FClock:一种面向 SSD 的自适应缓冲区管理算法

汤 显 孟小峰

(中国人民大学信息学院 北京 100872)

摘 要 现有的各种基于闪存的缓冲区管理算法针对闪存读写代价的不对称性进行改进,实际中既存在同一闪存读写代价的不对称性问题,也存在不同闪存不对称性之间的巨大差异性问题,而后者一直没有得到足够的重视.文章提出一种基于闪存硬盘(SSD)的自适应缓冲区管理算法 FClock,FClock 将数据页组织为两个环形数据结构(*CC* 和 *DC*),分别用于存储缓冲区中的只读数据页和已修改数据页.当需要选择置换页时,FClock 使用基于代价的启发式来选择置换页,可在未修改的数据页和已修改的数据页之间进行公平的选择,适用于不同种类的 SSD.针对数据库、虚存和文件系统中数据页访问存在高相关性的特点,提出基于"平均命中距离"的访问计数方法来调整数据页的访问频率.基于不同 SSD 和不同存取模式的实验结果说明,FClock 的综合性能优于已有方法.

关键词 闪存;数据库;缓冲区;置换策略;CLOCK

中图法分类号 TP391 **DOI 号**: 10.3724/SP.J.1016.2010.01460

FClock: An Adaptive Buffer Replacement Algorithm for SSD

TANG Xian MENG Xiao-Feng

(*School of Information, Renmin University of China, Beijing* 100872)

Abstract Different from existing flash-aware buffer replacement policies that focus on the asymmetry of read and write operations, the authors address the "discrepancy" of the asymmetry for different flash disks, which is the fact that exists for a long time, while has drawn little attention by researchers since most existing flash-aware buffer replacement polices are somewhat based on the assumption that the cost of read operation is neglectable compared with that of write operation. This paper proposes an adaptive replacement policy (FClock) which has two ring-shaped data structures, i.e. *CC* (their content remain unchanged) and *DC* (their content is modified), to manage clean pages and dirty pages in the buffer, respectively. When selecting a victim page, FClock uses cost-based heuristics to fairly make trade off between clean pages and dirty pages, and hence, can work well for different type of flash disks of large discrepancy. Further, for the problem of "correlated references" to database, virtual memory and file systems, this paper proposes a reference counter based on "average hit distance" to control the reference frequency. The experimental results on different traces and flash disks show that FClock not only adaptively tunes itself to workloads of different access patterns, but also works well for different kind of flash disks compared with existing methods.

Keywords flash; database; buffer; replacement policy; CLOCK

收稿日期:2010-06-11.本课题得到国家自然科学基金(60833005,60573091)、国家"八六三"高技术研究发展计划项目基金(2007AA01Z155,2009AA011904)、教育部博士点基金项目(200800020002)资助.**汤 显**,女,1978 年生,博士研究生,主要研究方向为闪存数据库.E-mail:txianz@gmail.com.**孟小峰**,男,1964 年生,教授,博士生导师,研究领域包括网络数据管理、云数据管理、移动数据管理、XML 数据管理、闪存数据库系统以及隐私保护.

1 引　言

基于闪存的存储设备以其低延迟、低能耗、小巧轻便及高抗震性等特点广泛应用于移动设备上,随着闪存容量的不断增大和价格的降低,其应用领域已逐步扩展到个人计算机和企业服务器市场.过去几年 NAND 型闪存的容量不断增长,并且这种趋势将至少持续到 2012 年[1].目前各种应用中都将闪存硬盘 SSD 看成一个块设备并使用与磁盘一样的存取接口,但这两种硬盘的 I/O 特性却存在很大的差异.闪存硬盘的随机读速度远快于其随机写速度,在一些对性能要求苛刻或者涉及频繁数据处理的应用场合,如数据库服务器,如果不能根据闪存的特性来设计合适的数据结构和算法,就难以获得最佳性能.

缓冲区是现代计算机最基本的组成部分之一.它在存储系统、数据库、网络服务器、文件系统以及操作系统中都有广泛的应用.缓冲区置换算法的任何进展都会影响现代计算机的整体性能.假设用磁盘做辅存且读写操作的时间延迟相同,那么对于给定大小的缓冲区,现存的缓冲区置换算法[2-8]的目标就是最小化缓冲区的缺页率.当缓冲区已满并且请求的数据页不在缓冲区中时,缓冲区置换算法首先从当前缓冲区选择一个用于置换的数据页,如果所选的数据页是脏页(其内容被修改过),就必须将该页的内容写回硬盘,然后才能将请求的数据页读入缓冲区,以保证数据的一致性.这种操作可能会成为系统性能的瓶颈,这是因为请求页的线程或进程必须等待写操作的完成.早在 20 年前,Effelsberg 等[9]已经认识到缓冲区中数据页的读写状态是影响置换策略的重要因素,由于闪存硬盘有望替代磁盘成为新一代的数据存储设备,并且闪存硬盘的读写代价存在不对称性问题,在设计置换策略时更应考虑读写状态的差异性.

针对闪存读写操作的不对称性问题,近几年研究者已经提出了几种适用于闪存的缓冲区置换策略[10-14].然而,这些置换策略在实际中存在以下问题:

(1)适用的闪存硬盘类型有限

已有的基于闪存的缓冲区置换算法的基本假设是闪存的随机读代价相对于随机写代价来说可忽略不计,因此这些缓冲区置换策略都是通过无条件先置换只读页来减少随机写操作的次数,从而提高系统的性能.然而,如图 1 所示,该假设和实际情况并不相符,即随机读代价和随机写代价相比,并非在所有情况下都可以被无条件忽略.尽管所有的闪存设备都表现出较快的随机读速度和较慢的随机写速度,但对于不同的闪存设备而言,读写代价的比例差别很大,显然在不考虑只读页的操作代价和访问频率的情况下就无条件首先置换只读页是不合理的.

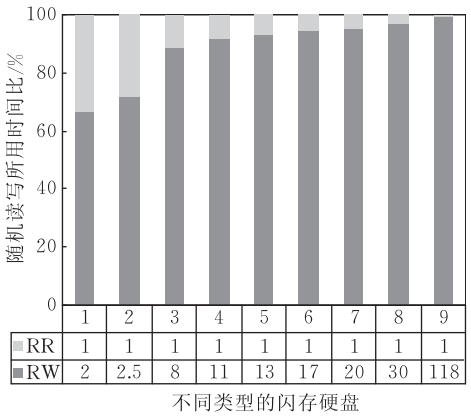

图 1　NAND 闪存硬盘随机读(RR)和随机写(RW)消耗的时间比例(X 轴为 9 种闪存硬盘,1 是三星 MCAQE32G8APP-0XA,2 是三星 K9WAG08U1A,3 是三星 K9XXG08UXM,4 是三星 K9F1208R0B,5 是三星 K9GAG08B0M,6 是现代 HY27SA1G1M,7 是三星 K9K1208U0A,8 是三星 K9F2808Q0B,9 是三星 MCAQE32G5APP①)

(2)多线程处理能力不足

已有的基于闪存的缓冲区置换策略都是基于 LRU 置换策略进行改进,LRU 的优势在于它实现简单,操作代价低,但 LRU 也有自身的局限性:①每次命中的数据页必须移动到最近使用(MRU)的位置.实际中多个线程可能都试图移动各自的数据页到 MRU 位置,而 MRU 位置通过锁保护来保证一致性和正确性.由于所有命中操作都在等待该操作的完成,就会引起大量锁争用的问题.在高性能和高吞吐量环境中,例如虚存、数据库、文件系统和存储控制器中,这种情况是不可接受的.②LRU 没考虑数据页的"访问频率".和 LRU 对应,CLOCK 算法克服了 LRU 算法的上述缺点.通过将数据页组织成时钟形式的环形缓冲区,CLOCK 算法无需在每次命中数据页后移动数据页的位置,因此不会出现锁争用的问题.因此,CLOCK 算法在实际系统中,如 DB2、SQL Server、Postgresql 等,得到了广泛应用.

