能实现 cache 大小的变化。针对 VMM 中特殊的页面管理方式，当需要进行页面替换时，必须在完成页面拷贝后，更新对应虚拟机的影子页表，以便依据虚拟地址能够找到正确的机器页面。当前实现的是一种最简单的页面替换算法：假定先前总的页面数为 P，颜色数为 N，则每当增加一种颜色时，都要依次替换第 N+1 个页面、第 2N+1 个页面……共计 $P/(N+1)$个页面。简单分析可知，这种算法将会带来大量的页面拷贝，系统开销非常大。在实现部分将说明如何用 lazy method 将系统开销约束到一个可以接受的范围。

2.4 算法实现

在 Xen3.3.0 VMM 中实现了本文的策略。实现时主要解决两个问题：

(1) L2 cache 失效率的采集。

(2) 页面替换开销的优化。

2.4.1 cache 失效率的采集

作为动态 cache 划分的先置条件，L2 cache 失效率的捕获非常重要。实现过程中通过调用 X86 处理器提供的性能监控单元(performance monitor unit, PMU)来实现失效率的采集[12]。在系统初始化时，根据不同的处理器架构为 PMU 写入相应的 Mask；当各虚拟机中程序运行时启动 PMU 进行 L2 cache 失效率的采集，建立一个全局数组来存放失效率；每当 vcpu 进行调度时，更新相应虚拟机的失效率；当时间段结束时，通过比较虚拟机的失效率，将其放

入不同的优先级队列中, 并将全局数组清零。

2.4.2 页面替换算法的优化

为降低页面替换算法开销, 从两个方面进行优化：

(1) 在程序运行过程中, 若本次测得的总失效率仅与上次相差 1%或者更少, 按算法应该进行分区调整, 但实际上此时进行分区调整并不能获得收益, 反而可能带来系统开销, 导致性能降低。因此在实际算法实现时, 将阈值设定为 5%, 只有超过 5%的变化才会去调整分区。

(2) 程序运行过程中许多页面拷贝并非必要的, 某些页面在进行替换后, 未被虚拟机访问就又被其他页面替换出去, 从而导致了系统的额外开销。针对此种情况, 使用 lazy method 来进行页面的替换, 即只有在虚拟机访问新页面时才进行页面拷贝, 这样可以大大减少页面拷贝操作。具体实现如下：首先为每个虚拟机建立一个链表, 用于存放替换前的机器页面号, 每次进行页面替换时, 判断此页面是否被替换过, 若未被替换过, 则将该页面标记为已被替换, 并将其机器页面号写入链表中。当虚拟机访问某虚拟地址时, 根据影子页表找到对应的机器页面, 若此页面被替换过, 则查找链表中对应项, 将旧的页面内容拷贝到新的页面中。实验测试发现, 该方法可以有效降低系统的开销。

3 实验与结果分析

3.1 实验环境

实验硬件平台配置如下：Intel Core2 Q6600 2.40 GHz 处理器, 内存为 6 GB DDR2 800 SDRAM。该处理器有四个核, 每两个核共享一个 4 MB 的 L2 cache(16 路组相连, cache 块 64 Byte)。每个核都有一个 32 KB 的指令 cache 和一个 32 KB 的数据 cache (8 路组相连, cache 块 64 Byte)。此 L2 cache 共可划分为 64 种颜色(系统最大颜色数=cache 容量/关联度/页面大小)。

软件环境如下：虚拟机管理器为实现了动态 cache 划分的 Xen3.3.0, 特权域(Dom0)为 CentOS 5.4, 内核版本为 Linux-2.6.21; 非特权域(DomU)使用 Red Hat Enterprise Linux 5.5, 内核为 Linux-2.6.18, 运行在全虚拟化模式下, 16 种颜色, 1 GB 内存。

3.2 工作负载和评价指标

采用 SPEC CPU 2006[13]测试程序集中部分程序的标准参考输入(ref)作为实验的工作负载。依据程序对 cache 的需求将其划分为以下三种类型：

(1) cache 敏感型程序(S)。cache 大小由 1 MB 增大到 4 MB 时, 程序性能提升大于 15%。

(2) cache 污染型程序(P)。cache 大小由 1 MB 增大到 4 MB 时, 程序性能提升小于15%, 且 L2 cache 失效率总是大于 25%。

