基于替换概率的闪存数据库缓冲区替换算法

林子雨　赖明星　邹　权　薛永生　杨思颖
（厦门大学计算机科学系　福建 厦门　361005）

摘　要　闪存具有和传统磁盘不同的特性，包括写前擦除、异地更新、读写延迟非对称等.传统的面向磁盘的缓冲区替换算法无法在闪存数据库系统中获得较好的性能.文中提出了一种新的面向闪存数据库的缓冲区替换算法——APB-LRU，其特点：(1)该算法将缓冲区分为冷区和热区，用来捕获数据访问频度，前者用于存放只访问过一次的数据页，后者用于存放至少访问过两次的数据页；(2)采用了其它研究所没有的概率替换机制，即以较大的概率替换冷区中的干净页，以较小的概率替换冷区中的脏页，从而避免了冷脏页长期驻留缓冲区的情况，提高了命中率，获得了较好的整体性能；(3)设计了冷、热区比例动态变化机制，可以根据工作负载的变化动态调整冷、热区所占缓冲区的比例，从而使得替换算法在不同的负载模式下都可以取得较好的性能.基于不同测试数据集的大量实验结果表明，APB-LRU 算法具有比其它已有的算法更好的性能.

关键词　闪存；数据库；缓冲区替换算法；概率
中图法分类号　TP311　　**DOI 号**　10.3724/SP.J.1016.2013.01568

Probability-Based Buffer Replacement Algorithm for Flash-Based Databases

LIN Zi-Yu　LAI Ming-Xing　ZOU Quan　XUE Yong-Sheng　YANG Si-Ying
(*Department of Computer Science, Xiamen University, Xiamen, Fujian*　361005)

Abstract　Different from traditional disk, flash memory has characteristics of erase-before-write, out-of-place update and asymmetric I/O latencies for read, write, and erase operations. Traditional buffer replacement algorithms are not optimized for flash-based database systems and do not take the characteristics of flash memory into consideration, which means they are not able to achieve good performance when directly used in flash-based database systems. This paper proposes a new approach to buffer management for flash-based database systems, i.e., APB-LRU. Firstly, in APB-LRU, the buffer is divided into two regions, i.e., cold region and hot region, so as to get the access frequency information of various data pages. Cold region holds those pages only accessed once, and hot region contains those pages accessed more than once. Secondly, unlike other existing methods, APB-LRU adopts a new mechanism of replacement based on probability, in which clean pages are replaced with greater probability and dirty pages are replaced with smaller probability, so that clean pages in cold region will not immediately be replaced and cold dirty pages can not reside in the buffer for a long time. Through this way, APB-LRU achieves a high buffer hit ratio and a better overall performance than other available methods. Thirdly, dynamical adjustment of the ratio between the sizes of cold region and hot region is proposed, which is able to dynamically change the ratio according to the real workloads with various access pat-

收稿日期：2013-03-15；最终修改稿收到日期：2013-06-03.本课题得到厦门大学基础创新科研基金(中央高校基本科研业务费专项资金)(2011121049，2012121030)、国家自然科学基金(61001013，61102136，61202012)、福建省自然科学基金(2011J05156，2011J05158，2013J05099)资助.**林子雨**，男，1978 年生，博士，助理教授，中国计算机学会(CCF)会员，主要研究方向为数据库、数据仓库和数据挖掘.E-mail：ziyulin@xmu.edu.cn.**赖明星**，男，1989 年生，硕士研究生，主要研究方向为闪存数据库.**邹　权**(通信作者)，男，1982 年生，博士，助理教授，主要研究方向为生物信息学和数据挖掘.E-mail：zouquan@xmu.edu.cn.**薛永生**，男，1946 年生，教授，主要研究领域为数据库理论.**杨思颖**，女，1992 年生，本科生，主要研究方向为闪存数据库.

terns, so that good performance can be achieved under various workloads. We carry out large amounts of experiments with different datasets, and the experimental results show that APB-LRU is superior to its competitors in most cases.

Keywords　flash; database; buffer replacement algorithm; probability

1　引　言

闪存是一种典型的电可擦除可编程只读存储器(Electrically Erasable Programmable Read Only Memory,EEPROM),属于非易失性存储,断电后数据也不会丢失.基于闪存的存储设备,具有速度快、延迟小、能耗低、体积小和可抗震等优良特性,已经广泛地应用于数码相机、移动电话、笔记本等消费类电子设备和企业数据存储产品中[1].

