软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(12):2951−2964 [doi: 10.3724/SP.J.1001.2011.03967]

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

基于代价的闪存数据库缓冲区置换算法*

汤 显[1,2+], 孟小峰[1], 梁智超[1], 卢泽萍[1]

[1](中国人民大学 信息学院,北京 100872)

[2](燕山大学 经济管理学院,河北 秦皇岛 066004)

Cost-Based Buffer Management Algorithm for Flash Database Systems

TANG Xian[1,2+], MENG Xiao-Feng[1], LIANG Zhi-Chao[1], LU Ze-Ping[1]

[1](School of Information, Renmin University of China, Beijing 100872, China)

[2](School of Economics and Management, Yanshan University, Qinhuangdao, Hebei 066004)

+ Corresponding author: E-mail: txianz@gmail.com

Tang X, Meng XF, Liang ZC, Lu ZP. Cost-Based buffer management algorithm for flash database systems. *Journal of Software*, 2011,22(12):2951−2964. http://www.jos.org.cn/1000-9825/3967.htm

Abstract: Different from existing flash-aware buffer replacement policies that focus on the asymmetry of read and write operations, this paper addresses the "discrepancy" of the asymmetry for different flash disks which has existed for a long time. The study proposes an adaptive replacement policy (CBLRU), which assigns to each page a weighted value that combines the IO cost and the influence of pages staying in the buffer. When selecting a victim page, the one with the minimum weighted value will be selected as the victim page, thus, one can avoid the problem of occupying the valid buffer space by dirty pages that are not used for a long time. Moreover, as the cost of read and write operations will be adjusted according to different flash disks, CBLRU can wisely tune itself and adapt to different kinds of flash disks. Finally, a theorem is proposed addressing the stability of the relationship between pages, based on which CBLRU organizes all data pages into two LRU lists: one for clean pages and the other for dirty pages, and then the CPU complexity is reduced form $O(k\log k)$ to $O(1)$. The experimental results show that compared with existing buffer-aware replacement algorithms, CBLRU is very efficient when being used for different kinds of flash disks.

Key words: flash; flash database; buffer management algorithm; cost

摘 要: 提出一种基于闪存硬盘(solid state disk,简称 SSD)的自适应缓冲区管理算法 CBLRU,其将数据页的置换代价与其驻留内存的影响相结合,为每个数据页附加一个权值,当发生页缺失问题时,选择具有最小权值的数据页进行置换,从而可以在延长修改页驻留缓冲区的同时,避免某些修改页长期占用缓冲区中有效空间问题的发生.由于该权值会根据不同闪存的读写代价进行动态调整,因此可适用于不同类型的闪存硬盘;进一步,提出了同类型数据页的权重关系稳定性结论,基于该结论,CBLRU 将缓冲区中的数据页组织为两个 LRU 队列,分别用于管理只读页和修改页,从而将内存的 CPU 操作代价从 $O(k\log k)$降低为 $O(1)$.基于不同闪存硬盘和不同存取模式的实验结果说

* 基金项目: 国家自然科学基金(60833005, 60573091); 国家高技术研究发展计划(863)(2007AA01Z155, 2009AA011904); 国家教育部博士点基金(200800020002)

收稿时间: 2009-09-03; 修改时间: 2010-05-05; 定稿时间: 2010-11-05

明,CBLRU 可有效应用于不同类型的闪存硬盘,且综合性能优于已有方法.

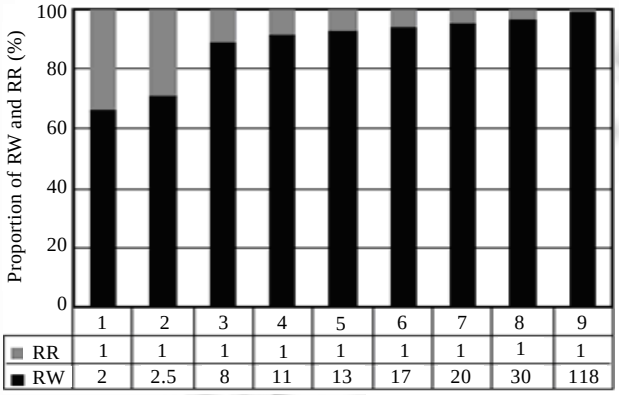
关键词: 闪存;闪存数据库;缓冲区置换算法;代价

中图法分类号: TP311 文献标识码: A

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.闪存的广泛应用,进一步促进了闪存制造工艺的发展.基于闪存的存储设备已经稳步进入个人计算机和企业服务器市场,并极有可能在企业数据库服务器中替代磁盘以获取高性能[1].目前,各种应用中都将闪存硬盘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,广泛应用于存储系统、数据库、网络服务器、文件系统以及操作系统.现有基于磁盘的缓冲区置换算法[2−8]假设读写操作的时间延迟相同,对于给定大小的缓冲区,其目标就是最小化缓冲区的缺页率.早在 20 年前,文献[9]已经意识到,缓冲区中数据页的读写状态是影响置换策略的重要因素,由于闪存的读写代价存在不对称性问题,在设计基于闪存的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,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以忽略不计,因此这些缓冲区置换策略都是通过无条件优先置换只读页来减少随机写操作的次数,从而提高系统性能.然而,如图 1 所示(*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 MCAQE32G5APP(http://www.datasheetcatalog.net)),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,读写代价的比例差异很大.当把已有方法应用于读写代价差异较小的闪存时,在不考虑只读页操作代价的情况下就无条件优先置换只读页是不合理的,甚至可能出现性能下降的问题,从而无法适用于不同种类的闪存硬盘.



	1	2	3	4	5	6	7	8	9
RR	1	1	1	1	1	1	1	1	1
RW	2	2.5	8	11	13	17	20	30	118

Fig.1 Normalized proportion of time consumed by random write (RW) and random read (RR) operations for NAND flash disks

图 1 NAND 闪存随机读(RR)和随机写(RW)消耗的时间比例

本文的研究以提升数据库系统的性能为目标,从缓冲区管理策略着手,针对闪存 I/O 的特点,提出一种面向不同种类闪存的基于代价的数据页置换算法——CBLRU.该方法通过为缓冲区中的每个数据页附加一个权值,

用来表示该页驻留内存所节省的代价.该权值综合考虑了具体的 I/O 代价及数据页因驻留内存所造成的影响.根据最小化代价的原则,在选择置换页时,CBLRU 选择权值最小的数据页作为置换页.

与已有的置换算法假设闪存随机读代价远远小于随机写代价不同,本文的研究仅假设闪存的读写代价存在差异.针对不同闪存读写代价存在巨大差异的问题,当选择置换页时,CBLRU 并非无条件优先置换只读页,而是通过权值的比较,在只读页和修改页之间进行公平的选择,从而避免已有方法带来的性能下降问题,可以适用于各种类型的闪存硬盘.

本文第 1 节简单回顾相关工作并对其进行分析.第 2 节介绍基于代价的 CBLRU 算法.第 3 节通过实验验证本文所提方法的有效性.最后总结全文.

