小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems

2012 年 10 月 第 10 期
Vol. 33 No. 10 2012

# 一种面向 SSD-HDD 混合存储的热区跟踪替换算法

刘圣卓[1,2],姜进磊[1,2],杨广文[1,2]

[1](清华大学 计算机科学与技术系,北京 100084)
[2](清华信息科学与技术国家实验室(筹),北京 100084)
E-mail: szhuoliu@gmail.com

**摘 要:** 固态驱动器(SSD)读写性能优越,但成本高,因此在实践中人们往往利用 SSD 和普通硬盘(HDD)构建混合存储系统以获取较高的性价比. 在混合存储系统中,如何使更多的 IO 请求能够命中 SSD 是充分利用 SSD 性能的关键. 针对多任务共享存储环境下集中访问和随机访问 IO 存取模式并存,且通常情况下 IO 工作流大部分请求相对集中于有限区域内的特点,本文提出一种基于热区跟踪(HZT)的缓存替换算法. HZT 算法充分考虑了 IO 工作流的空间局部性和时间局部性,利用 IO 工作流的历史访问信息,跟踪当前热区,并为热区数据块赋予更高的驻留 SSD 的优先级,能够有效提高混合存储中 SSD 缓存的命中率. 经测试,在典型多任务共享存储环境下 HZT 算法可以使 SSD 缓存的命中率比使用 LRU(Least Recently Used)算法的系统提高 12%. 采用适当的预取策略,该算法的命中率与 LRU 算法相比可获得 23% 的提升.

**关 键 词:** 共享存储; 混合存储系统; 替换算法; 固态驱动器
**中图分类号:** TP301 **文献标识码:** A **文 章 编 号:** 1000-1220(2012)10-2255-04

## A Hot-Zone-Tracing-based Replacement Algorithm for Hybrid SSD-HDD Storage System

LIU Sheng-zhuo[1,2], JIANG Jin-lei[1,2], YANG Guang-wen[1,2]

[1](*Department of Computer Science & Technology, Tsinghua University, Beijing* 100084, *China*)
[2](*Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing* 100084, *China*)

**Abstract:** The read and write performance of Solid State Drive (SSD) is superior, but the cost is very high. Therefore, it becomes practical to construct hybrid storage system with SSD and hard disk drive (HDD) to obtain higher cost-performance ratio. In such a hybrid storage system, making more IO requests hit the data on SSD is a key point to take full advantage of SSD. Based on our observation that concentrated access pattern and random access pattern coexist in shared storage environment and in most cases nearly all IO requests concentrate in limited space, we put forward a new cache replacement algorithm called HZT. HZT algorithm defines the concept of hot zone and utilizes the history of IO requests to trace current hot zones. In addition, data blocks in hot zones are assigned to a high priority, meaning they can stay in SSD without being replaced whenever possible. Such a way takes full account of spatial locality and temporal locality of IO workload in hybrid storage system and therefore, can improve the hit rate of SSD buffer efficiently in shared storage environments. The experimental results show that HZT algorithm can achieve 12% more hit rate of SSD buffer than LRU (Least Recently Used) algorithm. When combined with appropriate prefetching strategy, the hit rate of HZT algorithm can be improved by 23% than that of LRU algorithm.

**Key words:** shared storage; hybrid storage system; replacement algorithm; solid state drive



## 1 引 言

随着多核技术和云计算的出现与快速发展,存储系统被越来越多的处理单元或任务所共享,存储性能变得日益重要. 在这一情况下,人们开始探索新的存储技术以满足应用对于存储性能的要求,SSD(固态驱动器)就是其中的一种.

SSD 是由 Flash(闪存)存储阵列构成的存储设备,通过其控制器内实现的闪存转换层 FTL(Flash Translate Layer)[1],对外提供与普通硬盘(HDD)相同的访问接口,系统可以通过逻辑扇区号(LBN)以扇区为单位对其访问. SSD 读写性能大大优于 HDD,尤其是随机读写性能. 此外 SSD 还有能耗低、重量轻、抗震性能好等优点. 尽管如此,SSD 的单位成本要远远高于 HDD. 使用 SSD 替换 HDD 固然可以获得巨大的性能提升,但由此带来的成本开销也是十分昂贵的. 因此,采用 SSD 和 HDD 构建混合存储系统成为兼顾性能与成本的折中方案.