针对现有基于闪存的缓冲区置换算法存在的问题,本文提出一种基于闪存的 clock 算法 FClock 来解决以上问题.FClock 为每个数据页维护一个"访问位"并将缓冲区中的数据页组织成两个时钟形式

① http://www.datasheetcatalog.net

的环形结构 *CC*(Clean Clock)和 *DC*(Dirty Clock),分别用于管理未修改的数据页和修改过的数据页.当数据页被初次读入缓冲区时,它的访问位被置为0.当命中某个数据页时,将其访问位加1.当需要选择一个置换页时,FClock 并非直接置换出未修改页,而是首先根据启发式规则判断被置换的数据页是已修改页还是未修改页,然后再从 *DC* 或者 *CC* 中找访问位是0的数据页进行置换,以此来进行自适应的调整.最后,受文献[3]的"局部性过滤"原则的启发,FClock 使用基于平均命中距离的技术来消除短期频繁访问而长期不访问的数据页长时间驻留内存的问题.

2 背景及相关工作

2.1 闪存存储器

一般来说,有两种不同类型的闪存芯片,分别是NOR 型闪存和 NAND 型闪存.NOR 型闪存芯片和 EPROM 以及 SRAM 一样,有专用的地址和数据总线;而 NAND 闪存芯片无专用的地址和数据总线,用一个 IO 接口来控制输入输出.NOR 型闪存芯片可用来替换可编程的只读存储器(PROM)和可擦除的 PROM(EPROM)来进行有效的随机存取;NAND 型闪存芯片由于其存储容量较高,主要用来存储数据.闪存硬盘(SSD)中使用的通常是 NAND 型芯片.

NAND 型闪存芯片上有3种基本操作:读、写和擦除.读和写都是以数据页为单位进行操作;擦除是以块为单位进行操作,一个块通常包含64个页.NAND 型芯片不支持原地更新,如果某个页上有数据,就无法对该页直接进行覆盖写操作.为了避免对某些块进行频繁的写和擦除操作之后所造成的数据块失效的问题,通常使用磨损平衡技术将写和擦除操作均匀地分布在所有的数据块上.

为了克服闪存芯片的物理限制,闪存硬盘利用一个软件层来模拟块设备的功能,并尽量使得擦除操作的延迟不为用户所见,这个软件层通常称为闪存转换层(FTL),它一般存储在 SSD 的 ROM 芯片中.FTL 的主要作用是将对一个数据页的写请求重新映射到一个已擦除的空白数据页上.因此,FTL 需要维护一个内部映射表来记录逻辑地址和物理地址之间的映射信息.该映射表在系统启动时构造,并在 SSD 的易失性存储器中进行维护.FTL 的实现细节与具体的设备相关,由制造商提供,对用户是透明的.

2.2 缓冲区置换策略

典型的计算机系统包含两层存储器,分别是主存(缓冲区)和辅存(外部存储介质,如磁盘或者 SSD).缓冲区的存取速度远快于辅存,二者一般使用相同大小的数据页.

已有基于磁盘的缓冲区置换策略[2-8]假定每次置换操作的代价相同,其目标是最小化缓冲区的缺页率.缺页率反映了必须从辅存读入缓冲区的数据页的比例.CLOCK[2]、FBR[3]、LRU-K[4]、2Q[5]、LIRS[6]、ARC[7]和 LRFU[8]等算法主要通过使用启发式方法来提高系统的性能,通过考虑数据页在缓冲区中的滞留时间和使用频率来减少缺页率.由于闪存的读写时间不对称,以上假设对于闪存来说是不成立的.因此,当设计基于闪存的缓冲区管理算法时,需要考虑读写的不对称性问题.

具有不对称存取时间的缓冲区置换问题可模拟成加权缓冲区问题,其目的是最小化请求序列的总代价.针对该问题,文献[15]提出了复杂度为 $O(sn^2)$ 的最优离线算法,其中 s 表示缓冲区中数据页的个数,n 表示请求序列的长度.该问题可进一步归结为最小代价最大流问题[16]进行求解.由于该算法的时间和空间复杂度很高,即使提前知道完整的请求序列,其运行也需要耗费大量的时间和空间资源.对于在线算法,不可能提前知道任何未来的请求序列.研究者已经提出了许多在线的基于闪存的缓冲区管理算法.

基于闪存的缓冲区置换策略(FAB)[11]维护了一个块层 LRU 链表,同一物理块的数据页被聚集到一起.FAB 主要用在多数写请求都是顺序写的便携式媒体播放器上.

BPLRU[12]也维护了一个块层 LRU 链表.与 FAB 不同,BPLRU 使用 SSD 内部的 RAM 作为缓冲区,将随机写变成顺序写来提高写操作的效率和减少擦除操作的次数.

CFLRU[10]是利用闪存读写性能的不对称性提出一种优先置换只读页的缓冲区置换策略,这种策略假设闪存的写代价远远大于读代价.CFLRU 的基本思想如图2所示.其中 LRU 链表分成两个部分:工作区(Working Region)和置换区(Clean-First Region).每当发生缺页中断时,如果在置换区中存在只读的数据页,CFLRU 就会从中选择最近最少使用的只读页进行置换,如图2的 $p6$.只有当置换区中没有只读页时,才选择链表尾部的修改页 $p7$ 进行置换.置换区的大小是由参数 w 控制的.

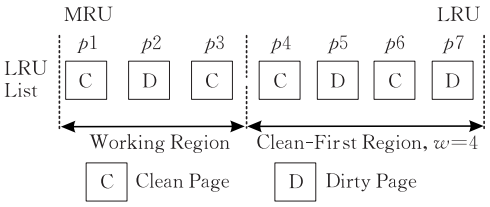

图 2　CFLRU 置换策略示意图

基于相同的思想,文献[13]将 CFLRU 置换区中的数据页根据其修改状态组织为不同的队列,从而可以将选择置换页操作的时间复杂度降为 $O(1)$. CFDC[14]通过对 CFLRU 置换区中的数据页进行重新组织来提升 CFLRU 算法的执行效率. 如图 3 所示,CFDC 的缓冲区也分为两部分,分别是工作区(Working Region)和置换区(Priority Region). 在 CFDC 的置换区中,根据数据页是否为修改页将其组织到两个队列中,其中只读页放在 Clean queue 中,所有的修改页放在不同的 Cluster 中,这些 Cluster 用 Dirty queue 进行组织,同一个集合中修改页的物理位置比较接近. 和 FAB 算法的块层 LRU 算法相比,CFDC 中的块大小是可变的.