随着闪存的容量的增加和价格的降低,闪存相对于磁盘的竞争优势变得更加明显,已经大量应用于企业级别的数据存储系统中,比如许多企业的数据库系统已经开始使用闪存固态盘来替换磁盘作为底层的存储介质.闪存比磁盘具有更高的读写速度,因此闪存数据库系统可以比基于磁盘的数据库系统获得更好的性能.但是,传统的数据库系统中的各类数据结构和算法(比如索引和缓冲区替换算法),都是专门针对磁盘开发的,没有考虑闪存的自身特性,因此无法充分发挥闪存的独有特性来最大程度地提升数据库的整体性能,甚至在某些特殊的负载下,会表现出比基于磁盘的数据库更差的性能.因此,必须重新设计针对闪存数据库的数据结构和算法.目前,在这个方面已经存在大量相关的研究工作[2-5],其中一个比较热门的研究问题就是设计面向闪存数据库系统的缓冲区替换算法.

缓冲区替换算法,可以优化 I/O 序列,减少磁盘的访问次数,从而提高存储系统的整体效率.但是,已有的面向磁盘的缓冲区替换算法[6-8]通常假设底层的存储设备具有相同的读写操作延迟,在设计算法时,只是简单地以最大化命中率为目标,从而达到减少磁盘读写操作次数的目的.但是,这种读写延迟一致性的假设在闪存中是不成立的,闪存的读操作速度要比写操作速度快一个数量级.缓冲区替换算法必须对写操作和读操作进行区分,才能获得更好的性能.例如,在缓存中保留一个写操作比较集中的脏页所能带来的收益,要比保留一个读操作集中的干净页的收益大.而在传统的替换算法中,往往会把很多脏页替换出去.替换脏页会带来大量的 I/O 开销,这就恶化了基于闪存的数据库的性能.

为了减少脏页回写的代价,在设计面向闪存的缓冲区替换算法时,可以以适当增加读操作为代价,来减少写操作的次数,从而获得整体性能的提升.CCF-LRU[9]和 AD-LRU[10]是两个面向闪存的缓冲区替换算法,它们都将缓冲区分成两个区,分别存放不同替换代价的数据页,以此来捕获数据访问的频度,并且它们都在替换区中无条件优先替换干净页,以达到增加读操作次数,减少写操作次数的目的,所以在使用固态盘作为底层存储设备的数据库系统中获得了较好的性能,但是它们都有两个严重的缺陷:(1)在替换区中无条件优先替换干净页,这种替换方式会导致脏页完全占据缓冲区,从引起干净页刚读入缓冲区就立即被选择作为驱逐页的情况,这种替换方式在某些负载下会严重恶化命中率,增加大量的读操作,这在读写代价差异较小的固态盘上尤为不利;(2)冷脏页长期驻留缓冲区,浪费了宝贵的缓冲区资源.

为了解决上述问题,本文首先提出了一种朴素算法 PB-LRU,该算法将缓冲区分为两个 LRU 队列——冷区和热区,分别存放不同访问频度的数据页,其中在冷区队列中存放那些只访问过一次的数据页,在热区队列中存放那些至少访问过两次的数据页,冷区和热区的大小是固定的.对于冷区队列,PB-LRU 算法采用了完全不同于其它研究的替换策略,即以不同的概率替换不同的数据页,采用这种策略有如下好处:(1)以较大概率替换干净页,以较小的概率替换脏页,虽然增加了读操作次数,却减少了写操作的次数,从而有效提升了算法总体性能;(2)能够有效避免冷脏页长期驻留缓冲区,浪费宝贵的缓冲区资源;(3)避免了脏页完全占据缓冲区而导致的“干净页刚读入缓存就立即被选择为驱逐页”的情况.在此基础之上,我们进一步将冷区分为干净页队列和脏页队列,以此加快驱逐页的查找速度;此外,通过实验我们还发现,对于访问局部性较高的工作负载,热区所占缓冲区比例较大时能获得较好的性能,对于访问局部性较低的工作负载,热区

所占比例适当减少时能获得较好的性能.由此,本文在朴素算法 PB-LRU 的基础上,进一步提出了一种自适应的、基于替换概率的缓冲区置换算法——APB-LRU,该算法能够根据数据访问模式动态调整冷、热区所占的比例.实验结果表明,APB-LRU 算法能够明显提高缓冲区的命中率,且在不同访问模式的工作负载下,都能够取得比其它已有算法更好的性能.

本文第 2 节介绍相关工作并对几种代表性研究成果的优缺点进行分析;第 3 节介绍本文提出的基于替换概率的朴素算法——PB-LRU 算法;第 4 节给出朴素算法的改进版本,即一种自适应的、基于替换概率的算法——APB-LRU 算法;第 5 节给出实验设计和结果分析;第 6 节总结全文并展望未来工作.