(3) cache 无关型程序(U)。cache 大小由 1 MB 增大到 4 MB 时, 程序性能提升小于 15%, 且 L2 cache 失效率低于 25%。

依据在静态划分中的测试结果, 将选定的测试程序区分如下, 见表 2。

Table 2 Benchmark classification

表 **2** 测试程序分类

分类	1 MB→4 MB	4 MB	测试程序
S	>15%	–	bzip2, astar, omnetpp
P	<15%	>25%	milc, libquantum, bwaves
U	<15%	<25%	hmmer, sjeng, gromacs

cache 敏感型程序可以从较大的 cache 中获益, 因此应该分配较大的内存; cache 污染型程序往往具有流程序特点, 其工作集很大且局部性不好, 无法从大 cache 中获益, 只需分配最小 cache 即可; cache 无关型程序工作集较小, 预分配的 cache 往往可以满足其需要。因此, 为测试本文的动态划分算法, 构建工作负载如表 3 所示, 括号内为选定的静态划分比例。

为评价动态 cache 划分算法的性能, 着重分析了以下两个参数：(1) 平均加速比(average speedup); (2) 总的 L2 cache 失效率(total cache miss rate)。为便于比较, 测试了上述工作负载在未划分、静态划分和动态划分三种情况下的性能, 并根据测试结果进行了相应分析。

Table 3 Workloads
表 **3** 工作负载

类型组合	工作负载
S+S	SS1: bzip2 + astar (32∶32)
	SS2: bzip2 + omnetpp (32∶32)
S+P	SP1: bzip2 + milc (48∶16)
	SP2: astar + libquantum (48∶16)
S+U	SU1: astar + sjeng (48∶16)
	SU2: bzip2 + gromacs (48∶16)
P+P	PP1: milc + libquantum (16∶48)
	PP2: bwaves + milc (32∶32)
P+U	PU1: milc + sjeng (32∶32)
	PU2: bwaves + hmmer (16∶48)
U+U	UU1: gromacs + sjeng (48∶16)
	UU2: sjeng + hmmer (16∶48)

3.3 实验结果

实验测试了工作负载在上述三种情况下的平均加速比和总的 L2 cache 失效率。采取的方法是为 Dom0 只分配一个 vcpu，并将其绑定到 cpu0 上，避免对其他虚拟机产生影响；为两个虚拟机各分配一个 vcpu，分别绑定到共享同一 L2 cache 的两个物理核 cpu2 和 cpu3 上。测试结果见图 3、图 4。

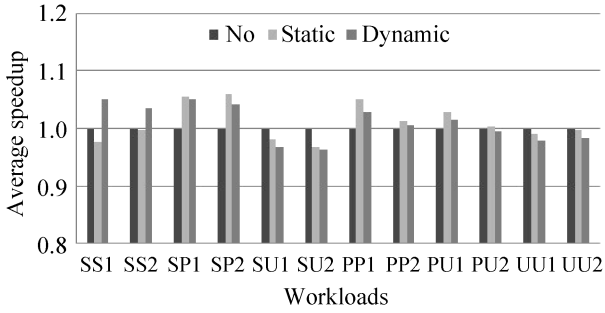

Fig.3 Average speedup of workloads in three configurations
图 **3** 三种情况下工作负载平均加速比

从图 3 和图 4 中可以看出，在多数情况下，动态划分都获得了大于 1 的加速比，而很多情况下，静态划分的加速比都小于 1。对于不同类型组合，S+S 情况下，动态划分要比静态划分好，S+P 和 P+P 情况下，动态划分要比静态划分差，但也要好于未划分的情况。这是由于以下几点原因：

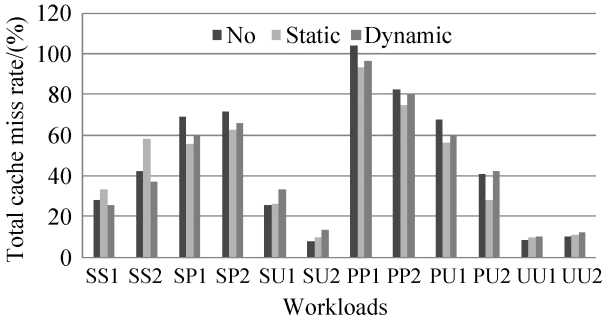

Fig.4 Total L2 cache miss rate of workloads in three configurations
图 **4** 三种情况下工作负载总的 **L2 cache** 失效率