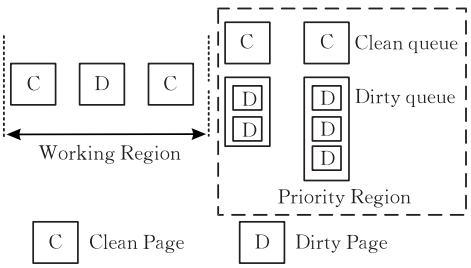

图 3　CFDC 置换策略示意图

3　FClock 算法

3.1　数据结构

如图 4 所示,FClock 将缓冲区中的数据页根据其读写状态组织为两个环形数据结构 CC 和 DC,分别维护只读页和修改页. 假定缓冲区可以存放 s 个数据页,则 $|CC \cup DC| = s \wedge CC \cap DC = \varnothing$. FClock 维护了一个全局计数器 $Counter$,每当发生一次数据页请求,$Counter$ 值加 1. 对于缓冲区中的每个数据页 p,FClock 为其关联 3 个变量:T、C 和 I,其中 T 表示 p 进入缓冲区的时间(当时的 $Counter$ 值),否则为最近一次被命中的时间);C 是 p 的访问位计数器,表示 p 被访问的频繁程度;I 表示 p 最近两次被命中之间对其它数据页访问的次数,称为"命中距离". 例如,假设数据页的请求序列为"r_1, r_2, r_3, r_4, r_1",则在访问完第一个 r_1 之后,$Counter=1$,因此 $r_1.T=1$,第二次访问完 r_1 后,$r_1.I=Counter-r_1.T-1=3$,表示最近两次命中 r_1 之间对其它数据页进行了 3 次访问,$r_1.T=Counter=5$. 另外 FClock 还为 CC 和 DC 各维护一个变量 F,用于表示 CC 和 DC 从最近一次命中数据页后到目前为止发生页缺失的次数. 图 4 中的虚线环用于处理循环和序列模式的数据页访问,称为子环,CC 和 DC 中的子环分别用 SC_{CC} 和 SC_{DC} 表示. 本文所用符号的意义如表 1 所示.

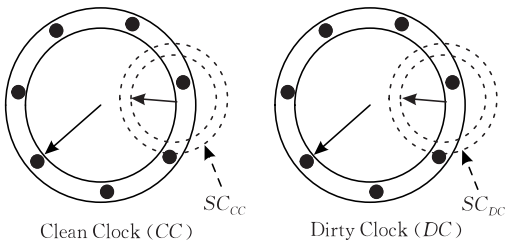

图 4　FClock 置换策略示意图

表 1　本文所用符号及其意义说明

符号名称	意义说明
CC	维护只读页的 clock(Clean Clock)
DC	维护修改页的 clock(Dirty Clock)
s	缓冲区可容纳的数据页个数
SC_{CC}	CC 中的子环
SC_{DC}	DC 中的子环
ζ	过去 s 次访问的命中页的平均访问间隔
C_r	从 SSD 读入一个数据页的代价
C_w	向 SSD 写入一个数据页的代价
M_{CC}	过去一段时间内(s 个数据页访问)CC 中数据页的物理操作次数
M_{DC}	过去一段时间内(s 个数据页访问)DC 中数据页的物理操作次数
R_{CC}	过去一段时间内(s 个数据页访问)CC 中数据页的逻辑操作次数
R_{DC}	过去一段时间内(s 个数据页访问)DC 中数据页的逻辑操作次数
$Counter$	全局计数器

3.2　基于代价的置换页选择策略

如果缓冲区满且当前请求的数据页 p 在缓冲区中,则可直接从缓冲区中访问此页;反之,FClock 置换策略将按照"代价"从 CC 或 DC 中选择一个页 x 进行置换,并从 SSD 读入数据页 p. FClock 的基本思想是:CC 和 DC 的大小应该和其在过去一段时间内由于数据页缺失所付出的代价成比例(式(1)). 假设缓冲区最多可放 s 个数据页,FClock 的置换策略可表述为:若 $|CC|<\beta s$,则 DC 过大,那么选择 DC 中指针所指的计数为 0 的数据页进行置换;反之从 CC 中选择计数为 0 的数据页进行置换. 本文中过去一段时间指过去 s 次访问,CC 的代价记为 C_{CC},DC 的代价记为 C_{DC}. 式(1)表示 CC 的大小占总缓冲区的比例.

$$\beta = C_{CC}/(C_{CC}+C_{DC}) \tag{1}$$

当数据页驻留缓冲区时,在连续两个物理操作之间可能发生多次逻辑操作. 尽管物理操作体现了

CC 和 DC 实际付出的代价，访问序列却是以逻辑操作的方式呈现的. 虽然逻辑操作和物理操作不同，但对于系统的性能来说都有影响. 一方面，物理操作对存取模式变化本身的反应比较迟钝；另一方面，虽然考虑逻辑操作可以快速侦测存取模式的变化，但对于存取模式剧烈变化的情况不太适用. 可见，单纯使用任何一种操作计算代价都不够全面.

为了最小化物理操作的代价，受文献[17]的启发，本文提出一种基于时钟数据结构并结合物理操作和逻辑操作优点的代价计算方案. 假定对数据页的存取是相互独立的，缓冲区可以存放 s 个数据页，n 是被处理文件中数据页的个数，则对某个数据页的逻辑操作在缓冲区中命中的概率是 s/n，而一个逻辑操作被转换为物理操作的概率是 $(1-s/n)$. 如式(2)和式(3)所示，本文提出的代价计算方案同时考虑了逻辑操作和物理操作.

$$C_{CC}=(R_{CC}\cdot(1-s/n)+M_{CC})\cdot C_r \tag{2}$$

$$C_{DC}=(R_{DC}\cdot(1-s/n)+M_{DC})\cdot(C_w+C_r) \tag{3}$$

当选择一个置换页时，通过考虑逻辑操作的影响，FClock 可以较快识别存取模式的变化并进行相应的调整. 而且，由于物理操作的影响也考虑在内，FClock 可以适应存取模式剧烈变化的情况.

如前所述，代价的计算基于过去一段时间内(s 个数据页访问)的统计结果. 本文通过使用一个最大长度为 s 的队列来记录过去一段时间数据页的访问情况，队列中每个元素 e 对应了一次数据页的访问，e 由两个数据成员组成：nC 和 $bHit$，其中 nC 的取值可以是 CC 或者 DC，表示相应的数据页请求发生在 CC 或者 DC 中，$bHit$ 表示对数据页的访问请求是否命中，以此来区别逻辑操作和物理操作. 基于该队列，可以在每次发生数据页请求时，在 $O(1)$ 代价的基础上更新 CC 和 DC 的代价值. 由于队列长度有限，因此维护该队列所需的内存非常有限. 代价的计算分为 3 步：(1) 去除头元素 e_h，(2) 加入新元素 e_t，(3) 根据以上介绍的方案计算代价 C_{CC} 和 C_{DC}. 其中第(1)步去除队头元素后，和第(2)步加入新元素后需要根据去除的元素和加入的新元素修改当前 CC 和 DC 对应的逻辑操作和物理操作的次数.

3.3 数据页访问位修改策略

虚存、数据库以及文件系统中经常会出现对同一个数据页多次连续访问后不再访问或者间隔较长时间后再访问的问题. 假设频繁访问的数据页之间的间隔大于两倍缓冲区大小. 原始的 clock 算法(即二次机会算法的改进)为每个数据页关联一个访问位，数据页初次进入缓冲区时，访问位置 0，当某个数据页在缓冲区中命中时，访问位置 1，当时钟指针扫过该页且其访问位为 1 时，访问位置 0. 显然这种方法会每隔一段时间(大于两倍缓冲区大小)将频繁访问的数据页置换出缓冲区. 而改进的 clock 算法要么在每次访问后都让访问位加 1，要么在加 1 的基础上为访问位设定一个最大值，都会导致短时间内频繁访问但以后不再访问的数据页长时间驻留内存. 因此，对于短时间内频繁访问的问题，本文提出用平均命中距离来解决这一问题.

定义 1(平均命中距离 ζ). 对于过去一段时间的访问序列“$r_1,r_2,\cdots,r_n$”而言，平均命中距离指该序列中所有被命中的数据页的命中距离的平均值，计算方法见式(4)，其中 m 指过去 n 次访问数据页时命中的次数.

$$\zeta=\frac{\sum_{i=1}^{m}r_i.I}{m} \tag{4}$$

例如，对于访问序列“$r_1,r_2,r_2,r_3,r_2,r_5,r_1,r_2$”而言，假设缓冲区大小为 8，初始状态为空. 当访问最后一个 r_2 时，可知在过去 8 次访问中，r_1 在第 7 次访问时被命中，其命中距离 $r_1.I=5$，而 r_2 在第 3、第 5 及第 8 次访问时被命中，其命中距离分别为 0、1 和 2. 由式(4)可知该序列过去 8 次访问的平均命中距离是 $(5+0+1+2)/4=2$.