由于 SSD 具有读取速度方面的优势,因此将系统常用的文件或数据块保存在 SSD 中可以有效提高系统的 I/O 性能. 然而,SSD 容量占混合存储系统中的比例较小,因此需要根据系统的访问情况动态替换 SSD 中的数据块,以确保更多的数据块请求可以从SSD中获取,这是典型的缓存替换问题,替

收稿日期: 2012-06-30 收修改稿日期: 2012-08-13 基金项目: 国家"九七三"重点基础研究发展计划项目(2011CB302505)资助; 国家自然科学基金项目(61073165, 61170210)资助; 国家"八六三"高技术研究发展计划项目(2011AA040505, 2010AA012401)资助. 作者简介: 刘圣卓,男,1975 年生,博士研究生,主要研究方向为云计算、分布式存储系统; 姜进磊,男,1975 年生,博士,副教授,主要研究方向为分布式计算、工作流管理; 杨广文,男,1963 年生,博士,教授,博士生导师,主要研究方向为数据网格、云计算、分布式存储系统.

换算法的优劣将直接影响 SSD 的利用效率和混合存储系统的整体性能. 虽然混合存储系统可以使用 Cache 系统的替换算法,如 LRU( Least Recently Used) 和 FIFO( First In First Out) 等,但在一些应用环境下这些算法的表现并不能令人满意.

本文的目标是面向云计算这样的多任务共享存储系统的环境,设计适合 SSD-HDD 混合存储系统使用的替换算法. 论文的主要工作是: 针对多任务共享存储环境中 I/O 工作流的特性,通过分析 I/O 访问的历史,设计实现了基于系统 I/O 访问热点跟踪的替换算法,能够避免发生 SSD 数据块替换时,热点区域数据块被替换掉.

论文工作的主要贡献为: 1) 分析得到了多任务共享存储环境中 I/O 存取模式的特性,即多种 I/O 存取模式同时存在,但 I/O 工作流的访问热点相对集中,且随着任务的执行,I/O 工作流的热点也将随之发生变化; 2) 针对多任务共享存储环境的特点,提出了 I/O 访问热区识别方法,能够兼顾 I/O 工作流的空间局部性和时间局部性; 3) 提出了一种高效的数据结构用来保存数据块的访问历史信息,能够降低内存开销.

## 2 相关工作

混合存储系统中 Flash 或 SSD 的容量通常较小,高效的替换算法是系统性能的必要保证. 由于混合存储系统的原理与 Cache 系统相似,一些早期的混合存储系统大多采用 Cache 系统中的替换算法. Marsh 等人用 Flash 作为二级文件系统缓存以减少耗能和存取延时[2] ,并采用 FIFO 和 LRU 替换算法. Conquest[3] 则将具有电池支持的 RAM 合并到基于 HDD 的存储系统中. 用持久 RAM 来缓存小文件和元数据,其他数据则放在 HDD 上. 一些系统[4,5] 使用小容量的 flash 设备作为内存的扩展或 HDD 的缓存以加快 I/O 访问速度. Differentiated Storage Services[6] 利用 HDD 和 SSD 组成混合存储系统,并通过修改操作系统的内核,使 I/O 系统可以区分数据请求的类型. 并根据请求类型采取选择性分配和选择性替换算法向 SSD 中调入数据块或清除其上的数据块,以提高 I/O 系统性能.

与将 SSD 做为存储系统的一个新层次不同,一些研究人员提出将 SSD 和 HDD 放在同一个层次上使用. ComboDriver[7] 利用桥设备把 SSD 和 HDD 连接成统一的存储空间,以文件为单位采用按照文件类型的静态放置算法和根据存取特征的动态替换算法. 文献 [8] 提出将读密集的数据块放到 SSD 上,写密集的数据块放到 HDD 上,该系统采用的替换算法兼顾数据块的存取频率和替换代价.