(1) 对于静态划分，为简化实验过程，只是选择了几个简单分区比例中较优的一个，但是实际上并不一定就是最优的分区方案。

(2) 对于未划分的情况，虽然程序之间存在着竞争关系，但是对于两个工作集都剧烈波动的程序，同时运行有可能得到很大的益处，这点从 SS1 中可以明显地看到。

(3) 对于动态划分，它一方面避免了程序之间的竞争，另一方面能够及时捕获程序工作集的变化，因此能够通过分区的调整获得较好的收益。

3.4 性能分析

本节以工作负载 SS1(bzip2+astar)为例分析系统的性能。作为 cache 敏感型的两个程序 bzip2 和 astar，其特点是能够从较大的 cache 中获得较高的性能提升，这点从以往的研究中都能够发现。通常认为，bzip2 和 astar 同时运行，由于程序之间竞争 cache，从而导致未划分的性能低于静态划分的性能。但实验结果正好相反，其原因在上节有所描述，在此将进行深入分析。

图 5~图 7 是工作负载 SS1 在三种情况下每一个时间段(实验中选择为 5 s)的 L2 cache 失效率。从图中可以清楚地看到工作负载在每一个时间段的失效率的具体情况。为便于观察，选取 4 个有特点的区域进行数据分析，分别标记为 A、B、C、D。