2 相关工作

LRU 是经典的缓冲区替换算法,在现有的系统中,基本上都采用 LRU 算法或类 LRU 算法.LRU 根据页面的新颖度(recency)信息进行缓冲区管理.在实现时,它会构造一个链表,每个链表元素代表一个缓存页面.当发生缓冲区"脱靶"(访问的页面不在缓冲区中,需要到外部存储中读取该页)且缓冲区未满时,则把该页插入到链表的 MRU(Most Recently Used)位置;如果缓冲区已满,则替换链表的 LRU 位置上的页,并把新页插入到 MRU 位置;若访问的页面在缓冲区中"命中"(访问的数据已经在缓冲区中,不需要到外部存储中去读取),则把该页面移到链表的 MRU 位置.LRU 算法在访问局部性较高的时候,能够获得较好的性能,而且 LRU 的各个操作都是常量时间复杂度.但是,LRU 算法存在如下缺点:(1)没有捕获数据访问的频度(frequency)特征;(2)缓冲区中的数据容易被一次扫描应用污染.

包括 LRU 在内的传统的缓冲区替换算法,通常假设底层存储介质的读操作和写操作代价相同,但是对于闪存而言,写入一个页的代价比读取一个页的代价高很多,因此直接将传统的缓冲区替换算法应用于固态盘中,是无法获得较好的数据库整体性能的.

CFLRU[11]是第一个面向闪存的缓冲区替换算法,它利用闪存读写延迟的非对称性,提出了一种优先替换干净页的缓冲区替换策略,主要思想是:(1)将 LRU 链表逻辑上分为工作区和替换区两部分;(2)替换操作总是发生在替换区,并且总是优先替换替换区中的干净页,如果替换区中没有干净页,则采用和 LRU 算法一样的顺序替换其它页.CFLRU 在替换区中总是优先替换干净页,导致算法命中率有所降低,但是它减少了闪存的写操作和擦除操作的次数,因此可以获得整体性能的提升.CFLRU 算法的缺陷主要表现在 4 个方面:(1)该算法需要人为确定替换区的大小(w),而一个固定的 w 值很难适应不同的工作负载;(2)在某些情况下查找代价较高,因为该算法在每次选择驱逐页时,都需要沿着 LRU 链表反向查找干净页驱逐页,当链表较长时,就会需要较高的查找代价;(3)和 LRU 算法一样,没有充分利用数据页的访问频度特征,从而无法获得较好的性能;(4)不能阻止顺序扫描操作对缓冲区的污染.

LRU-WSR[12]算法是对 CFLRU 算法的一种改进,该算法捕获了脏页的频度特征,能够有效地避免冷脏页长期驻留内存.LRU-WSR 算法为每个链表元素添加一个冷标识位,用于标识该页是否为冷页.LRU-WSR 算法在选择驱逐页时,首先检查 LRU 位置上的页是否为脏页,如果是脏页,并且冷标识位为 0,则认为该页是非冷的脏页,那么就将其冷标识位设置为 1,并将该页移到 MRU 位置,然后,再一次选择 LRU 位置上的页进行下一次判断.如果当前选择的页是干净页,或者是冷标识位为 1 的脏页,则替换该页.此外,在任何时候引用脏页,都会将它的冷标识位设置为 0.LRU-WSR 算法的不足是,虽然考虑了脏页的冷热属性,但是没有考虑干净页的冷热属性,而且该算法不能阻止一次扫描操作对缓冲区的污染.

CCF-LRU(Cold-Clean-First LRU)[9]算法捕获了干净页和脏页的频度特征,它将缓冲区分为两个 LRU 链表,即 L_1(Mixed LRU List)和 L_2(Cold Clean LRU List),前者用于存放脏页和热干净页,后者用于存放冷干净页.CCF-LRU 算法总是优先替换 L_2 链表中的数据页,当 L_2 链表为空时,才在 L_1 链表中用 LRU-WSR 算法选择驱逐页.CCF-LRU 算法的优点是:(1)捕获了干净页和脏页的频度特征;(2)将缓冲区分为两个队列,能够有效地阻止一次扫描操作对缓冲区的污染;(3)在访问局部性很高的时候非常有效.但是,CCF-LRU 算法也存在明显的不足,它无法控制 L_2 链表的长度,在某些情形下,可能导致刚读入缓冲区的干净页马上被替换出去,严重降低命中率.