Hystor[9] 使用三级树结构保存每个数据块访问历史,并根据数据块的访问历史信息选定需要被替换的数据块. Hystor[9] 根据请求的大小和频率来度量一个数据块是否将被替换,对 I/O 工作流热点变化不敏感,不适合在访问热点经常变化的多任务环境下使用. 因为需要为每个被存取过的数据块分配一个记录存取历史信息的内存单元,所以内存开销比较大.

## 3 多任务环境的 I/O 工作流

不同类型的任务运行时对数据的访问模式通常存在较大的差异. 例如邮件服务器的读取模式是以小文件的随机读写为主. 而一些高性能运算 I/O 存取密集且访问请求相对集中. 在云计算环境下,多任务共享底层存储的情况十分普遍,针对底层存储的集中 I/O 访问和随机 I/O 访问同时存在. 我们分析了典型的多虚拟机服务器环境下的 I/O 工作流,利用 blktrace[10] 采集了运行 4 个 web 服务器和 1 个邮件服务器的虚拟机环境的 I/O 工作流. 通过分析 I/O 请求的分布情况,我们发现了明显的集中访问和随机访问的 I/O 工作流特性.

我们将存储空间分成大小一致的若干个连续的区( Zone) . 当区的大小为 256MB 时,我们分别统计了各个区在一段时间内的访问次数. 从统计的结果,我们观察到每个区的访问次数差异很大. 我们将一个时期内区内数据块访问次数相对多的区称为这个时期的“热区( Hot Zone) ”,访问少的区叫做“冷区( Cold Zone) ”. 集中访问模式的大部分 I/O 工作流都集中在一定数量的热区内,显然热区的数据块被再次访问的概率高于冷区. 冷区的访问大部分是由随机 I/O 工作流产生的请求.

## 4 热区跟踪替换算法设计

因为成本原因,混合存储系统中的 SSD 远小于硬盘的存储空间,通常占整个存储空间的 5% 到 10% . 当 SSD 缓存的空间装满后,需要根据适当的替换算法选择被替换的数据块. 我们根据热区的特性设计了热区跟踪( Hot Zone Tracing,HZT) 算法,算法包括热区管理、替换方法、预取策略等 3 个部分.

### 4.1 热区管理

根据热区的定义,一段时间内,热区的访问次数要高于冷区的访问次数. 我们可以简单地把热度值定义为一个区内数据块被访问的次数. 但这样做有两个问题: 一是不能有效识别当前时间段的热区; 二是查找热区效率低.

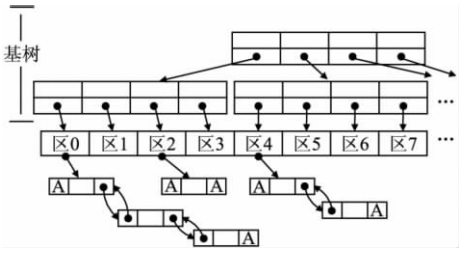


图 1 区( zone) 的管理结构

Fig. 1 Management structure of zones

为提高内存利用效率,我们使用了类似于 Linux 内核的页高速缓存机制的基树( Radix Tree) 的数据结构来管理区访问历史. 基树的叶子节点指向每个区,每个叶子节点记录区的热度值( 访问计数) 、区数据块计数、节点访问计数. 非叶子节点记录其所有子节点的汇总信息. 每个节点包含的子节点的个数称为基,基树的高度由基树的基和区的个数决定. 基数的叶子节点包含一个指向该区数据块 LRU 链表的指针. 图 1 给出了一个基为 4 的 2 层基树. 当一个区内的数据块被访问时,该区的热度值增加,其所有父节点的热度值也将增加.

使用基树数据结构的优点是查找速度快、占用空间小. 设基树的基为 R,层数为 L,区的个数为 A,则 $L = \lceil \log_R A \rceil$ . 以

1TB 存储空间为例,块的大小为 4KB,如果一个区包含 1K 个块,基树的基为 64,则基树的层数为 3 层. 在选择替换块时,需要 64* 3 = 192 次查找. 基树共包括 1 + 64 + 64* 64 = 4161 个节点,记录每个区的信息需要 6 个字节,则共需 1.5MB 左右的空间保存基树.