图中 A 区位于时间段 1 至 26 之间，之所以选取这段时间，是因为结合动态划分算法，空闲颜色将会在这段时间内分配完毕。从失效率可以看出，对

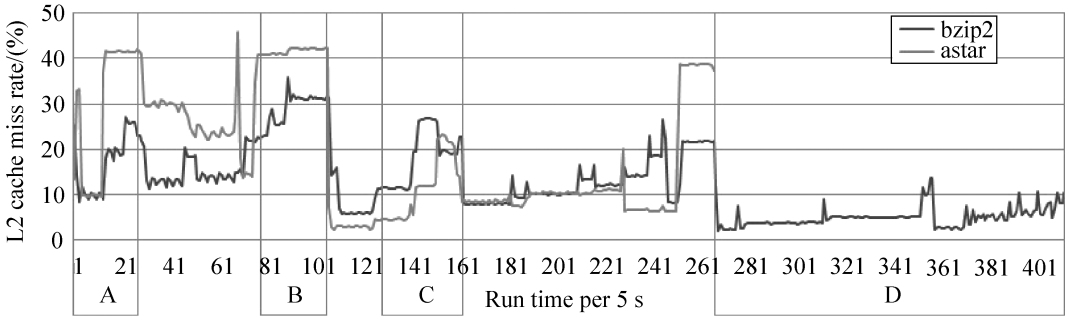

Fig.5 L2 cache miss rate in unpartitioning

图 **5** 未划分情况下 **L2 cache** 失效率

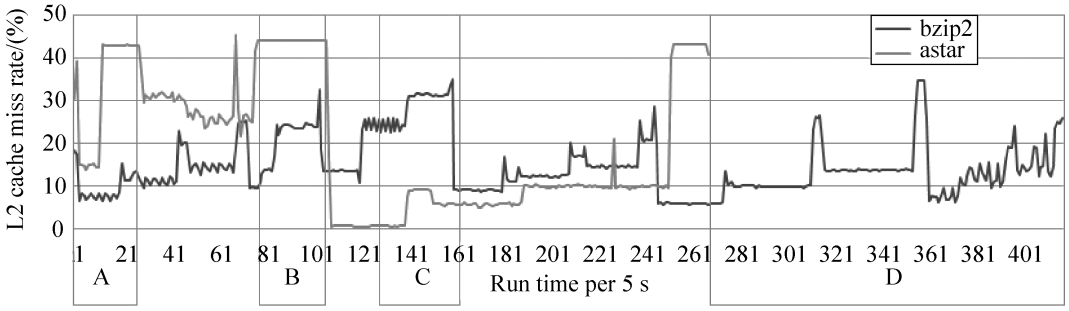

Fig.6 L2 cache miss rate in static partitioning

图 **6** 静态划分时 **L2 cache** 失效率

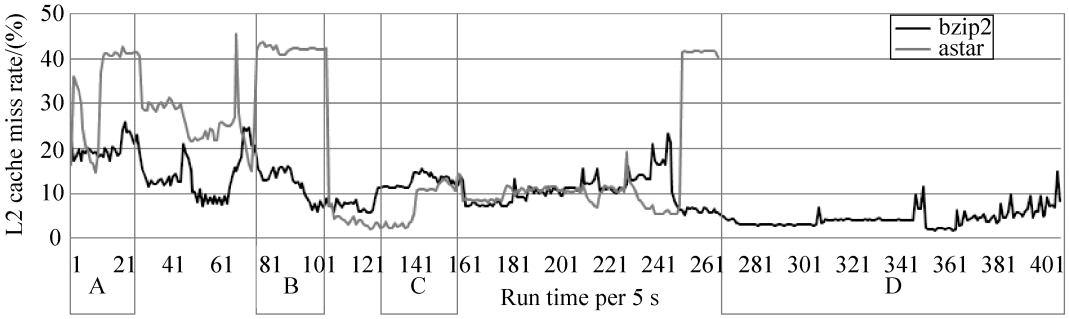

Fig.7 L2 cache miss rate in dynamic partitioning

图 **7** 动态划分时 **L2 cache** 失效率

于动态划分，这段时间内 bzip2 和 astar 都获得了新的颜色，但因为其颜色数是缓慢增加的，即所使用的 cache 空间是缓慢增大的，所以效果没有静态划分好；对于未划分情况来说，bzip2 在这段时间内工作集较小，因此 astar 也就可以使用较多的 cache，从而也获得了较低的 cache 失效率(时间段 1 至 10 之间大概为 10%)。

对于 B 区，首先看静态划分情况，发现 bzip2 和 astar 的失效率都很高(前者在 25%左右，后者高于 40%)；对于未划分的情况，由于二者对 cache 的需求都很大，存在很强的竞争关系，竞争的结果是 bzip2 的失效率急剧增大到 30%以上，astar 的失效率略微减小，系统的整体性能下降；对于动态划分来说，由于 astar 工作集极大，导致失效率一直维持在 40%以上，此时根据动态算法，会不断减少 astar 的颜色，增加 bzip2 的颜色，从而使得 bzip2 所用的 cache 增大，失效率降低，这点从图 7 B 区可以看出。

对于 C 区，首先看静态划分，在 32∶32 的分区

比例下，前半部分，bzip2 的失效率在 25%左右，astar 的失效率几乎为零，即 astar 此时对 cache 的需求极低，因此在未划分的情况下，bzip2 能够利用较多的 cache，从而获得较好的性能(此部分失效率在 10%左右)；动态划分也可以捕获到这点，从而充分利用资源(图 7 C 区前半部分，失效率 11%左右)。对于后半部分，从静态划分看，bzip2 此时失效率急剧上升到 30%以上，而同时 astar 的失效率也超过 10%；未划分情况下，二者都需要较大的 cache，从而引发竞争，总的失效率上升(图 5 C 区后半部分，bzip2 失效率 26%，astar 失效率 22%)；动态划分却可以在避免竞争的同时进行权衡，从而获得较低的总失效率(图 7 C 区后半部分，bizp2 失效率 15%，astar 失效率 12%)。

最后是 D 区，此时 astar 已经运行完毕(实验中 astar 运行完毕后虚拟机自行关闭)，静态划分因为 bzip2 的 cache 仍为 2 MB，所以它并不能获得其他虚拟机释放的 cache，从而也就不能获得性能的提升。而对于未划分和动态划分的情况，此时 bzip2 可以获得接近 4 MB 的 cache，性能获得极大提升。

4 总结

本文提出并实现了一种简单有效的虚拟机动态 cache 划分机制，它能够在保证虚拟机之间良好的 cache 隔离性的同时，实时地捕获程序运行时的特征，通过动态调整虚拟机可以使用的 cache 大小，获得较好的性能提升。实验结果表明，本文方法在较低的系统开销下，在多数情况下，可以获得比未划分方法高的性能加速比，在多个敏感型程序同时运行的情况下，可以获得比静态划分方法高的性能。

致谢 感谢北大“973”虚拟化小组提供的 CAPA 静态 cache 划分工具，感谢温翔同学在实验环境搭建上提供的支持。

References:

[1] Chang Jichuan, Sohi G S. Cooperative cache partitioning for chip multiprocessors[C]//Proceedings of the 21st ACM International Conference on Supercomputing (ICS ’07), Seattle, USA, June 18-20, 2007. New York, NY, USA: ACM, 2007: 242–252.