AD-LRU(Adaptive Double LRU)[10]算法是对CCF-LRU算法的改进，主要思想如下：(1)将缓冲区分为热区和冷区，热区中存放那些至少访问过两次的页，冷区中存放那些只访问过一次的页；(2)冷、热区的大小是动态调整的；(3)冷区容量有一个下界(min_lc)，当冷区的容量大于等于min_lc时，替换操作发生在冷区，当冷区的容量小于min_lc时，替换操作发生在热区.相对于CCF-LRU算法，AD-LRU算法有了进一步的提升，在读写局部性较低的情况下，也能表现出较好的性能.但是，它并没有彻底解决CCF-LRU算法中的问题.首先，AD-LRU在冷区中总是优先替换干净页，这种替换策略依然无法避免干净页刚读入缓冲区就被替换的情况；其次，优先替换干净页将导致冷脏页长期驻留缓冲区，浪费了宝贵的缓冲区资源，降低了命中率，在干净页较多的访问序列中尤为不利.

相反，本文提出的PB-LRU算法能够有效地解决CCF-LRU算法和AD-LRU算法中存在的各种问题，避免干净页刚读入缓冲区就被替换的情况发生，防止冷脏页长期驻留缓冲区.

3　基于替换概率的缓冲区替换算法

本节首先论述PB-LRU算法的基本思想(3.1节)，然后给出算法的设计细节，并列举一个实例来演示PB-LRU算法的执行过程(3.2节)，最后将对PB-LRU算法与其它算法进行比较(3.3节).

3.1　基本思想

PB-LRU算法采用与LRU-WSR算法类似的冷数据判断机制，将缓冲区中的数据页依据其访问特点，划分为冷干净页、冷脏页、热干净页和热脏页.在这种冷数据检测方法中，缓冲区中的每个数据页都被赋予一个冷标识位，当冷标识位为1时，表明该数据页为冷页，当冷标识位为0时，表明该数据页为热页.此外，当缓冲区中的冷页被再次访问时，需要将其标记为热数据，即将冷标识位设置为0.由于闪存的写操作的代价高于读操作的代价，因此在4种数据页中，热脏页替换代价最高，冷干净页替换代价最低，热干净页的替换代价要高于冷脏页的替换代价.基于以上分析，可以得到4种数据页替换代价的关系表达式：

$$C_{cc}<C_{cd}<C_{hc}<C_{hd} \tag{1}$$

其中，C_{cc}表示冷干净页替换代价，C_{cd}表示冷脏页替换代价，C_{hc}表示热干净页的替换代价，C_{hd}表示热脏页的替换代价.

从式(1)可知，若想提高闪存的整体I/O性能，必须尽可能减少替换出热脏页和热干净页.为了达到这个目的，替换算法应该对冷页和热页进行区分，从而让热页在缓冲区驻留更长的时间.然而，一些已有的替换算法(比如CFLRU和LRU-WSR)都没有考虑数据的访问频度，不对冷页和热页做任何区分，因此不能获得较好的性能.另外一些替换算法(比如CCF-LRU和AD-LRU)，虽然考虑了数据的访问频度，但是它们都无法避免“干净页刚读入缓冲区就立刻被选择作为驱逐页”的情况，也无法避免冷脏页长期驻留缓冲区的情况，因此降低了缓冲区的命中率，也降低了闪存的整体I/O性能.

我们提出的基于替换概率的缓冲区替换算法PB-LRU，可以有效克服上述缺陷，其主要思想如下：

(1)将缓冲区分为冷区和热区，冷、热区的大小是固定的，热区中存放至少访问过两次的页，冷区中存放只访问过一次的页或已经很久没有再访问的热页；

(2)所有替换操作都发生在冷区，采用较大的概率替换冷区中的干净页，较小的概率替换冷区中的脏页，以此来避免冷脏页长期驻留缓冲区的情况；

(3)刚读入缓冲区的数据页存放在冷区的MRU位置，冷区中的一个页命中时，将该页转移到热区的MRU位置，并使用LRU-WSR算法选中热区中的一个页，转移到冷区的MRU位置；如果热区中的某个页被命中，则将其转移到热区的MRU位置；