1 相关工作

1.1 闪存存储器

闪存一般分为 NOR 型和 NAND 型两种.NOR 型闪存芯片与 EPROM 和 SRAM 一样,有专用的地址和数据总线,只能按字节和字读写,支持芯片内执行(execute in place,简称 XIP),应用程序可以直接在闪存内执行;NOR 型闪存的读速度很高,但是写入和擦除速度较低、容量小,一般用来存储代码和少量的数据.NAND 闪存芯片无专用的地址和数据总线,用一个 I/O 接口来控制输入/输出.闪存硬盘(solid state disk,简称 SSD)中通常使用的是 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.而且,闪存芯片的每个块在经过了特定次数的写和擦除操作之后,性能可能会变得不稳定.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.

1.2 缓冲区管理算法及分析

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.

当系统需要操作一个数据页时,缓冲区算法需要按照如下步骤为系统准备所需的数据页:

(1) 检查请求页是否在当前缓冲区中;

(2) 若请求页在当前缓冲区中,则直接取得所需数据,结束;

(3) 否则,按照某种策略选择一个置换页;

(4) 若置换页为只读页,则直接读入请求页;否则,首先将置换页的内容写回外存,然后读入请求页.

其中,前两步都可通过 $O(1)$代价完成.因此,处理一个请求页的代价由第(3)步的 CPU 代价和第(4)步的外存操作代价组成.

对于第(3)步操作,虽然一般情况下 CPU 操作代价相对于外存操作来说较小,实际上,尤其是在出现频繁换页的情况下,如果第(3)步的操作代价太高的话,将会影响到系统性能.

对于第(4)步的外存 I/O 操作,假设:

(i) 缓冲区中无空闲空间放置新的请求页;

(ii) 数据页访问序列为 $r_1,r_2,\ldots,r_N$,其中,发生页缺失的序列是 $r'_1,r'_2,\ldots,r'_M$ ($M \leqslant N$);

(iii) 访问 $r_i(1\leqslant i\leqslant N)$的 I/O 代价为 C_{r_i}.

则处理整个数据页访问序列的外存 I/O 操作代价为

$$C_{Total}=\sum_{i=1}^{N}C_{r_i} \tag{1}$$

第(4)步操作的目标就是最小化访问 n 个数据页的IO总代价,当请求页 r 在缓冲区中时,可以直接得到.这时,访问外存的 I/O 代价为 C_r=0,因此,公式(1)可以简化为

$$C_{Total}=\sum_{j=1}^{M}C_{r'_j} \tag{2}$$

假设 M 次页缺失对应的 M 个置换页中,有 M'个数据页在缓冲区进行了修改,从缓冲区写回外存的代价用 C_W表示,从外存读入缓冲区的代价用 C_R表示.相应的,磁盘上的操作代价用 C_W^D 和 C_R^D 表示,而闪存上的操作代价用 C_W^F 和 C_R^F 表示.对于磁盘来说,已有方法[2−8]的基本假设是磁盘的读写代价相同,即 $C_W^D=C_R^D$,公式(2)变为公式(3),其中, C_W^D 和 C_R^D 统一用 C 表示:

$$C_{Total}^D=\sum_{j=1}^{M}C_{r'_j}=M'(C_R^D+C_W^D)+(M-M')C_R^D=M'C_W^D+MC_R^D=kMC(1\leqslant k\leqslant 2) \tag{3}$$

因此,对于磁盘来说,降低 n 个数据页的 I/O 操作代价 C_{Total}^D 可以通过减少页缺失次数 M,即降低缺页率(提高命中率)来获得.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]及 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.

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.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.由于闪存的读写代价不对称,因此公式(2)变为公式(4):

$$C_{Total}^F=\sum_{j=1}^{M}C_{r'_j}=M'(C_R^F+C_W^F)+(M-M')C_R^F=M'C_W^F+MC_R^F \tag{4}$$

注意,在公式(4)中, C_W^F 中包含由于写出操作引起的擦除操作的代价.可见,对于闪存来说,由于其读写代价不对称,单纯减少随机写操作的次数并不一定能够降低 I/O 操作的总代价,其外存 I/O 操作代价 C_{Total}^F 受两个因素的影响——读次数和写次数.假设降低写操作的次数 M'所节省的代价为 ΔC_W^F ,因此造成读次数增多所带来的额外代价为 ΔC_R^F ,则一个基于闪存的缓冲区算法通过降低写操作的次数所获得的好处为 $\delta=\Delta C_W^F-\Delta C_R^F$.如果置换算法设计不合理,导致δ<0,则系统性能反而下降.同时,已有的基于闪存的缓冲区置换算法[10−14]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,如图 1 所示,该假设与实际情况不符,即使在某种类型的闪存硬盘上δ>0,在其他类型的闪存硬盘上也可能出现δ<0 的情况.

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.当命中一个数据页时,包含该数据页的整个块都移动到 LRU 链表的头部.当发生缺页时,包含最多数据页的块将被置换出缓冲区,所有这个块中的已修改页都将写回闪存.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.然而,这种方法并不能真正减少内存缓冲区中写请求的次数.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 所示的 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.置换区的大小由参数 w 控制.与 LRU 策略相比,CFLRU 在很大程度上减少了对闪存的写操作,

但对于不同类型的闪存来说,所需的读写操作代价的变化非常大,在某些类型的闪存上可能会出现性能下降的问题.

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),其中,Priority Region 和 CFLRU 的 Clean-First Region 相对应.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,只读页放在干净页队列中,所有的修改页放在不同的聚类中,这些聚类用脏页队列进行组织,同一个聚类中修改页的物理位置比较接近.与 FAB 算法的块层 LRU 算法相比,CFDC 中的块大小是可变的.

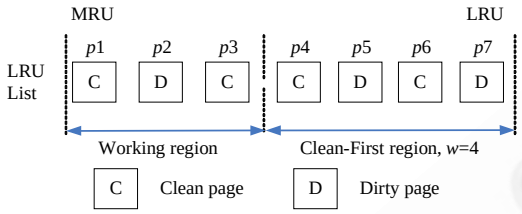


Fig.2 CFLRU replacement policy

图 2 CFLRU 置换策略示意图

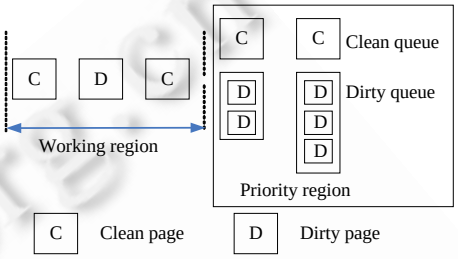


Fig.3 CFDC replacement policy

图 3 CFDC 置换策略示意图

与已有的基于闪存的缓冲区置换策略不同,本文的研究针对实际中同一闪存读写代价不对称和不同闪存不对称性之间存在巨大差异性(如图 1 所示)的问题,在第(3)步实现 $O(1)$操作代价的同时,在第(4)步通过设计合理的置换策略,从而可以在只读页和修改页之间进行公平地选择,也可以应用于各种类型的闪存硬盘,而不是仅适用于读写代价相差非常悬殊的闪存硬盘.