FClock 使用平均命中距离来衡量某次命中是否为短时间内的频繁访问，其访问位修改策略可表述为：如果某个数据页的命中距离不小于平均命中距离 ζ，则该数据页的访问计数加 1，否则保持不变. 需要注意的是，虽然从表面上看该策略可能造成一直频繁访问的数据页的访问位永远不能增加的问题，实际上，如图 5 和算法 1 所示的过程，Hash 表中维护了数据页的有效访问位置，通过该变量，可知频繁访问数据页的访问位的值会得到慢慢增加. 这一点在算法 1 后面的例子中进行具体的说明.

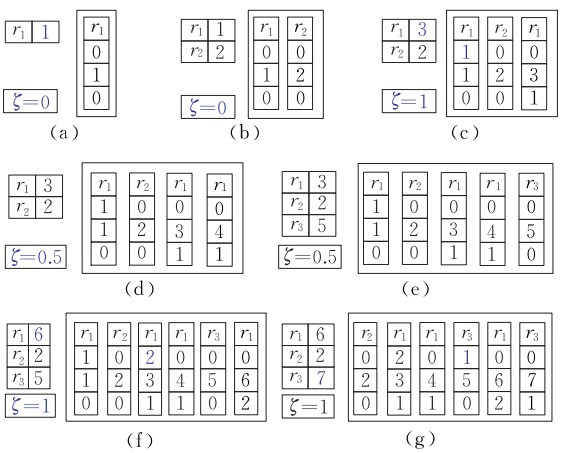

图 5 计算平均命中距离示意图

由于平均命中距离是基于过去一段时间的访问序列来进行计算的,每当执行一次新的数据页访问时,如果按照式(4)进行计算的话,显然时间复杂度太高.为此,本文提出一种在常量时间内计算平均命中距离的方法.

如图 6 所示,计算平均命中距离时使用两种数据结构,分别是队列和 Hash 表.队列用于维护过去一段时间内数据页的访问情况,本文实验中,过去一段时间指过去 s 次数据页的访问,s 是缓冲区中最多容纳的数据页个数,队列中元素的成员 I、T、C 的意义在 3.1 节进行了阐述.Hash 表用于记录在过去 s 次访问中队列中的数据页在队列中的有效位置,该有效位置用于计算每个数据页的命中距离以及整个缓冲区的平均命中距离 ζ.限于篇幅,这里用例子来说明如何在常量时间内计算平均命中距离 ζ.

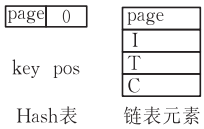

图 6　计算平均命中距离的数据结构

例如,对于访问序列"$r_1, r_2, r_1, r_1, r_3, r_1, r_3$",假设 $s=6$,初始情况下队列和 Hash 表均为空.为了说明的方便,Hash 表中数据页在队列中的位置用数据页的 T 值表示.该序列的处理过程如下:

(1)访问 r_1(图 5(a)),r_1 的 I、T 的值分别是 0、1,$\zeta=0$,然后将 r_1 加入 Hash 表中;

(2)访问 r_2(图 5(b)),r_2 的 I、T 值分别是 0、2、0,$\zeta=0$,然后将 r_2 加入 Hash 表中;

(3)访问 r_1(图 5(c)),r_1 的 I、T 值分别是 0、3,由于 r_1 命中,通过 r_1 在 Hash 表中找其在队列中的有效位置 1,得到有效位置处的 T 值 1,进而可以得到 r_1 的命中距离 1.由于 $1>\zeta=0$,将 Hash 表中 r_1 的有效位置对应的 I 值更新为 1,同时更新 Hash 表中 r_1 的有效位置更新为 3,最后计算平均命中距离 $\zeta=1$;

(4)访问 r_1(图 5(d)),r_1 的 I、T 值分别是 0、4,由于 r_1 命中,通过 r_1 在 Hash 表中找其在队列中的有效位置 3,得到有效位置处的 T 值 3,进而可以得到 r_1 的命中距离 0.由于 $0<\zeta=1$,则 Hash 表中 r_1 的有效位置及其对应的 I 值不变,最后计算平均命中距离为 $\zeta=0.5$;

(5)访问 r_3(图 5(e)),r_3 的 I、T 值分别是 0、5,由于 r_3 没有命中,将 r_3 及其有效位置 5 加入 Hash 表中,ζ 保持不变;

(6)访问 r_1(图 5(f)),r_1 的 I、T 值分别是 0、6.由于 r_1 命中,通过 r_1 在 Hash 表中找其在队列中的有效位置 3,得到有效位置处的 T 值 3,进而可以得到 r_1 的命中距离 2.由于 $2>\zeta=0.5$,则 Hash 表中 r_1 的有效位置 3 对应的 I 值变为 2,同时将 Hash 表中 r_1 的有效位置更新为 6,最后计算平均命中距离为 $\zeta=1$;

(7)访问 r_3(图 5(g)),r_3 的 I、T 值分别是 0、7.由于 r_3 命中,通过 r_3 在 Hash 表中找其在队列中的有效位置 5,得到有效位置处的 T 值 5,进而可以得到 r_3 的命中距离 1.由于 $1\geq\zeta=1$,则 Hash 表中 r_3 的有效位置 5 对应的 I 值变为 1,同时将 Hash 表中 r_3 的有效位置更新为 7,最后计算平均命中距离为 $\zeta=1$.

计算平均命中距离的方法在 FClock 算法中用 update(ζ)来表示.

3.4　FClock 算法

FClock 算法的具体流程如算法 1 所示.在缓冲区未满的初始阶段,即 $|CC\cup DC|<s$,如果请求页 p 没有命中(算法 1 中的 Case II),则根据对 p 的读(8～10 行)或者写(12～14 行)操作类型从 SSD 上读取到 CC 或者 DC 中,最后在第 15 行更新平均命中距离的值.如果 p 命中,即 $p\in CC\cup DC$,对应算法 1 的 Case I,如果 $p\in CC$(第 1 行),则 CC 的逻辑操作次数加 1,然后将其命中距离置 0;否则如果 $p\in DC$(第 2 行),则 DC 的逻辑操作次数加 1,然后将其命中距离置 0.如果操作类型是 write 并且 $p\in CC$(第 3 行),则将 p 从 CC 中移到 DC 中.在第 4 行,如果 p 的命中距离不小于平均命中距离,则将 p 的访问计数加 1.最后在第 5 行更新平均命中距离的值.