热度值需要反映区内数据块的近期访问和远期访问情况. 设 $h_{i,j}$ 表示第 i 层第 j 个节点的热度值. 热度值的计算方法:

1) 每次对区内数据块的访问,该区的热度值加 1,该节点所有父节点的热度值也相应加 1:

$$h_{R-i,n/(Ri)} = h_{R-i,n/(Ri)} + 1(1 \leqslant i \leqslant L) \quad (1)$$

2) 当热度值溢出时,本节点所有热度值右移 1 位(除以 2):

$$h_{i,j} = \frac{1}{2} h_{i,j}(i/R \leqslant j \leqslant (i/R + R-1)) \quad (2)$$

3) 当节点访问计数大于一定阈值时,节点所有热度值右移 1 位,同时访问计数清"0",热度值得计算与公式(2)相同.

随着访问量增加,热度值可能溢出(尤其是上层节点),节点内所有热度值右移 1 位不会改变本节点内的热度排序,所以不需要修改同层不同节点和不同层节点. 当节点访问计数大于一定阈值时节点内所有区的热值右移 1 位可以隔离历史访问信息. 热值右移 1 位相当于使这一个时刻之前的热度值的权重降为 1/2,使近期的访问量在热度值中占有更高的权重. 当一个区内的数据块长时间没有访问时,其热度值将不断变小直至变为 0,成为冷区.

### 4.2　替换方法

当需要调入数据块且 SSD 的读写缓存已满时,就从已有数据块中选择替换块. 被替换数据块的选取原则是从 SSD 中热度值最低的冷区中选取,当被选中的冷区包含多个数据块时,则依据 LRU 算法选取. 冷区的查找是从根节点开始的,首先找到本层热度值最低且在 SSD 中有数据块的区节点,然后依次向下一层查找,最后检索到的叶子节点所指向的区就是要查找的冷区. SSD 缓存的数据块的调入由后台模块定期或空闲时完成.

### 4.3　预取策略

根据多任务的 I/O 工作流具有顺序访问和区域集中的特性,可以采取预取策略增加命中率. 可以采用两种预取策略: 一是定期扫描热区的信息,选择热度值高的区,调入这些区的未调入的数据块. 二是在数据块调入时,查看该区的热度值是否达到一定门限,如果是,则试图调入该数据块之后的几个数据块. 我们采用的是后一种预取策略.

## 5　测试平台实现

我们在 linux 系统下的通用块层实现了一个混合存储系统的原型,采用逻辑块映射的方法将整个存储空间分别映射到 SSD 和 HDD 上. 混合存储系统将经常访问的数据块调入 SSD,当上层请求这些数据块时,就可以从 SSD 上读取所需数据. 当 SSD 装满后,系统将使用 HZT 替换算法选取数据块进行替换. 该原型包括映射模块和管理模块两部分.

如图 2,映射模块位于系统的通用块层和块驱动层之间. 我们采用和 Hystor[3] 类似的方法,在 linux 内核的 MD(multi-ply driver)模块的基础上,修改了部分代码实现了映射模块. 从用户角度,HDD 和 SSD 作为一个整体存储空间使用. 映射

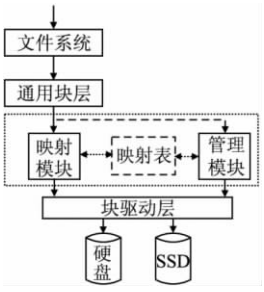


图 2　实验原型在操作系统中位置

Fig. 2　Location of experiment prototype in OS

模块接收通用块层的 I/O 请求. 如果是读操作,首先查找该块是否在映射表中,如果在则将请求映射到 SSD 上,否则向 HDD 请求数据; 若不在映射表,则通知管理模块调入该数据块. 管理模块接收映射模块发送的 I/O 工作流信息,维护热区基树、完成数据块的调入,并负责管理映射表.

## 6　实验结果

测试平台为 Intel® Xeon® 2.4GHz 4 核 CPU,4GB 内存. 操作系统为 Ubuntu 1104,内核版本是 2.6.38.8. 我们使用两块普通硬盘(型号为 WDC WD2500AAJS-75B4A0,容量 250GB),一块 SSD(型号为 TEAM XS2 SSD,容量 60GB,只使用 40GB)组成了混合存储系统.