2 基于代价的缓冲区置换算法 CBLRU

CBLRU 的基本思想是,为缓冲区中的每个数据页附加一个权值 W,表示该数据页因驻留内存所带来的好处,该权值会随着时间的推移,逐步递减.每次需要选择一个置换页时,选择权值最小的数据页进行置换.与已有的基于闪存的缓冲区管理算法相比,该方法在挑选置换页时不会强制性地先置换只读页,而是根据每个数据页的权值,选择对系统性能贡献最小的数据页进行置换.

2.1 数据结构

定义 1(向后距离). 假定 r 是缓冲区中的数据页,r 的向后距离 $r.bkw$ 表示从 r 最后一次被访问到目前为止系统访问数据页的次数.

定义 1 提到的向后距离可在算法中用于预测未来的访问情况.例如,假定访问序列为 r_1,r_2,r_4,r_5,r_4,r_3,在 r_3 被访问后,根据定义 1,$r_1.bkw$=6,$r_2.bkw$=5,$r_3.bkw$=1,$r_4.bkw$=2,$r_5.bkw$=3.假设缓冲区的大小是 5,即所有数据页都在缓冲区中,如果需要选择置换页,由于 $r_1.bkw$=6,因此,r_1 可以作为置换页被移出缓冲区,为新来的请求页准备空间.

在 CBLRU 算法中,缓冲区的每个数据页 r 附加有 3 个变量:第 1 个是 F,用以标识 r 是否被修改过,$r.F$=Read 表示 r 为只读页,$r.F$=Write 表示 r 为修改页;第 2 个是向后距离 bkw;最后一个是权值 W,用以表示随着时间的推移,r 驻留内存能够节省的代价,W 的值根据公式(5)得到:

$$r.W=r.Cost/r.bkw \tag{5}$$

其中,$Cost$ 表示 r 被换出并且读入请求页时需要付出的 I/O 代价.当 r 为只读页,即 $r.F$=Read 时,$r.Cost$ 等于从闪存读入一个数据页的代价;当 r 为修改页,即 $r.F$=Write 时,$r.Cost$ 由 3 部分代价组成:(1) 将 r 的内容写回闪存所需的写代价;(2) 从闪存读入请求页的代价;(3) 将 r 的内容写回闪存所引起的级联操作的平均代价.注意,$r.Cost$

仅表示将 r 换出时需要付出的具体操作代价,该代价不包含由于 r 驻留缓冲区对系统性能造成的影响.如果仅仅根据 $r.Cost$ 作为挑选置换页的标准,则只读页将首先被强制换出.由于 $r.bkw$ 表示从 r 最后一次被访问以来系统访问其他数据页的次数,通过比较不同数据页的向后距离,可以预测这些数据页在未来被访问的可能性.因此,我们使用加权值 $r.W$ 作为挑选置换页的标准.$r.W$ 通过将 r 的换出代价和其向后距离综合考虑,可以表示随着时间的变化,r 因驻留缓冲区对系统性能贡献的变化情况.只读页和修改页的置换代价可以通过闪存硬盘制造商公开的资料获取,或者通过执行一定数量的随机读写操作后取平均值得到.

2.2 基本CBLRU算法

CBLRU 将缓冲区中的数据页组织为一个链表,每个数据页按照权值的大小排序.假设缓冲区大小为 5,初始数据页的读入顺序为 r_1,r_2,r_3,r_4,r_5,只读页的置换代价为 1,修改页的置换代价为 100.当 r_5 读入后,由于 $r_1.bkw$=5,$r_2.bkw$=4,$r_3.bkw$=3,$r_4.bkw$=2,$r_5.bkw$=1,根据权值计算公式,缓冲区的状态如图 4(a)所示.

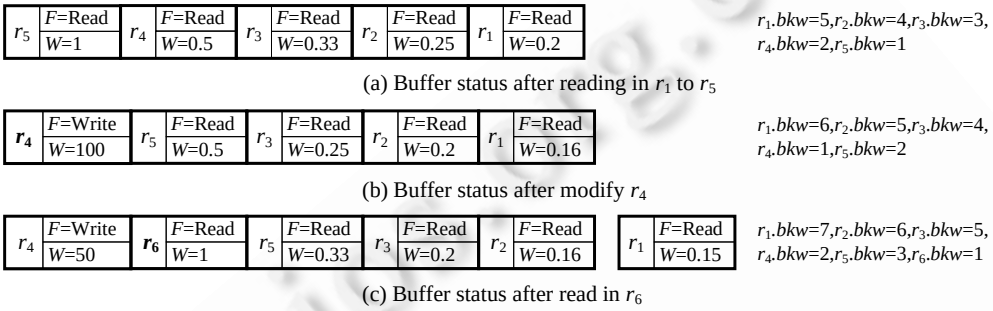


Fig.4 Illustration of the organization of buffer pages

图 4 缓冲区数据页组织示意图

若接下来的存取序列为 r_4,r_6,并且系统对 r_4 的内容进行了修改,则 r_4 和 r_6 之后的缓冲区状态分别如图 4(b)、图 4(c)所示.在图 4(c)中,为了读入 r_6,需要选择一个置换页,通过比较数据页的权值,r_1 被换出.在 r_6 被读入后,由于 $r_6.W<r_4.W$,r_6 按照权值大小插入 r_4 之后.CBLRU 算法见算法 1.

算法 1. **CBLRU**(**page** r, **Buffer B**) /*r 是当前请求页, B 是缓冲区*/
1: **if** *isIn*(*B*,*r*) **then** //判断 r 是否在缓冲区 B 中
2: **if** r is to be modified **then**
3: $r.F$=Write
4: $r.W$←$r.Cost$ //根据读写标记 $r.F$ 使用相应的读或写 $Cost$
5: **else**
6: Remove from B the last page p that has the minimum value W
7: **if** $p.F$=Write **then**
8: Write the content of p into flash disk
9: Fetch r into B
10: $r.F$=Read; $r.W$←$r.Cost$ //根据读写标记 $r.F$ 使用相应的读或写 $Cost$
11: $r.bkw$←1;
12: **for** each page p of B except r **do**
13: $p.bkw$++; $p.W$←$p.C/p.bkw$
14: Adjust the order of all pages in B according to their W

算法 1 的第 1 行判断当前请求页 r 是否在缓冲区 B 中,若在,则在第 2 行、第 3 行根据是否要修改 r 来确定其读写标志,然后在第 4 行根据 $r.F$ 将相应的读写代价赋值给 $r.W$.如果 r 不在 B 中,则在第 6 行首先从 B 中选择权值最小的数据页 p 作为置换页.如果 p 为修改页,即 $p.F$=Write,则在第 8 行首先将 p 的内容写回闪存,然后在第 9 行读入请求页 r,并在第 10 行给 r 的变量赋值.第 11 行将 $r.bkw$ 赋值为 1,对于 B 中剩余的数据页,在第 12 行、

第 13 行调整其权值,并在第 14 行根据数据页的权重调整这些数据页的顺序.