算法 1.　FClock(page p, type T).

```
/*FClock 在每次系统请求数据页 p 时被触发,T 表示
  对 p 的操作类型,可以是 read 或者 write */
Case I: p∈CC∪DC                    /*p 被命中*/
1. if (p∈CC) then {R_CC ← R_CC+1; CC.F ← 0;}
2. else {R_DC ← R_DC+1; DC.F ← 0;}
3. if (T=write and p∈CC) then Move p to DC;
4. if (p.I≥ζ) then {p.C ← p.C+1; }
5. update(ζ);
Case II: p∉CC∪DC                   /*p 没有命中*/
6. if (|CC∪DC|=s ) then evictPage();
7. if (T=read) then
8.      R_CC ← R_CC+1; M_CC ← M_CC+1;
9.      fetch p from the disk;
10.     Insert (p,CC);
11. else
12.     R_DC ← R_DC+1;
13.     fetch p from the disk;
14.     Insert (p,DC);
```

```
15. update(ζ);
过程 1. evictPage()
1. β←C_CC/(C_CC+C_DC)
Case I. |CC| <βs /*DC 过大,从 DC 中移除数据页*/
2. M_DC← M_DC+1;
3. Let q be the page pointed by clock hand of DC (or SC_DC);
4. while (q.C>0) do {q.C←q.C−1; q← q−>next;}
5. write q's content to SSD; delete q;
Case II. |CC|≥βs /*CC 过大,从 CC 中移除数据页*/
6. Let q be the page pointed by clock hand of CC (or SC_CC);
7. while (q.C >0) do {q.C←q.C−1; q←q−>next;}
8. delete q;
过程 2. Insert(p,clockx).
        /*p 为数据页,clockx 可以是 CC 或者 DC */
1. if (clockx.F≥λ|clockx|) then
        /*λ 是调节因子,取值在[0,1]区间*/
2.    add p to SC_clockx;
3. else add p to clockx;
4. clockx.F←clockx.F+1.
```

如果缓冲区已满,当数据页命中时,其操作和前面一段所介绍的内容相同. 如果数据页没有命中,则FClock 会在算法 1 的第 6 行首先调用 evictPage( ) 从 CC 或者 DC 中选择一个数据页进行置换. evictPage 的具体操作见过程 1,其基本思想和计算方法已在 3.2 节中进行了说明. 然后在第 7～14 行根据对 p 的操作类型从 SSD 读入 p 并将其放入 CC 或者 DC 中,最后在第 15 行更新平均命中距离的值.

在 FClock 中,将数据页插入 CC(DC)时,调用了 Insert( )过程,在第 1 行会判断 CC(DC)的 F 值(自从最近一次命中后发生的连续页缺失次数),如果 $F>\lambda|clockx|$,则 FClock 认为目前的数据页存取模式为序列存取,同时构造 CC(DC)的子环 SC_{CC}(SC_{DC}),并将 p 放入 SC_{CC}(SC_{DC})中;否则直接在第 3 行将 p 放入 CC(DC)中. 注意子环 SC_{CC}(SC_{DC})是 CC(DC)的一部分. 在第 4 行,将 CC(DC)的缺页数 F 加 1. 过程 Insert 中 λ 是调节因子,取值在[0,1]区间.

例如,对于访问序列"$r_1, r_2, r_1, r_1, r_3, r_1, r_3$"而言,其过程如下:

(1)访问 r_1,没命中. 从 SSD 上读入 r_1 并按照其操作类型放入 CC 或者 DC 中,同时在算法 1 的第 15 行使用 3.3 节介绍的方法更新 $\zeta=0$.

(2)访问 r_2,没命中. 处理过程同(1).

(3)访问 r_1,命中. 则在算法 1 的第 4 行根据 $r_1.I=1>\zeta=0$,则 $r_1.C$ 加 1,如图 5(c)所示,然后更新 ζ 的值为 1.

(4)访问 r_1,命中. 由于 $r_1.I=0<\zeta=1$,则 $r_1.C$ 保持不变,如图 5(d)所示,然后更新 ζ 的值为 0.5.

(5)访问 r_3,没命中. 和 r_2 的处理相同,如图 5(e)所示,ζ 保持不变,$r_3.C=0$.

(6)访问 r_1,命中. 注意这时 $r_1.I$ 的计算依赖于图 5(e)中的 Hash 表,可知 r_1 的有效位置为 3,进而可知 $r_1.I=2>\zeta=0.5$,因此 $r_1.C=r_1.C+1=2$,如图 5(f)所示,然后在算法 1 第 5 行更新 $\zeta=1$.

(7)访问 r_3,命中. 由于 $r_3.I=1\geq\zeta=1$,因此 $r_3.C$ 加 1,如图 5(g)所示,随后在算法 1 第 5 行更新 $\zeta=1$.

3.5 分 析

FClock 的自适应性体现在两方面:(1)基于代价的置换策略,当需要选择一个置换页时,FClock 从 CC 或者 DC 中根据各自的累加代价和公平地选择合适的置换页. 当读操作较多时,CC 会慢慢变大,相反,DC 会慢慢变大. 因此 FClock 能很好地处理同一闪存读写的不对称性以及不同闪存读写不对称性的巨大差异性,可以应用到不同类型的 SSD 上;另外,由于 FClock 在计算代价时同时考虑了物理操作和逻辑操作,FClock 可以适应不同的存取模式. (2)FClock 使用平均命中距离来控制数据页的引用计数,可以使得频繁访问的数据页的引用计数的值慢慢而不是快速增加,可以避免二次机会算法快速换出间隔较长时间后频繁访问的数据页被过早换出的问题,同时可以避免每次命中就加 1 的改进 CLOCK 算法所造成的无用数据页长时间驻留内存的问题. 和 CFLRU 及 CFDC 相比,FClock 考虑了引用计数,并且可以避免 LRU、CFLRU 及 CFDC 存在的锁争用问题.

FClock 可以很好地处理序列存取模式. 当需要处理序列引用时,FClock 使用 F 来检测 CC(DC)的页缺失次数,当达到一定程度时,即可认为出现了序列存取模式,这时,FClock 通过构造 CC(DC)的子环 SC_{CC}(SC_{DC})来处理新来数据页的插入和移除操作,从而不会对子环以外的数据页产生影响. 相比之下,CFLRU 及 CFDC 没有考虑存取模式的影响,这一点在第 4 节的实验结果部分也得到了证明.

当循环请求序列涉及的操作类型既包含读操作,也包含写操作时,FClock 可以很好地处理循环存取模式. 原因在于闪存读写代价不对称,而 FClock 根据代价而不是存取的先后顺序选择置换页,因此对于长循环而言,FClock 将打乱循环存取模式的置换顺序. 相比之下,CFLRU 和 CFDC 由于首先置换只读页,因此可以一定程度上打乱循环存取模式的置换顺序,但 FClock 的自适应性使得这种

打乱存取模式的行为具有自适应性,可以根据不同闪存的读写特点进行调整,而 CFLRU 和 CFDC 不具备这一特点,从而导致其性能下降,这一点在第 4 节的实验结果中也得到了进一步证明.

4　实　验

4.1　实验环境

本文的实验目的是验证 FClock 算法针对不同读写代价的 SSD 的有效性.我们选择两种 SSD 进行实验:(1)三星 MCAQE32G5APP,简便起见,用 FD1 表示;(2)三星 MCAQE32G8APP-0XA,用 FD2 表示.FD1 和 FD2 的随机读写的比率分别是1∶118 和 1∶2.这两个闪存硬盘的读写性能存在巨大的差异,这是因为 FD1 是由 MLC 类型的 NAND 芯片构成,而 FD2 是由 SLC 类型的 NAND 芯片构成.

对 SSD 来说,缓冲区置换算法的性能受物理读写次数的影响,然而 FTL 层的实现是设备相关的,由硬盘制造商提供,并没有为用户提供跟踪读写次数的接口.因此,我们选择使用模拟器[18]来进行测试.我们实现了 5 种置换策略来进行比较,即 LRU、CLOCK[2]、CFLRU[10]、CFDC[14] 及本文提出的 FClock.所有的置换策略都用 Visual C++实现的.我们将 CFLRU 算法中"置换区"的"窗口大小"设为缓冲区大小的 75%,将 CFDC 的"置换区"的"窗口大小"设为缓冲区的 50%,将 CFDC 的"聚类大小"设为 64.参数取自对应文献实验中所采用的数值.

我们将数据库的文件大小模拟为 64MB,相当于 32000 个的物理页,每页为 2KB.缓冲区的大小范围从 2000 个页到 8000 个页.本文实验中,模拟器假定数据页的大小是 2KB,每个数据块包含 64 个数据页.