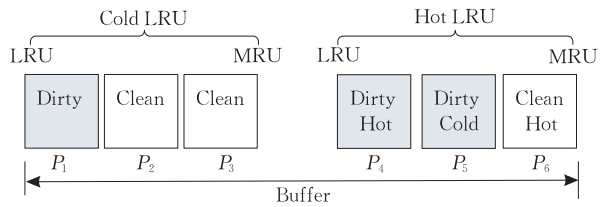
(4)使用LRU-WSR选择热区中的一个页时，需要使用到冷标识位来判断一个页属于冷页还是热页，因此PB-LRU算法会为热区中的每个数据页设置冷标识位(冷区中的数据页不需要冷标识位).标识位为“0”表示该页是热页，标识位为“1”表示该页为冷页.使用LRU-WSR算法将热区中的数据页转移到冷区时，首先获取热区中LRU位置的数据页，如果该页是干净页或冷标识位为1的脏页，就直接将该页转移到冷区的MRU位置；如果该页是冷标识位为0的脏页，则给该页第2次机会，将该页的冷标识位设置为1，并转移到热区的MRU位置，然后再次选择热区中LRU位置的数据页，重复上述判断过程.

为了更好地理解PB-LRU算法的思想，例1演示了该算法命中和脱靶的情况.

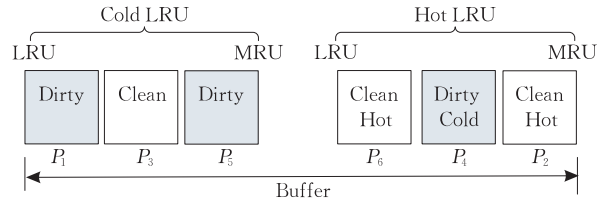
例1.　假设缓冲区最多只能同时容纳6个页.

在初始阶段(见图 1(a)),缓冲区中包含了 P_1,P_2,P_3,P_4,P_5,P_6,其中,P_1 是冷脏页,P_2 和 P_3 是冷干净页,P_4 是冷标识位为 0 的热脏页,P_5 是冷标识位为 1 的热脏页,P_6 是热干净页.

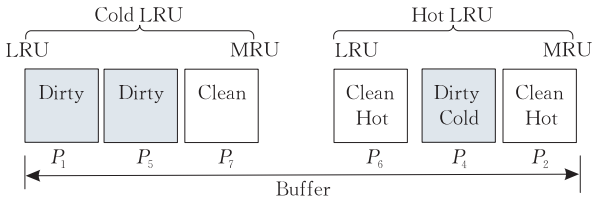
命中的情况. 在某个时刻,一个针对页 P_2 的读操作到达,PB-LRU 算法检查缓冲区后发现 P_2 已经在缓冲区中(在冷区中),就直接从缓冲区读取 P_2;P_2 被访问后,PB-LRU 算法会将其转移到热区的 MRU 位置,并将其冷标识位设置为 0. 由于冷、热区的大小是固定的,而此时热区已满,为了将冷区中的 P_2 转移到热区的 MRU 位置,需要使用 LRU-WSR 算法将热区中的一个页转移到冷区的 MRU 位置,从而在热区中腾出空间存放 P_2,P_2 被转移到热区以后,冷区就腾出了一个页的空间,此时,可以把热区中被转移出来的页,存放到冷区的 MRU 位置. 使用 LRU-WSR 算法在热区选择一个被转移页的方法是:首先,获取热区中 LRU 位置的页 P_4,由于 P_4 是冷标识位为 0 的脏页,说明 P_4 是热脏页,不会被马上转移,而是给它第 2 次机会,因此将 P_4 的冷标识位设置为 1,并将 P_4 转移到热区的 MRU 位置;接下来,再次获取热区中 LRU 位置的数据页 P_5 作判断,因为 P_5 是冷标识位为 1 的脏页,所以 LRU-WSR 算法不会给它第 2 次机会,直接将 P_5 确定为转移页. 采用上述方法确定 P_5 为转移页后,就可以将 P_5 转移以腾出一个页空间,从而将冷区中的 P_2 转移到热区的 MRU 位置,最后,把 P_5 放入冷区的 MRU 位置. 上述过程完成以后,缓冲区的当前状态如图 1(b)所示.



(a) PB-LRU算法的初始状态



(b) 将最近访问的页转移到MRU位置后的状态



(c) 驱逐一个页并读入一个页以后的状态

图 1 PB-LRU 算法的一个实例

脱靶的情况. 在上面的针对 P_2 读操作完成以后,一个新的针对页 P_7 的读操作到达,PB-LRU 算法检查缓冲区后发现 P_7 不在缓冲区中,需要到外部存储中读取 P_7 放入冷区. 但是,此时缓冲区已满,需要替换冷区中的一个页以腾出空间存放 P_7. PB-LRU 算法在选择驱逐页时,以较大的概率选择干净页,以较小的概率选择脏页,因此在大多数情况下,PB-LRU 算法将会选择冷干净页 P_3 为驱逐页. 然后从外部存储中读取 P_7 放入到冷区的 MRU 位置. 上述过程完成以后,缓冲区的当前状态如图 1(c)所示.