与已有的基于闪存的缓冲区管理算法相比,算法 CBLRU 通过使用新定义的权值将置换代价与时间局部性整合,可以避免已有算法无条件先置换只读页所导致的性能下降问题;同时,由于数据页置换的标准发生了变化,只读页和修改页可以进行公平竞争,从而能够适用于不同类型的闪存.然而,CBLRU 算法仅仅改进了第 1 节所提到的 I/O 代价优化问题.在后续的内容中,将从其他方面对算法的效率进行改进.

2.3 优化的CBLRU算法

在第 1 节的分析中提到,在出现频繁换页的情况下,如果 CPU 操作代价过高,同样会影响到系统性能.对应于算法 1,CPU 操作的代价主要体现在算法 1 的第 12 行~第 14 行,其中,第 12 行、第 13 行修改所有数据页的权值,其代价是 $O(k)$.这里,k 是缓冲区可容纳的数据页个数(以下简称为“缓冲区大小”).第 14 行需要根据权值调整这些数据页的位置,其操作代价是 $O(k\log k)$.即不管数据页是否在缓冲区中,除了 I/O 代价外,CBLRU 的 CPU 操作代价是 $O(k\log k)$.当缓冲区变大时,系统性能将受到严重影响.

定理 1(关系稳定性). r 为当前请求页,r_1 和 r_2 为缓冲区中的两个不同数据页,若 $r_1.F=r_2.F$,则 r_1 和 r_2 的权重关系在系统处理 r 的前后保持不变.

证明:由 $r_1.F=r_2.F$ 可知,$r_1.Cost=r_2.Cost$.

由 r_1 和 r_2 不可能被同时访问可知,$r_1.bkw\neq r_2.bkw$,因此 $r_1.W\neq r_2.W$.

假设 $r_1.W<r_2.W$,根据公式(5)可知,处理 r 之前,$r_1.bkw>r_2.bkw$.

进一步可知,处理 r 之后,$r_1.bkw+1>r_2.bkw+1$,则处理 r 之后,$r_1.W<r_2.W$.

同理可知,当 $r_1.W>r_2.W$ 时,r_1 和 r_2 的权重关系在系统处理 r 前后保持不变.证毕. □

定理 1 说明了对于缓冲区中的非请求页而言,所有的只读页在处理请求页前后的排序关系保持不变,所有的修改页在处理请求页前后的排序关系也保持不变.由于修改页的置换代价较大,在系统处理请求页之后,其顺序可能需要调整到只读页前面,因此在调整数据页的顺序时,可以从权值最小的修改页开始向前调整,而只读页不需要调整.当碰到一个修改页 r 时,假设其前面的修改页为 r_1,则只需将位于 r 和 r_1 之间的只读页的权值与 r 的权值相比较,从而将 r 调整到合适的位置.这就使得算法 1 的第 14 行排序操作的时间复杂度降低为 $O(k)$,k 为缓冲区的大小,从而将整个算法的时间复杂度降低为 $O(k)$.

进一步观察发现,如果将只读页和修改页分开组织为两个链表,则根据定理 1,每次处理请求页前后都无需执行算法 1 第 14 行的排序操作,需要改变的是每次当一个数据页从只读状态被修改后,将其移到修改页链表中,则算法 1 的执行可以得到进一步的简化.这时,缓冲区的组织如图 5 所示(R 表示只读页,W 表示修改页,两个队列的队头为最左边权值最大的数据页).每次选择置换页时,只需要比较两个队列尾部两个数据页的权值,选择权值较小的数据页即可.

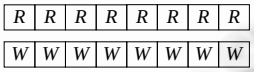


Fig.5 Organization of buffer pages

图 5 缓冲区组织方式

当把缓冲区组织为图 5 所示的方式时,其操作代价仍为 $O(k)$.这是因为每次处理请求页后需要在算法 1 的第 12 行、第 13 行调整缓冲区中所有数据页的权值.为了降低时间复杂度,需要解决两个问题:(1) 每个被处理的请求页应调整到队列的哪个位置?(2) 如何解决每次处理请求页时修改所有数据页权值的问题?

定理 2. r_1 和 r_2 为缓冲区中的两个数据页,并且 $r_1.F=r_2.F$.若 $r_1.bkw>r_2.bkw$,则 $r_1.W<r_2.W$.

证明:数据页的 F 值只能取 Read 或者 Write 中的一个,假设 $r_1.F=r_2.F$=Read,则 $r_1.Cost=r_2.Cost$.根据权值计算公式(5)可知,$r_1.W<r_2.W$.同理,当 $r_1.F=r_2.F$=Write 时,同样有 $r_1.W<r_2.W$. □

定理 2 说明了如果 $r_1.F=r_2.F$,则最近使用的数据页的权值大.根据定义 1,当前请求页的向后距离为 1,因此

对于第(1)个问题,每个被处理的请求页因为在相应队列中有最大权值,只需简单放置在相应队列的队头位置即可.对于第(2)个问题,根据定理 1,每次处理请求页时,缓冲区中其他页的相对位置保持不变.因此,为了简化在每次访问时需要修改所有数据页的权值问题,我们引入一个全局计数器 C,每次处理一个请求页时,C 的值加 1,同时为每个数据页引入一个新的时间戳变量 T,用来表示其最后一次作为请求页时的访问时间,T 的取值并非真正的时间值,而是当前的计数器值.这样,就可以将 CBLRU 算法的时间复杂度从 $O(k)$降低到 $O(1)$.算法 2 是改进的 CBLRU 算法.

算法 2. **CBLRU**(**page** r, **Counter** C, **Buffer B**) /*r 是请求页,C 是计数器,B 是缓冲区*/

```
1:  if isIn(B,r) then                          //判断 r 是否在缓冲区 B 中
2:      if isIn(RQ,r) then                     //判断 r 是否在只读队列 RQ 中
3:          if r is to be modified then Q←WQ   //判断系统是否要对 r 执行修改操作
4:          else Q←RQ
5:      else Q←WQ
6:      SetStatus(r,Q,C,B)                     //设定数据页 r 的状态并将其放入 Q 的队头位置
7:  else
8:      if B is full then
9:          if not Empty(RQ) and not Empty(WQ) then
10:             r1←RQ.GetLastPage()            //第 11 行~第 16 行比较两个队列尾部两个数据页的权值
11:             r2←WQ.GetLastPage()
12:             if r1.W>r2.W then              //r1.W=r1.Cost/(C–r1.T),r2.W 的计算相同
13:                 p←r2
14:             else p←r1
15:         else
16:             if Empty(RQ) then p=WQ.GetLastPage()
17:             else p=RQ.GetLastPage()
18:         if p.F=Write then Write the content of p into flash disk
19:     Fetch r into B and insert it to the head of RQ
20:     r.T←C++
```

过程. **SetStatus**(**page** r, **Queue** Q, **Counter** C, **Buffer B**)

```
1:  r.T←C++
2:  Move r to the head of Q
```

算法 2 将只读页和修改页分别用两个链表维护.如果 r 在缓冲区中,则在第 2 行~第 5 行记录 r 将要放入的队列,并在第 6 行调用 SetStatus 过程处理 r 的移动.如果 r 不在缓冲区中并且缓冲区不空,则在第 9 行~第 18 行通过比较只读队列和修改队列队尾数据页对应的权值,选择权值小的数据页予以置换.然后在第 19 行从闪存上读入 r,并将其插入只读队列 RQ 的队头即可.注意,在第 12 行比较权值的时候,其计算公式在该行后面的注释中.