我们生成了 4 种类型的测试数据,其统计数据如表 2 所示,其中"读/写比率"列中的"$x\%/y\%$"表示对某种测试数据来说,所有请求的 $x\%$为读操作、$y\%$为写操作;"局部性"列中的"$x\%/y\%$"表示对某种测试数据来说,在 $y\%$的页上有 $x\%$的操作.

表 2　实验所用测试数据的统计信息

编号	总的请求	读/写比例	局部性
T1	3000000	90%/10%	60%/40%
T2	3000000	80%/20%	50%/50%
T3	3000000	60%/40%	60%/40%
T4	3000000	80%/20%	80%/20%

表 1 中读写代价 C_r和 C_w可以通过 SSD 的技术手册得到,或者通过执行一定量的读写操作后取平均值来获得.本文实验所用数据来自于技术手册.

我们选择以下标准来评价缓冲区置换策略:(1)物理读操作的次数,(2)物理写操作的次数,(3)运行时间.其中运行时间是通过将读操作和写操作次数之和相加得到的.

4.2　性能比较和分析

4.2.1　读写操作代价差异巨大的 SSD 上性能比较

读操作性能比较. 图 7 展示了 4 种已有方法和基于本文提出的基于代价的 FClock 方法在 FD1 上运行 T1、T2、T3 及 T4 时随机读次数比较.可以看出,和 LRU 及 CLOCK 相比,基于闪存的算法

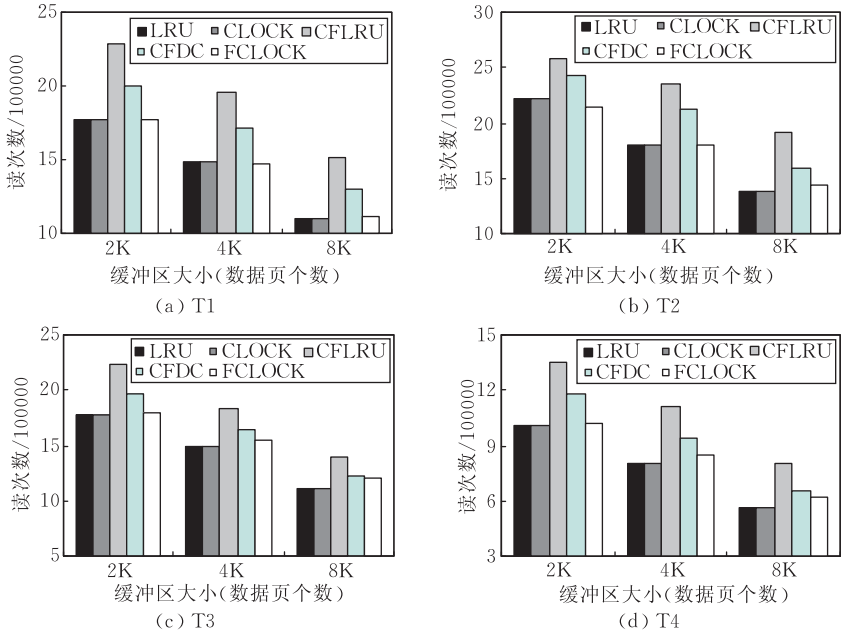

图 7　不同方法在 FD1 上运行 T1～T4 时读次数比较

(CFLRU、CFDC、FClock)需要更多的读次数,但本文提出的方法 FClock 所需的读次数在 T1 到 T4 上远少于 CFLRU 和 CFDC.可见,不考虑只读页的操作频率就直接进行置换导致 CFLRU 和 CFDC 需要付出很多不必要的物理读操作.

写操作性能比较. 图 8 展示了 4 种已有方法和基于本文提出的基于代价的 FClock 方法在 FD1 上运行 T1、T2、T3 及 T4 时随机写次数比较.可以看出,基于闪存的算法(CFLRU、CFDC、FClock)涉及的写操作的次数远少于基于磁盘的 LRU 和 CLOCK 算法.同时,尽管 CFLRU 首先置换只读页,本文提出的方法 FClock 依然好于 CFLRU,原因是进行置换时,由于 FD1 的读写比例差异巨大,本文方法将在缓冲区中保留更多的修改页.

运行时间比较. 图 9 展示了不同方法在 FD1 上运行 T1、T2、T3 及 T4 时运行时间的比较.可以看出,基于闪存的算法(CFLRU、CFDC、FClock)所需的运行时间远少于 LRU 和 CLOCK 算法.这是因为