我们在测试平台上运行了 9 个虚拟机任务,其中任务 1-8 为 web 服务器,其存取模式为集中式 I/O,任务 9 为邮件服务器,其存取模式为随机式 I/O. 任务 9 工作在 1,000 个本地目录,每个目录平均有 2,000 个文件,文件的大小为 2KB 到 32KB. 在所有 I/O 操作中,读操作占 85%,同步写操作的比例为 7.2%,异步写操作的比例为 7.8%. 我们在该多任务环境下对使用 LRU、HZT 和带预取策略的 HZT(Pre-HZT)算法进行对比测试. 每个区包含 256 个块,基树的基为 64. 预取的热度值门限为 30,每次预取 4 个块.

### 6.1　缓存大小的影响

我们在不同的缓存大小的情况下,测试了 3 种算法的命中率,以确定缓存大小对命中率的影响.

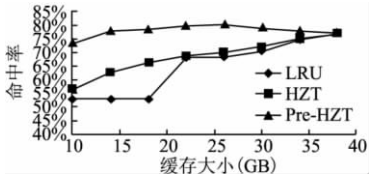


图 3　不同缓存大小的命中率

Fig. 3　Hit rate of various buffer size

从图 3 的测试结果可以看出,LRU 和 HZT 算法的命中率均随着缓存的大小增大而增加. 在 10GB 到 20GB 的缓存容量下,HZT 的命中率都明显高于 LRU 算法. 当缓存容量超过 34GB,两者的命中率相同,这是因为 SSD 缓存已经覆盖了整

个工作集. 其中在 18GB 大小的缓存条件下,HZT 比 LRU 的命中率高 12.7%. 出现这种情况主要是在缓存满后,LRU 算法用新出现的随机数据块替换了热区数据块; 而 HZT 算法则从访问量少的随机请求的区内选取替换块,使得热区数据块保留在缓存中. 对于 Pre-HZT 算法在大部分情况下命中率都高于前两者,最高时比 LRU 算法高 23%.

图 4 显示了采用三种算法的混合存储系统在不同缓存容量下的 I/O 访问的累计响应延时. 实验结果表明 HZT 算法优于LRU 算法. 在 SSD 为 18 GB 时,HZT 算法比 LRU 的响应时间下降 23%. Pre-HZT 算法则优于前两个算法,但在缓存较小时差距不大,这是由于预取操作产生了额外的开销. 在 SSD 大于 14GB 时,预取策略使系统的性能得到明显提升. 实验表明,在多任务环境下,HZT 算法在适当缓存大小的情况下缓存效率明显优于 LRU 算法. 通过适当预取数据块,可以使算法的性能进一步提升.

### 6.2 任务切换的影响

为测试 HZT 算法在任务切换的多任务环境下的性能,我们设计了下面的实验. 我们先运行任务 1-4,当这些任务完成后立即运行任务 5-8,而任务 9 则一直运行.

我们在不同缓存的情况下测试了 3 种算法的命中率情况,测试结果如图 5 所示. 在任务切换的环境中 HZT 算法在缓存增大时命中率上升速度大于 LRU 算法. 这是由于在任务切换条件下,HZT 算法能快速识别访问热区,避免热区数据块被随机数据块替换. 当 SSD 容量为 10GB 时 HZT 算法命中率比 LRU 算法高 9.7%,Pre-HZT 算法的命中率比 LRU 算法高 15.1%.