例 1:假设缓冲区大小为 4,读数据页的代价为 1,写数据页的代价为 2,对于操作序列 $r_1,r_2,r_3,r_4,r_4,r_3,r_5,r_2,r_5,r_6$ 而言,缓冲区状态的变化过程如图 6 所示(其中,细线条框表示只读链表中的数据页,粗线条框表示修改页链表中的数据页,虚的粗线框表示的是当前操作的数据页).当读入 r_1,r_2,r_3,r_4 后,状态如图 6(a)所示.当修改了 r_4 后,计数器的值 C=6,缓冲区状态如图 6(b)所示.修改完 r_3 后,C=7,缓冲区状态如图 6(c)所示.下一个需要读入的数据页是 r_5,由于 r_5 不在缓冲区中,需要选择一个置换页,根据算法 2 的第 9 行~第 18 行,只需从只读队列和修改队列的两个队尾元素中挑选一个即可.根据算法 2 第 12 行的计算公式,$r_1.W=r_1.Cost/(C-r_1.T)=1/6$,$r_4.W=r_4.Cost/(C-r_4.T)=2/2=1$.由于 $r_1.W<r_4.W$,因此将 r_1 换出,读入 r_5 的缓冲区状态如图 6(d)所示.接下来读取 r_2 和 r_5,状态分别如图 6(e)、图 6(f)所示.最后,当需要操作 r_6 时,由于 r_6 不在缓冲区中,首先需要选择一个置换页,根据算法 2,需要从 r_2 和 r_4 中选择一个权值小的数据页进行置换.由于 $r_2.W=1/2=0.5$,而 $r_4.W=2/5=0.4$,因此 r_4 将被首先换出,然后读入 r_6,如图 6(g)所示.尽管 r_4 为修改页,其置换代价较大,但由于长时间没有使用,有必要将其换出缓冲区从而扩大缓冲区的有效空间.

另外,计数器 C 的值会随着对数据页的访问不断变大,如果不加以控制,可能会发生数据溢出的情况.

推论 1. 对于只读页或者修改页链表中的两个数据页 r_1,r_2,即 $r_1.F=r_2.F$,若 $r_1.W>r_2.W$,则 $r_1.T>r_2.T$.

证明:由于 $r_1.F=r_2.F$,则 $r_1.Cost=r_2.Cost$.

由于 $r_1.W>r_2.W$,根据公式(5)可知,$r_1.bkw<r_2.bkw$.

根据算法 2,$r_1.bkw=C-r_1.T$,即 $r_1.T=C-r_1.bkw$,$r_2.T=C-r_2.bkw$.

由于 $r_1.bkw<r_2.bkw$,因此 $r_1.T>r_2.T$. □

推论 2. 只读页或者修改页队列中,从队头至队尾,数据页的 T 值按照降序排序.

证明:由于只读页和修改页队列中的数据页的权值按降序排序,根据推论 1,数据页的 T 值按照降序有序.□

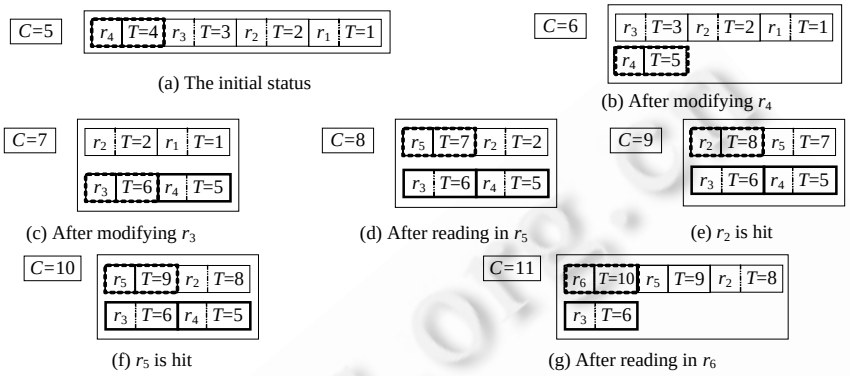


Fig.6 Illustration of the optimized CBLRU for sequence $r_1,r_2,r_3,r_4,r_4,r_3,r_5,r_2,r_5,r_6$

图 6 优化的 CBLRU 算法执行操作序列 $r_1,r_2,r_3,r_4,r_4,r_3,r_5,r_2,r_5,r_6$ 时的缓冲区状态变化

推论 1 说明,在同一个数据页队列中,若一个数据页权值越大,则其 T 值越大;而推论 2 则说明了在只读页或者修改页队列中,数据页的 T 值按照降序有序.这就为我们处理计数器数值溢出提供了便利.我们用 $r_{Max}.T-r_{Min}.T$ 表示最大 T 值的数据页和最小 T 值的数据页之间 T 值之差.其中,r_{Max} 为两个数据页队列的队头元素中 T 值较大的数据页,而 r_{Min} 为两个数据页队列队尾元素中 T 值较小的数据页.$r_{Max}.T-r_{Min}.T$ 的值越小,说明数据页的访问模式趋向于均匀分布的随机访问模式;反之,若 $r_{Max}.T-r_{Min}.T$ 越大,说明部分数据页循环访问.这时,在没有数据页换出的情况下,部分数据页多次循环访问将造成计数器的值无谓增大.因此,我们对算法 2 的 *SetStatus*(*r*,*Q*,*C*,*B*)过程进行改进,以消除计数器数值溢出的情况.我们用 ρk 来控制 $r_{Max}.T-r_{Min}.T$ 的大小,ρ 为自定义变量,其值的确定根据实验情况而定;k 为缓冲区大小.过程 1 中,第 1 行判断 $r_{Max}.T-r_{Min}.T\leqslant\rho k$ 是否成立(显然,$\rho\geqslant 1$):若成立,则在第 2 行给 $r.T$ 赋值并将计数器值加 1,然后在第 3 行将其插入队列 Q 的头部;否则,说明缓冲区中存在循环存取的序列.这时,为了控制计数器值无谓增大的情况,只需要在第 5 行、第 6 行交换队头元素 p 和 r 的值及位置即可.如果 C 即将溢出,即 $C+\rho k=C_{MAX}$,则在第 8 行~第 10 行将计数器本身及缓冲区中的所有数据页的 T 值同时减去 $r_{Min}.T$.实际运行中,至少处理 $C_{MAX}-\rho k$ 个数据页后才调整 C 及数据页的 T 值.即,过程 1 中第 8 行~第 10 行的执行概率是 $k/(C_{MAX}-\rho k)$.因此,尽管第 8 行、第 9 行的时间复杂度为 $O(k)$,由于 $k<<(C_{MAX}-\rho k)$,过程 1 的平均时间复杂度仍为 $O(1)$.