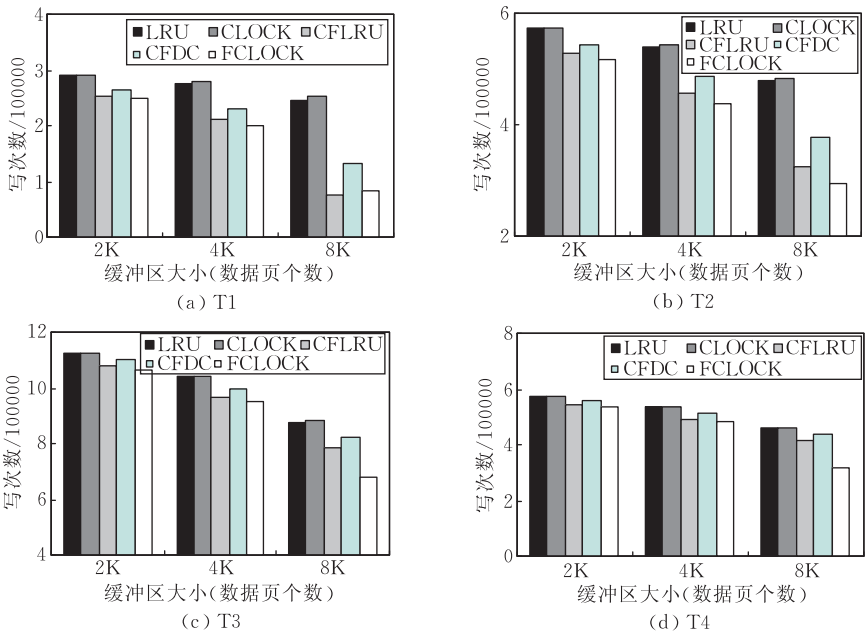

图 8 不同方法在 FD1 上运行 T1～T4 时写次数比较

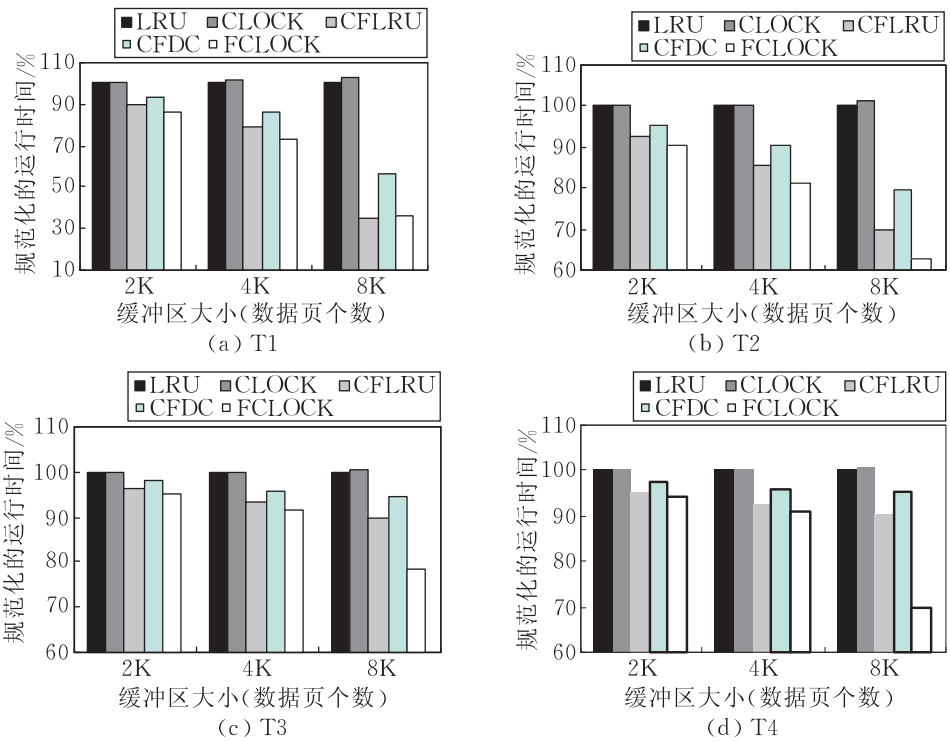

图 9 不同方法在 FD1 上运行 T1～T4 时运行时间比较

对于 FD1 来说,读写代价相差 118 倍,而且对基于闪存的置换算法来说,CFLRU 和 CFDC 优先置换只读页,FClock 算法给予写操作更高的权重,因此会大量减少写操作的次数,最终导致整体性能提升.

4.2.2　读写操作代价差异小的 SSD 上的性能比较

读操作性能比较. 图 10 展示了 4 种已有方法和基于本文提出的基于代价的 FClock 方法在 FD2 上运行 T1、T2、T3 及 T4 时随机读次数比较. 可以看出,和 LRU 及 CLOCK 相比,CFLRU 和 CFDC 由于优先置换只读页,因此需要更多的物理读操作. FClock 既考虑了不同读写状态的数据页操作代价,同时也能更好地处理数据页的存取模式,整体而言,虽然也给予写操作较大的权重,但依然能达到和 LRU 及 CLOCK 算法类似的读操作次数.

写操作性能比较. 图 11 展示了不同方法在 FD2 上运行 T1、T2、T3 及 T4 时随机写次数比较. 可以看出,由于 CFLRU 和 CFDC 无条件优先置换只读页,因此所需写操作的次数最少. 而本文提出的方法综合考虑存取模式及物理操作代价,因此在读写操作代价相差不大的情况下,与 CFLRU 和 CFDC 相比,将更多考虑读操作的权重,因此写操作的次数明显多于 CFLRU 和 CFDC,但仍然少于 LRU 和 CLOCK 算法.

运行时间比较. 图 12 展示了不同方法在 FD2

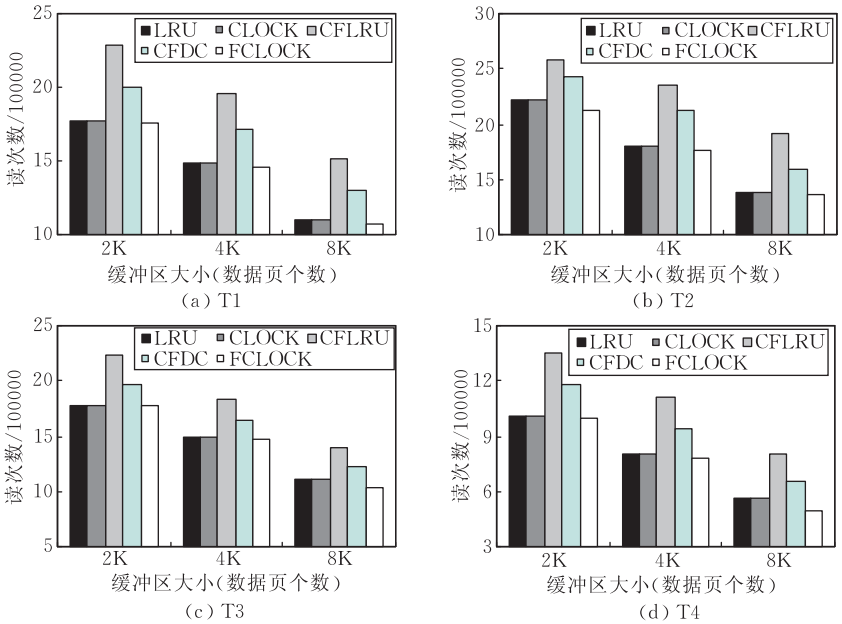

图 10　不同方法在 FD2 上运行 T1～T4 时读次数比较

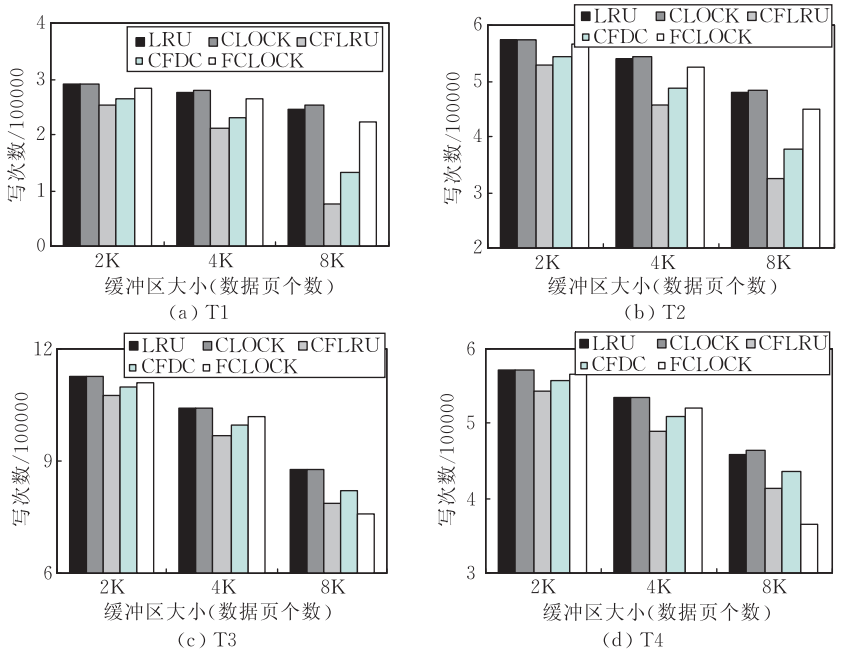

图 11　不同方法在 FD2 上运行 T1～T4 时写次数比较

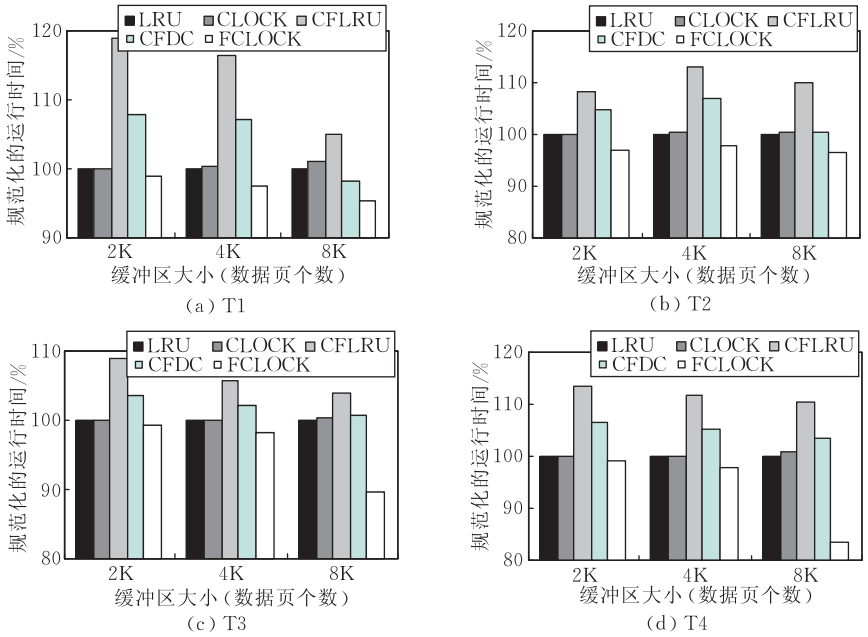

图 12 不同方法在 FD2 上运行 T1～T4 时运行时间比较

上运行 T1 及 T2 后的运行时间比较.可以看出,由于 FD2 读写操作的代价相差不大,而 CFLRU 和 CFDC 的读操作次数远远多于其他方法,因此二者所需的总运行时间远多于其他方法.而本文提出方法的读次数和 LRU 及 CLOCK 差不多,且写次数比 LRU 及 CLOCK 少,因此整体性能最好.