3.2 算法设计

算法 1. PB-LRU 算法.

输入:当前请求页(r),冷队列(L_c),热队列(L_h)

输出:请求页

```
1.  IF p is in L_h THEN   //如果 p 在热队列
2.    move p to the MRU position of L_h;
3.    clean cold-flag of p;   //清除 p 的冷标识位
4.    RETURN a reference of p in L_h;
5.  ELSE IF p is in L_c THEN //如果 p 在冷队列
6.    IF there is not free space in the L_h THEN
                          //将 p 从热队列转移到冷队列
7.      move a page to the MRU position of L_c from
        L_h with LRU-WSR algorithm;
8.    move p to the MRU position of L_h;
9.    RETURN a reference of p;
10. ELSE //如果 p 不在缓冲区中
11.   IF there is free space in L_c THEN
12.         insert p into MRU position of L_c;
13.         RETURN a reference to p of L_c;
14.   ELSE IF there is free space in L_h THEN
15.         insert p into MRU position of L_h;
16.         RETURN a reference to p of L_h;
17.   ELSE
18.         victim ← SelectVictim(L_c);
19.         IF victim is dirty THEN
20.           write page p to flash memory;
21.         insert p into MRU position of L_c;
22.   RETURN a reference to p in L_c.
```

PB-LRU 算法的具体执行过程如算法 1 所示. 如果热区中的某个页被命中,就将该页转移到热区的 MRU 位置,并清除它的冷标识位(第 1～4 行). 如果冷区中的某个页被命中,就将该页移到热区的

MRU 位置，如果此时热区已满，则使用 LRU-WSR 算法将热区中的一页转移到冷区的 MRU 位置，然后再将命中的数据页转移到热区的 MRU 位置（第 5～9 行）. 如果访问的数据页不在缓冲区中，则我们需要判断缓冲区是否已满，如果缓冲区未满，则直接读入数据页到缓冲区中（第 11～16 行），如果缓冲区已满，则需要调用 *SelectVictim* 函数，从冷区中选择一个驱逐页，替换出缓冲区（第 18 行）. 驱逐一页时，还需要判断该页是否脏页，如果是脏页，还需要回写到外部存储（第 19～20 行）.

SelectVictim 函数用来在冷区中选择一个驱逐页，该函数采用不同的概率替换干净页和脏页，以较大的概率替换干净页，以较小的概率替换脏页. 为了达到以不同概率替换干净页和脏页的目的，*SelectVictim* 首先会生成一个 0～N 的随机数，其中，N 是闪存读代价（$Cost_{\text{read}}$）与闪存写代价（$Cost_{\text{write}}$）之和，然后判断生成的随机数落入[0，$Cost_{\text{read}}$）区间还是[$Cost_{\text{read}}$，N）区间. 由于闪存的读代价小于闪存的写代价，即 $Cost_{\text{read}}<Cost_{\text{write}}$，这意味着随机数落入[$Cost_{\text{read}}$，$N$）区间的概率更大. 由此，当随机数落入[$Cost_{\text{read}}$，$N$）区间时，令变量 *replacePage* 为 0，用以表示算法本次将选择干净页为驱逐页；当随机数落入[0，$Cost_{\text{read}}$）区间时，令变量 *replacePage* 为 1，用以表示算法本次将选择脏页为驱逐页. 这就实现了以较大的概率替换干净页和以较小的概率替换脏页的目的.

SelectVictim 函数通过上面的方法确定需要被替换的页后，如果确定替换干净页，则还需要判断冷区中是否存在干净页，如果不存在，则替换脏页；如果确定替换脏页，则需要判断冷区中是否存在脏页，如果不存在，则替换干净页.

算法 2.　*SelectVictim* 函数.

输入：冷队列（L_c）

输出：驱逐页

1. *replacePage*←rand()%($Cost_{\text{read}}+Cost_{\text{write}}$)<$Cost_{\text{read}}$?1：0；

//*replacePage* 决定了替换干净页还是替换脏页

2. IF (*replacePage*==0 AND there exists clean page in L_c) OR (there is not dirty page in L_c) THEN
3.　　select a clean page from L_c as *victim*；
4. ELSE
5.　　select a dirty page from L_c as *victim*；
6. remove the *victim* page from L_c；
7. RETURN a reference to the *victim* page.