过程 1. **SetStatus**(**page** *r*, **Queue** *Q*, **Counter** *C*, **Buffer B**)

```
/*r 是请求页,p 是 r 在操作之后被放入的队列中的头元素,C 是计数器,B 是缓冲区*/
1:  if r_Max.T−r_Min.T≤ρk then          //r_Max 和 r_Min 的计算如前一段所述
2:      r.T←C++
3:      Move r to the head of Q
4:  else
5:      p←the first page of Q
6:      p↔r                             //交换 p 和 r 的 T 值及在队列 Q 中的位置
7:  if C+ρk=C_MAX then
8:      for each page p in B do
9:          p.T←p.T−r_Min.T
```

10: $C \leftarrow C - r_{\text{Min}}.T$

2.4 分 析

CBLRU 的自适应性首先体现在其基于代价的置换策略.当需要选择一个置换页时,CBLRU 从只读队列或者修改队列中根据数据页的权值公平地选择合适的置换页.当读操作较多时,只读队列会逐步变大;相反,修改队列会逐步变大.因此,CBLRU 能够很好地处理同一闪存读写代价的不对称性以及不同闪存读写代价不对称性之间的巨大差异性,可以应用到不同类型的闪存硬盘上.其次,CBLRU 的自适应性体现在可以很好地处理序列存取模式和循环存取模式.这是因为 CBLRU 根据数据页的权值来选择置换页,这样就使得数据页的置换顺序和请求顺序完全不同.

3 实验评价

3.1 实验目的、实验环境及测试方案

本文的实验目的是验证 CBLRU 算法针对不同读写代价的闪存硬盘的有效性.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 的随机读写操作代价的比率分别是 1:118 和 1:2,读写代价可以通过 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.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.这两种闪存硬盘已在许多应用中被当作辅存来使用.

对 SSD 来说,物理读写次数决定了缓冲区置换算法的性能.然而,FTL 层是设备相关的,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,为了获取精确的读写操作的次数,类似于很多已有文献[5,10−12]采用模拟环境的方法,我们采用文献[17]提出的模拟环境进行测试.我们实现了 4 种置换策略来进行比较,即 LRU,CFLRU[10],CFDC[14]及本文提出的 CBLRU.所有的置换策略都用 Visual C++实现.实验中,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.这些参数都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 32 000 个的物理页,每页为 2KB.缓冲区的大小范围从 2 000~8 000 个数据页.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.

为了尽可能真实地模拟实际数据库系统运行时的数据页访问模式,我们采用文献[5]所用的测试方式进行测试.我们生成了 4 种符合 Zipf 分布的测试数据,其统计信息见表 1.其中:“读/写比例”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.

Table 1 Statistics of the traces used in our experiment

表 1 实验所用测试数据的统计信息

Trace ID	Total request	Ratio of read to write	Locality
*T*1	3 000 000	90%/10%	60%/40%
*T*2	3 000 000	80%/20%	50%/50%
*T*3	3 000 000	60%/40%	60%/40%
*T*4	3 000 000	80%/20%	80%/20%

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.我们不用命中率作为评价标准,因为读写代价不对称使得命中率不能真正反映系统的整体性能.实验中的写操作次数的测试结果中包括由于对 SSD 进行擦除操作而导致的写操作的次数.

3.2 实验结果及分析

3.2.1 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(a)~图 7(d)展示了 3 种已有方法和本文提出的 CBLRU 方法在 FD1 上运行 *T*1~*T*4 时

随机读次数比较.可以看出,当闪存读写代价比例差异巨大时(1:118),与 LRU 相比,基于闪存的算法(CFLRU, CFDC 及 CBLRU)需要更多的读次数,但本文提出的 CBLRU 所需的读次数在 *T*1~*T*4 上明显少于 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.原因在于,当读写操作大代价比例过大时,CBLRU 中用于存储修改页的队列中包含的数据页远多于 CFLRU 和 CFDC,而这部分数据页也存在相应的读操作,因而从整体上导致 CBLRU 的读次数比 CFLRU 和 CFDC 少.

写操作性能比较.图 7(e)~图 7(h)展示了 3 种已有方法和本文提出的 CBLRU 方法在 FD1 上运行 *T*1~*T*4 时随机写次数比较.可以看出,基于闪存的算法(CFLRU,CFDC 及 CBLRU)涉及的写操作的次数远少于基于磁盘的 LRU 算法.同时,尽管 CFLRU 和 CFDC 首先置换只读页,本文提出的 CBLRU 方法依然好于 CFLRU 和 CFDC.原因在于,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,因而修改页的命中率反而上升,从而降低了随机写的次数.

运行时间比较.图 7(i)~图 7(l)展示了不同方法在 FD1 上运行 *T*1~*T*4 时运行时间的比较.可以看出,基于闪存的算法(CFLRU,CFDC 及 CBLRU)所需的运行时间远少于基于磁盘的 LRU 算法.这是因为对于 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,CBLRU 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,最终导致整体性能的提升.同时还可以看出,对于 FD1 来说,CBLRU 所需运行时间少于 CFLRU 和 CFDC.

3.2.2 读写操作代价差异小的 SSD 上性能比较

读操作性能比较.图 8(a)~图 8(d)展示了 3 种已有方法和本文提出的 CBLRU 方法在 FD2 上运行 *T*1~*T*4 时随机读次数的比较.可以看出:与 LRU 相比,由于 CFLRU 和 CFDC 在不考虑随机读操作代价的前提下优先置换只读页,因此所需的物理读操作次数远多于 LRU;而 CBLRU 采用权重作为选择置换页的标准,可以有效避免 CFLRU 和 CFDC 存在的问题,在读写操作代价差异较小的闪存上,CBLRU 所需的物理读次数在多数情况下和 LRU 相差甚微,只有少数情况下稍微多于 LRU.

写操作性能比较.图 8(e)~图 8(h)展示了 3 种已有方法和本文提出的 CBLRU 方法在 FD2 上运行 *T*1~*T*4 时随机写次数的比较.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;尤其是 CFLRU,由于其置换区大小比 CFDC 大,而这部分区间中没有只读页,因此 CFLRU 所需的物理写操作次数少于 CFDC;但同时,其所需的物理读次数也多于 CFDC.本文提出的 CBLRU 会根据不同闪存的读写代价比来调整不同状态数据页的权重,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.

运行时间比较.图 8(i)~图 8(l)展示了 3 种已有方法和本文提出的 CBLRU 方法在 FD2 上运行 *T*1~*T*4 时运行时间的比较.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.这是因为其节省的写操作的代价远小于浪费在读操作上的代价.而本文提出的 CBLRU 读次数和 LRU 差不多,且写次数比 LRU 少,因此整体性能最好.