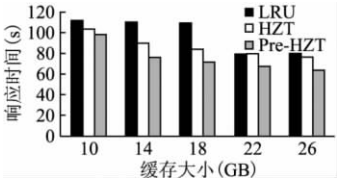


图 4 不同缓存大小的响应时间

Fig. 4 Response time of various buffer size

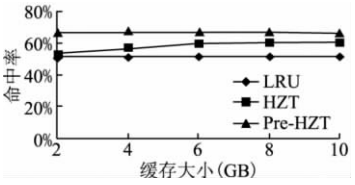


图 5 任务切换环境的命中率

Fig. 5 Hit rate of task switching

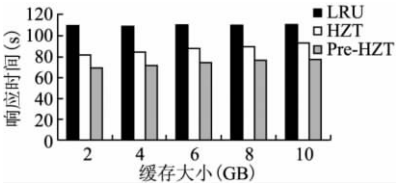


图 6 任务切换环境的响应时间

Fig. 6 Response time of task switching

图 6 是在任务切换环境下三种算法的累积响应时间. 可以看出 HZT 和 Pre-HZT 的响应时间明显小于 LRU 算法的响应时间. 实验表明,HZT算法比LRU算法更适合在I/O工作流热点不断变化的多任务环境下使用,采用适当的预取策略可以有效提升命中率.

### 6.3 实验结果分析

在多任务环境下,替换算法需要同时兼顾空间局部性和时间局部性. LRU 替换算法使得热区的数据块有可能被新到达的冷区的数据块所取代,从而使 SSD 缓存的命中率降低. HZT 替换算法通过有效识别热区的数据块,并为热区的块赋予较高的驻留缓存的优先级,有效提高 SSD 的命中率.

从实验结果可以看出,HZT 算法通过记录每个区内数据块在一段时间内的访问次数,通过为近期的访问次数赋予更高的权值的方法为每个区赋予不同的热值. 相比 Hystor[9] 记录每个数据块的访问历史的方法,以区为单位的记录方式更能准确识别多任务环境下 I/O 访问的热区,同时可以降低内存开销.

## 7 结论及进一步工作

混合存储方案以块为单位,将系统经常访问的数据块调入 SSD,同时将写回的数据块暂存在 SSD 上,降低系统的 I/O 访问延时. 利用基于热区跟踪的 HZT 替换算法充分利用 I/O 工作流的时间局部性和空间局部性,有效隔离历史访问信息识别当前时期热区,提高多任务环境下的缓存命中率. 在任务频繁切换的多任务环境下,HZT 替换算法的性能在大部分情况下都优于 LRU 算法. 附加适当的预取策略使 HZT 算法的命中率得到进一步提升. HZT 算法以区为单位使用基树管理数据块访问历史,大大降低内存开销,提高查找速度.

**References:**


[1] Corporation I. Understanding the flash translation layer (FTL) specification [EB/OL]. http://www.embeddedfreebsd.org/Documents/Intel-FTL.pdf, 2011.

[2] Marsh B, Douglis F, Krishnan P. Flash memory file caching for mobile computers [C]. Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, Hawaii, USA, 1994: 451-460.

[3] Wang A-IA, Kuenning G, Reiher P, et al. The conquest file system: better performance through a disk/persistent-RAM hybrid design [J]. Trans. Storage, 2006, 2(3): 309-348.

[4] Microsoft. Microsoft Windows readyboost [EB/OL]. http://windows.microsoft.com/en-US/windows7/products/features/readyboost, 2011.

[5] Matthews J, Trika S, Hensgen D, et al. Intel® Turbo memory: nonvolatile disk caches in the storage hierarchy of mainstream computer systems [J]. Trans. Storage, 2008, 4(2): 4:1-4:24.

[6] Mesnier M, Chen F, Luo T, et al. Differentiated storage services [A]. Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles [C], New York, USA: ACM, 2011: 57-70.

[7] Payer H, Sanvido M, Bandic Z Z, et al. Combo drive: optimizing cost and performance in a heterogeneous storage device [C]. First Workshop on Integrating Solid-state Memory into the Storage Hierarchy, 2009: 1-8.

[8] Koltsidas I, Viglas S D. Flashing up the storage layer [C]. Proceedings of International Conference on Very Large Data Bases, Auckland, New Zealand, 2008, 1(1): 514-525.

[9] Chen F, Koufaty D A, Zhang X. Hystor: making the best use of solid state drives in high performance storage systems [A]. Proceedings of the International Conference on Supercomputing [C], New York, USA: ACM, 2011: 22-32.

[10] Blktrace [EB/OL]. http://linux.die.net/man/8/blktrace, 2011.