4.2.3 不同 SSD 硬盘的性能比较

通过比较图 7～图 12 可以看出,对于读写操作代价差异巨大的 SSD,如 FD1、LRU 和 CLOCK 的整体性能不如基于闪存的置换算法,但对于读写操作代价差异不大的 SSD,如 FD2、LRU 和 CLOCK 的整体性能要好于 CFLRU 及 CFDC,而本文提出的 FClock 在选择置换页时,根据操作的代价进行操作,可以在只读页和修改页之间进行公平的选择,因此可以适用于不同读写比例的 SSD.

5 结论和展望

针对现有基于闪存的缓冲区管理算法没有考虑不同闪存读写代价不对称性之间的巨大差异性问题以及 LRU 算法存在锁争用问题,本文提出一种基于闪存硬盘(SSD)的自适应缓冲区管理算法 FClock,FClock 将缓冲区中的数据页组织为只读环和修改环,使用基于代价的启发式来选择置换页,可在未修改的数据页和已修改的数据页之间进行公平的选择,适用于不同种类的 SSD 及存取模式.针对数据库、虚存和文件系统中数据页访问存在高相关性的特点,提出基于"平均命中距离"的访问计数方法来调整数据页的访问频率.基于不同 SSD 和不同存取模式的实验结果说明,FClock 的综合性能优于已有方法.

参 考 文 献

[1] Hwang C-G. Nanotechnology enables a new memory growth model. Proceedings of the IEEE, 2003, 91(11): 1765-1771

[2] Babaoglu O, Joy W. Converting a swap-based system to do paging in an architecture lacking page-reference bits. ACM SIGOPS Operating Systems Review, 1981, 15(5): 78-86

[3] Robinson J T, Devarakonda M V. Data cache management using frequency-based replacement//Proceedings of the ACM SIGMETRICS. Boulder, Colorado, USA, 1990: 134-142

[4] O'Neil E J, O'Neil P E, Weikum G. The LRU-K page replacement algorithm for database disk buffering//Proceedings of the ACM SIGMOD Conference. Washington, 1993: 297-306

[5] Johnson T, Shasha D. 2Q: A low overhead high performance buffer management replacement algorithm//Proceedings of the 20th International Conference on Very Large Data Base. Santiago de Chile, Chile, 1994: 439-450

[6] Jiang S, Zhang X. Making LRU friendly to weak locality workloads: A novel replacement algorithm to improve buffer cache performance. IEEE Transactions on Computers, 2005, 54(8): 939-952

[7] Megiddo N, Modha D S. ARC: A self-tuning, low overhead replacement cache//Proceedings of the FAST'03 Conference on File and Storage Technologies. San Francisco, California, USA, 2003: 115-130

[8] Lee D, Choi J, Kim J-H, Noh S H, Min S L, Cho Y, Kim C-S. LRFU: A spectrum of policies that subsumes the least

recently used and least frequently used policies. IEEE Transactions on Computers, 2001, 50(12): 1352-1361

[9] Effelsberg W, Haerder T. Principles of database buffer management. ACM Transactions on Database Systems, 1984, 9(4): 560-595

[10] Park S-Y, Jung D, Kang J-U, Kim J, Lee J. CFLRU: A replacement algorithm for flash memory//Proceedings of the 2006 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES). Seoul, Korea, 2006: 234-241

[11] Jo H, Kang J-U, Park S-Y, Kim J-S, Lee J. FAB: Flash-aware buffer management policy for portable media players. IEEE Transactions on Consumer Electronics, 2006, 52(2): 485-493

[12] Kim H, Ahn S. BPLRU: A buffer management scheme for improving random writes in flash storage//Proceedings of the 6th USENIX Conference on File and Storage Technologies. San Jose, CA, USA, 2008: 239-252

[13] Koltsidas I, Viglas S. Flashing up the storage layer//Proceedings of the VLDB Endowment. Auckland, New Zealand, 2008, 1(1): 514-525

[14] Ou Y, Haerder T, Jin P. CFDC: A flash-aware replacement policy for database buffer management//Proceedings of the 5th International Workshop on Data Management on New Hardware (DaMoN'09). Providence, Rhode Island, USA, 2009: 15-20

[15] Chrobak M, Karloff H J, Payne T H, Vishwanathan S. New results on server problems. SIAM Journal on Discrete Mathematics, 1991, 4(2): 172-181

[16] Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. USA: The MIT Press, 2001

[17] Tang X, Meng X F. ACR: An adaptive cost-aware buffer replacement algorithm for flash storage divices//Proceedings of the 2010 Eleventh International Conference on Mobile Data Management (MDM). Kansas City, Missouri, USA, 2010: 33-42

[18] Jin P, Su X, Li Z. A flexible simulation environment for flashaware algorithms//Proceedings of the ACM Conference on Information and Knowledge Management. Hong Kong, China, 2009: 2093-2094

TANG Xian, born in 1978, Ph. D. candidate. Her research interests focus on flash database system.

MENG Xiao-Feng, born in 1964, professor, Ph. D. supervisor. His research interests include Web data management, Cloud data management, mobile data management, XML data management, Flash-aware DBMS, privacy protection.

Background

Flash disks are being widely used as an important alternative to conventional magnetic disks. Although accessed through the same interface by applications, their distinguished feature, i. e. different read and write cost in the aspects of time, makes it necessary to reconsider the design of existing replacement algorithms to leverage their performance potential.

Different from existing flash-aware buffer replacement policies that focus on the asymmetry of read and write operations, we address the "discrepancy" of the asymmetry for different flash disks, which is the fact that exists for a long time, while has drawn little attention by researchers since most existing flash-aware buffer replacement polices are somewhat based on the assumption that the cost of read operation is neglectable compared with that of write operation. In fact, this is not true for current flash disks on the market.

This paper proposes an adaptive replacement policy (FClock) which has two ring-shaped data structures, i. e. *CC* (their content remain unchanged) and *DC* (their content is modified), to manage clean pages and dirty pages in the buffer, respectively. When selecting a victim page, FClock uses cost-based heuristics to fairly make trade off between clean pages and dirty pages, and hence, can work well for different type of flash disks of large discrepancy. Further, for the problem of "correlated references" to database, virtual memory and file systems, this paper proposes a reference counter based on "average hit distance" to control the reference frequency. The experimental results on different traces and flash disks show that FClock not only adaptively tunes itself to workloads of different access patterns, but also works well for different kind of flash disks compared with existing methods.

This research was partially supported by the grants from the National Natural Science Foundation of China (No. 60833005); the National High Technology Research and Development Program (863 Program) of China (No. 2009AA011904); and the Doctoral Fund of Ministry of Education of China (No. 200800020002). This project aims at constructing the fundamental theory and design principles of flash-based database including a series of key problems such as system architecture, storage management and indexing, query processing, transaction processing, buffer management, etc. The work introduced in this paper belongs to buffer management and is very important for this project to construct flash-based databases.