综上,从实验结果可以看出,本文提出的 CBLRU 根据数据页的权重选择合适的数据页进行置换,以最小化系统性能为目标,可以在只读页和修改页之间进行公平的比较.根据第 3.2.1 节的实验结果,当给定闪存的读写操作代价相差过大时,CBLRU 给修改页以较高的权重.这时,其修改页和只读页所在队列的长度将动态变化,大多数情况下,修改页所在队列长度大于 CFLRU 和 CFDC 中保存修改页的队列长度,因此可以获得比 CFLRU 和 CFDC 更好的性能.同时,根据第 3.2.2 节的实验结果,当给定闪存的读写操作代价相差较小时,CBLRU 将给只读页以足够的重视,因此同样可以在读写代价相差较小的闪存上获得很好的性能.与 CBLRU 不同,LRU, CFLRU 和 CFDC 完全无视闪存读写代价的比例随着闪存硬盘类型的变化而变化的事实,因此,虽然 LRU 可以在读写代价相差不大的闪存上获得较好的性能,CFLRU 和 CFDC 可以在读写代价相差很大的闪存上获得较好的性能,但这些方法都不能适应所有类型的闪存.由于 CBLRU 会根据不同类型闪存的读写代价的比例调整数据页的权重,因此可以适用于各种类型的闪存硬盘.

4 总 结

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巨大差异性的问题,提出一种基于闪存硬盘的自适应缓冲区管理算法 CBLRU.该算法为每个数据页附加一个权值来综合衡量数据页的置换代价和其驻留内存的影响.当需要选择置换页时,具有最小权值的数据页被首先置换.与已有方法相比,CBLRU 算法在延长频繁访问数据页和修改页驻留缓冲区时间的同时,不至于使其长期占用缓冲区中的有效空间.实验结果验证了 CBLRU 在不同类型闪存硬盘上的性能优势.

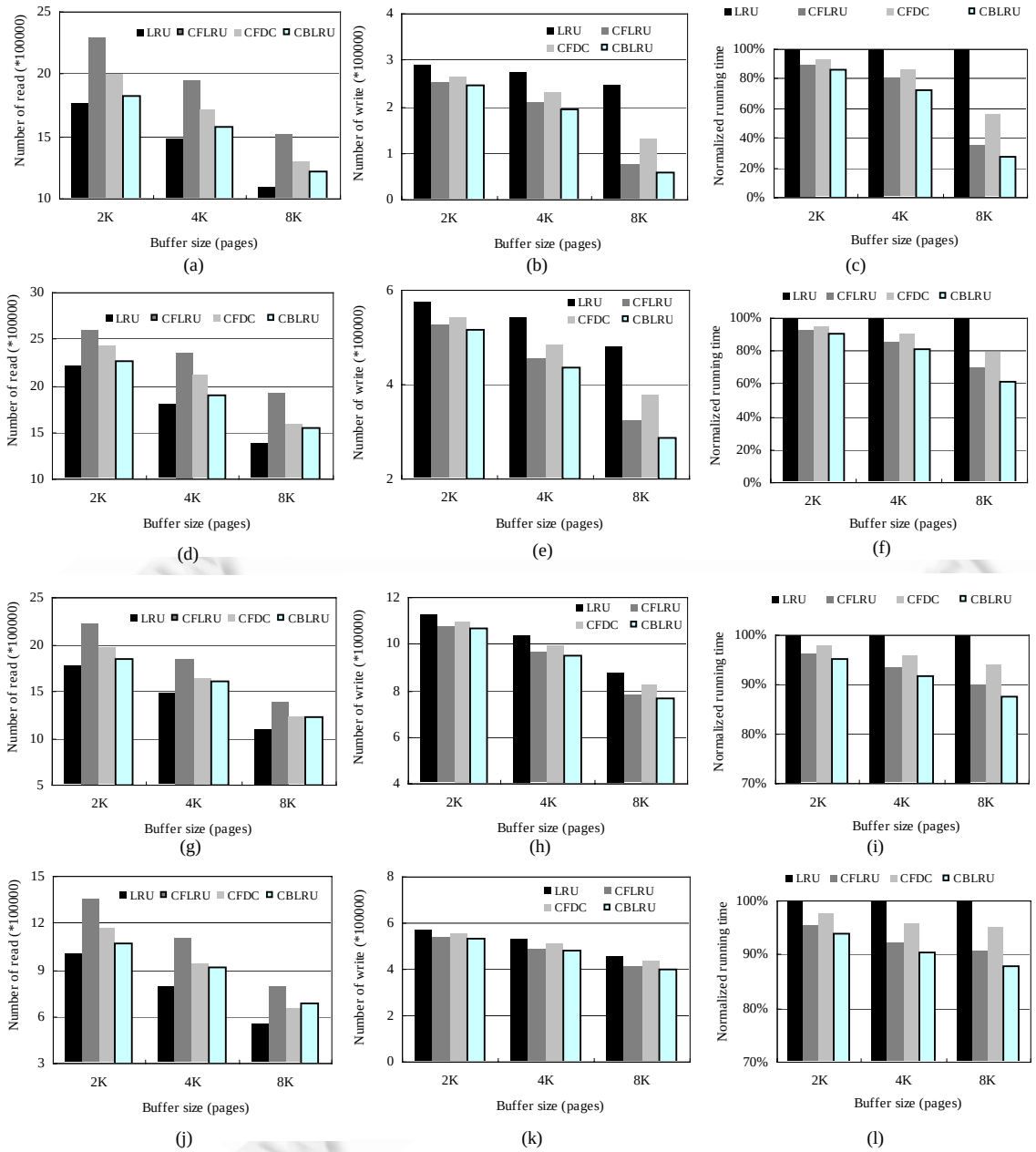


Fig.7 Comparison of read, write and running time for different methods when executing *T*1 to *T*4 on FD1