3.3　与其它算法的比较

下面通过两个实例来阐释 PB-LRU 算法相对于已有的其它算法的优点.

例 2.　假设缓冲区最多只能同时容纳 6 个页，并且缓冲区已满，缓冲区中当前包含的数据页（图 2 所示）包括 P_1，P_2，P_3，P_4，P_5，P_6，其中，P_1 和 P_6 是冷脏页，P_2 和 P_3 是热干净页，P_4 是冷干净页，P_5 是热脏页. 假设此时一个访问 P_7 的请求到达，由于缓冲区已满，所以需要将缓冲区中的一个页驱逐出去，以腾出空间存放 P_7. 在选择驱逐页时，PB-LRU 算法和 AD-LRU 算法都选择 P_4 作为驱逐页（如图 2 所示）；而对于 CFLRU 和 LRU-WSR 算法而言，如图 3 所示，前者则会选择热干净页 P_2 作为驱逐页，后者会选择冷脏页 P_1 作为驱逐页. 根据前面第 3.1 节的分析，一个好的缓冲区替换算法应该优先替换冷干净页，在不存在冷干净页时，再替换冷脏页，并尽可能让热数据页驻留内存. 根据这个判断标准，在本例中，数据页被替换的先后顺序依次是 P_4，P_1，P_6，P_2，P_3 和 P_5，即 P_4 应该最先被替换，P_5 应该最后被替换. 由此可以看出，CFLRU 做出了最差的选择，而 PB-LRU 算法和 AD-LRU 算法做出了最优的选择.

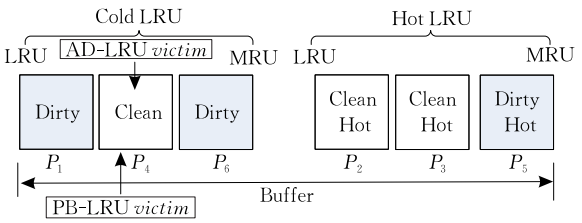


图 2　AD-LRU 和 PB-LRU 算法选择驱逐页实例

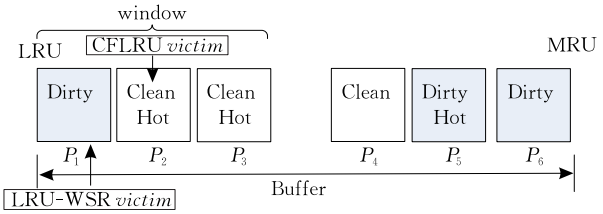


图 3　CFLRU 和 LRU-WSR 算法选择驱逐页实例

通过例 2 可以看出，没有考虑数据访问频度的替换算法（如 CFLRU 和 LRU-WSR）选择驱逐页时，可能选择热数据页为驱逐页，所以降低了缓冲区的命中率，而考虑了数据访问频度的替换算法（如 AD-LRU 和 PB-LRU）总是会优先替换冷的数据页，所以总体表现比没有考虑数据访问频度的算法更好. AD-LRU 算法虽然考虑了数据的访问频度，但是无法避免“干净页刚读入缓冲区就立刻被选择作为驱逐页”的情况，从而导致缓冲区的脱靶率的增加，而 PB-LRU 算法，通过使用基于概率的替换机

制，不仅能够有效地避免了干净页刚读入缓冲区就被选为驱逐页的情况，而且还能避免冷脏页长期驻留内存，从而充分利用了宝贵的缓冲区资源，显著提升了缓冲区命中率. 下面将通过例 3 来演示 PB-LRU 算法相对于 AD-LRU 算法的优势.

例 3. 在例 2 中，AD-LRU 和 PB-LRU 算法都选择 P_4 为驱逐页，然后读入 P_7，此时，缓冲区中的数据是 P_1，P_2，P_3，P_5，P_6，P_7，其中，P_1 和 P_6 是冷脏页，P_2 和 P_3 是热干净页，P_5 是热脏页，P_7 是冷干净页. 假设现在有一个新的请求访问 P_8，由于此时 P_8 不在缓冲区中，因此会发生缓存脱靶. 此时，AD-LRU 算法会选择刚进入缓冲区的数据页 P_7 作为驱逐页，然后将 P_8 存放到冷区的 MRU 位置. 假设接下来又有一个新的请求访问 P_7，此时，P_7 不在缓存中，AD-LRU 算法又会出现缓存脱靶的情况，这时 AD-LRU 算法会选择 P_8 作为驱逐页，并将 P_7 存放到冷区的 MRU 位置.