[2] Liu Chun, Sivasubramaniam A, Kandemir M. Organizing the last line of defense before hitting the memory wall for CMPs[C]//Proceedings of the 10th International Symposium on High Performance Computer Architecture (HPCA ’04), Madrid, Spain, Feb 14-18, 2004. Washington, DC, USA: IEEE Computer Society, 2004: 176–185.

[3] Chiou D T. Extending the reach of microprocessors: column and curious caching[D]. Massachusetts: Massachusetts Institute of Technology, 1999.

[4] Lee T, Tsou H. A novel cache mapping scheme for dynamic set-based cache partitioning[C]//Proceedings of the 2009 IEEE Youth Conference on Information, Computing and Telecommunication (YC-ICT ’09), Beijing, China, Sept 20-21, 2009. Washington, DC, USA: IEEE Computer Society, 2009: 459–462.

[5] Qureshi M K, Patt Y N. Utility-based cache partitioning: a low-overhead, high-performance, runtime mechanism to partition shared caches[C]//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO ’06), Orlando, USA, Dec 9-13, 2006. Washington, DC, USA: IEEE Computer Society, 2006: 423–432.

[6] Rafique N, Lim W-T, Thottethodi M. Architectural support for operating system-driven CMP cache management[C]//Proceedings of the 15th Parallel Architectures and Compilation Techniques Conference (PACT ’06), Seattle, USA, Sept 16-20, 2006. New York, NY, USA: ACM, 2006: 2–12.

[7] Cho S, Jin Lei. Managing distributed, shared L2 caches through OS-level page allocation[C]//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO ’06), Orlando, USA, Dec 9-13, 2006. New York, NY, USA: ACM, 2006: 455–468.

[8] Kessler R E, Hill M D. Page placement algorithms for large real-indexed caches[J]. ACM Transactions on Computer Systems, 1992, 10(4): 338–359.

[9] Suh G E, Rudolph L, Devadas S. Dynamic partitioning of shared cache memory[J]. The Journal of Supercomputing, 2004, 28(1): 7–26.

[10] Lin Jiang, Lu Qingda, Ding Xiaoning, et al. Gaining insights into multicore cache partitioning: bridging the gap between simulation and real systems[C]//Proceedings of the 14th International Symposium on High Performance Computer Architecture (HPCA '08), Salt Lake, USA, Feb 16-20, 2008. Washington, DC, USA: IEEE Computer Society, 2008: 367–378.

[11] Jin Xinxin, Chen Haogang, Wang Xiaolin, et al. A simple cache partitioning approach in a virtualization environment[C]//Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing with Application (ISPA '09), Chengdu, China, Aug 9-11, 2009. Washington, DC, USA: IEEE Computer Society, 2009: 519–524.

[12] Intel 64 and IA-32 architectures software developer's manual, volume 3B[CP/OL]. [2011-06]. http://intel.com/products/processor/manuals/.

[13] SPEC CPU2006 Standard performance evaluation corporation[S/OL]. [2011-06]. http://www.spec.org.

LI Jiawen was born in 1981. He is a master candidate at National University of Defense Technology. His research interest is computer virtualization.

李家文(1981—), 男, 山东五莲人, 国防科学技术大学硕士研究生, 主要研究领域为计算系统虚拟化。

SHEN Li was born in 1976. He is an associate professor and master supervisor at National University of Defense Technology, and the member of CCF. His research interests include computer architecture, compiler optimization and computer virtualization, etc.

沈立(1976—), 男, 陕西宝鸡人, 国防科学技术大学副教授、硕士生导师, CCF 会员, 主要研究领域为计算机系统结构, 编译优化, 计算系统虚拟化等。