图 7 不同方法在 FD1 上运行 *T*1~*T*4 时读、写次数及运行时间比较

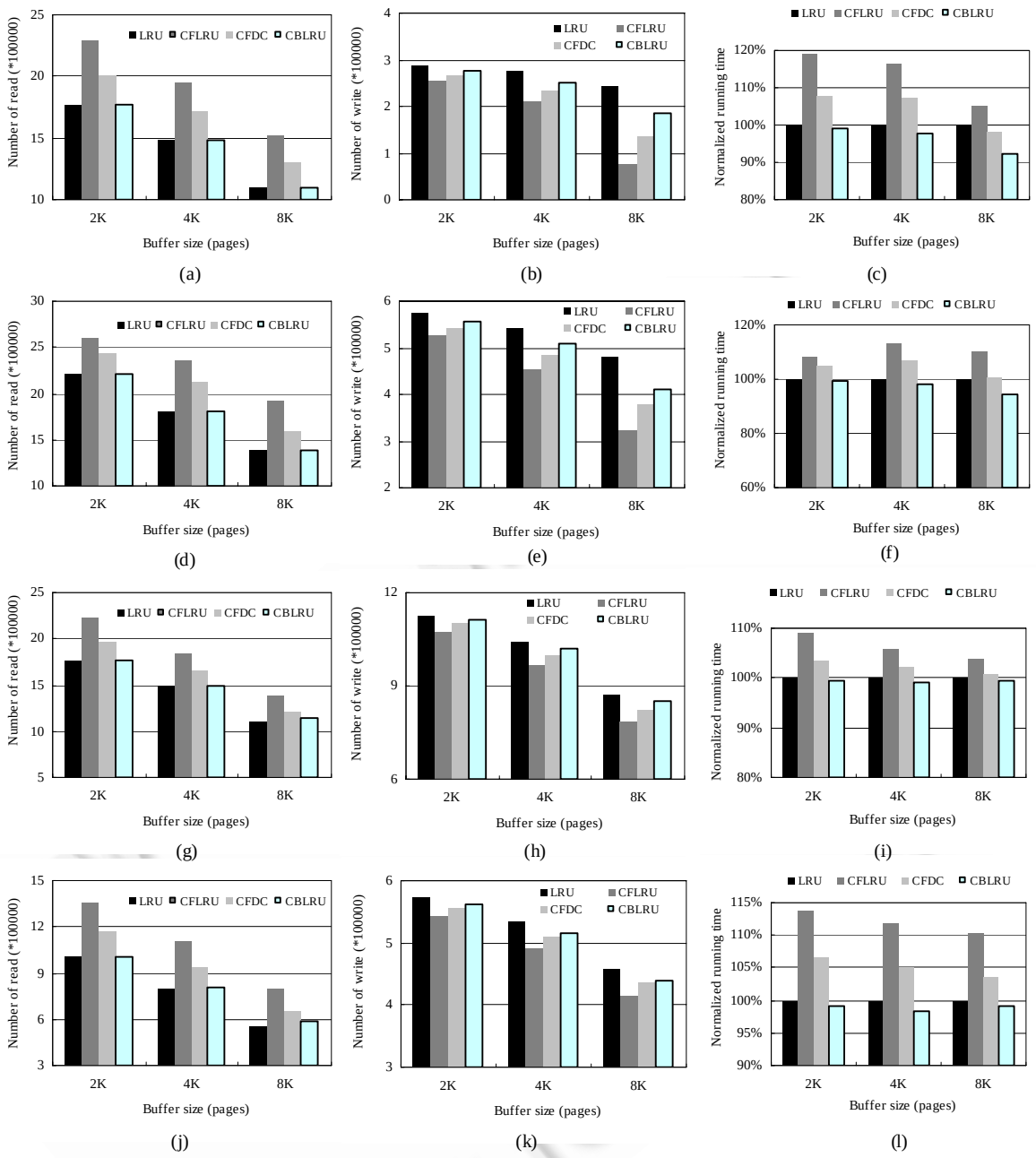


Fig.8 Comparison of read, write and running time for different methods when executing *T*1 to *T*4 on FD2

图 8 不同方法在 FD2 上运行 *T*1~*T*4 时读、写次数及运行时间比较

References:

[1] Grey J. A radical view of flash disks. 2006. http://research.microsoft.com/~Gray/talks/Flash_Is_Good.ppt

[2] Babaoglu Ö, 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. [doi: 10.1145/800216.806595]

[3] Robinson JT, Devarakonda MV. Data cache management using frequency-based replacement. In: Nutt GJ, ed. Proc. of the '90 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems. New York: ACM Press, 1990. 134−142. [doi: 10.1145/98457.98523]

[4] O'Neil EJ, O'Neil PE, Weikum G. The LRU-*k* page replacement algorithm for database disk buffering. In: Buneman P, Jajodia S, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1993. 297−306. [doi: 10.1145/170035.170081]

[5] Johnson T, Shasha D. 2Q: A low overhead high performance buffer management replacement algorithm. In: Bocca JB, ed. Proc. of the 20th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1994. 439−450.

[6] Jiang S, Zhang XD. Making LRU friendly to weak locality workloads: A novel replacement algorithm to improve buffer cache performance. IEEE Trans. on Computers, 2005,54(8):939−952. [doi: 10.1109/TC.2005.130]

[7] Megiddo N, Modha DS. ARC: A self-tuning, low overhead replacement cache. In: Honeyman P, ed. Proc. of the Conf. on File and Storage Technologies (FAST 2003). Berkeley: USENIX, 2003. 115−130.

[8] Lee D, Choi J, Kim JH, Noh SH, Min SL, Cho Y, Kim CS. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. on Computers, 2001,50(12):1352−1361. [doi: 10.1109/TC.2001.970573]

[9] Effelsberg W, Haerder T. Principles of database buffer management. ACM Trans. on Database Systems, 1984,9(4):560−595. [doi: 10.1145/1994.2022]

[10] Park SY, Jung D, Kang JU, Kim JS, Lee J. Cflru: A replacement algorithm for flash memory. In: Hong S, ed. Proc. of the 2006 Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems. New York: ACM Press, 2006. 234−241. [doi: 10.1145/1176760.1176789]

[11] Jo H, Kang JU, Park SY, Kim JS, Lee J. Fab: Flashaware buffer management policy for portable media players. IEEE Trans. on Consumer Electronics, 2006,52(2):485−493. [doi: 10.1109/TCE.2006.1649669]

[12] Kim H, Ahn S. Bplru: A buffer management scheme for improving random writes in flash storage. In: Proc. of the 6th USENIX Conf. on File and Storage Technologies. Berkely: USENIX, 2008. 239−252.

[13] Koltsidas I, Viglas SD. Flashing up the storage layer. Proc. of the VLDB Endowment, 2008,1(1):514−525. [doi: 10.1145/1453856.1453913]

[14] Ou Y, Härerder T, Jin PQ. Cfdc: A flash-aware replacement policy for database buffer management. In: Boncz PA, ed. Proc. of the 5th Int'l Workshop on Data Management on New Hardware. New York: ACM Press, 2009. 15−20. [doi: 10.1145/1565694.1565698]

[15] Chrobak M, Karloff HJ, Payne TH, Vishwanathan S. New results on server problems. SIAM Journal on Discrete Mathematics, 1991,4(2):172−181. [doi: 10.1137/0404017]

[16] Jin PQ, Su X, Li Z, Yue LH. A flexible simulation environment for flash-aware algorithms. In: Cheung DW, ed. Proc. of the 18th ACM Conf. on Information and Knowledge Management. New York: ACM Press, 2009. 2093−2094. [doi: 10.1145/1645953.1646319]

汤显(1978－),女,山东荣成人,博士生,主要研究领域为闪存数据库.

梁智超(1986－),男,硕士生,主要研究领域为闪存数据库.

孟小峰(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为 Web 数据集成,XML 数据库,移动对象管理.

卢泽萍(1985－),女,硕士生,主要研究领域为闪存数据库.