图 4 显示了当采用 AD-LRU 算法时，在经过上述对 P_7，P_8，P_7 的访问以后缓冲区的当前状态. 可以看出，在上述过程中，AD-LRU 算法共出现了 3 次脱靶的情况，即第 1 次访问 P_7 时，发生一次脱靶，访问 P_8 时，发生一次脱靶，第 2 次访问 P_7 时，发生一次脱靶. 相比较而言，我们的 PB-LRU 算法在整个过程中只会出现两次脱靶的情况. 在第 1 次请求 P_7 时，此时 P_7 不在缓冲区中，发生第 1 次脱靶，由于缓冲区已满，PB-LRU 算法会选择 P_4 作为驱逐页，并将 P_7 存放到冷区的 MRU 位置. 当一个新的请求访问 P_8 时，由于 P_8 不在缓冲区中，此时发生第 2 次脱靶，与 AD-LRU 算法不同的是，PB-LRU 此时会选择冷脏页 P_6 作为驱逐页，而不会把刚刚进入缓存的 P_7 作为驱逐页. 当第 2 次访问 P_7 时，P_7 命中并成为热页，算法将 P_7 转移到热区的 MRU 位置，并将热区中的 P_2 转移到冷区的 MRU 位置.

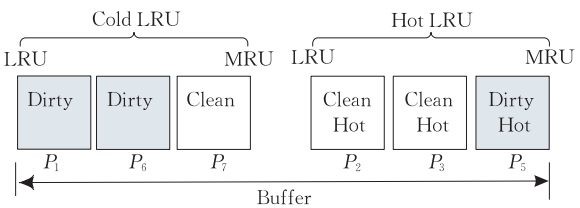


图 4 AD-LRU 算法替换实例

图 5 显示了采用 PB-LRU 算法时，在经过上述对 P_7，P_8，P_7 的访问以后缓冲区的当前状态. 可以看出，在上述过程中，PB-LRU 算法只发生了两次脱靶的情况，性能要优于 AD-LRU 算法.

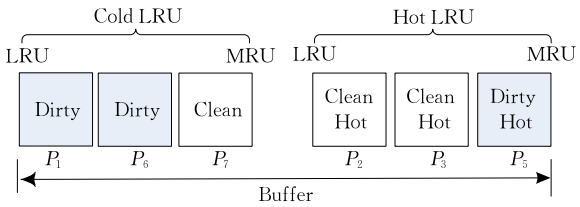


图 5 PB-LRU 算法替换实例

4 自适应的基于概率的缓冲区替换算法

PB-LRU 算法是一种朴素算法，它虽然通过基于概率的替换机制，解决了 CCF-LRU 和 AD-LRU 算法中存在的“干净页刚读入缓冲区就被选择为驱逐页”的情况，但是，PB-LRU 算法也存在如下缺陷：(1) 寻找干净页的开销较大，该算法在冷区中替换干净页时，必须反向搜索冷区的 LRU 链表来获得一个干净页，当 LRU 链表较长时，时间开销就会比较大；(2) 冷、热区的大小固定，难以适应不同访问模式的工作负载.

因此，我们提出了针对 PB-LRU 的改进算法——自适应的 PB-LRU 算法，简称 APB-LRU (Adaptive PB-LRU). APB-LRU 针对 PB-LRU 的改进主要体现在两个方面：(1) PB-LRU 在冷区只有一个 LRU 链表，与 PB-LRU 不同的是，APB-LRU 将冷区分为两个链表，即干净页链表和脏页链表，替换干净页时直接获取干净页链表中 LRU 位置的页，避免了 PB-LRU 算法中反向搜索冷区 LRU 链表的开销，从而加快了驱逐页的查找速度；(2) APB-LRU 能够动态调整冷、热区大小比例，从而可以适应不同访问模式的工作负载.

本节内容首先介绍 APB-LRU 算法的基本思想(4.1 节)，然后详细阐述算法的设计(4.2 节)，最后分析了算法的时间复杂度(4.3 节).

4.1 改进策略

PB-LRU 算法存在两个问题：(1) 反向搜索冷区的 LRU 链表获取干净页的时间开销较大；(2) 冷、热区大小固定，难以适应不同访问模式的工作负载. 针对这两个问题，我们提出了 PB-LRU 算法的改进算法，即 APB-LRU.

针对第 1 个问题，APB-LRU 算法将冷区分成两个链表，即干净页链表和脏页链表，分别存储干净页和脏页. 当将热区中的一个数据页转移到冷区时，需要先判断该页是脏页还是干净页，如果是干净页，

就转移到冷区的干净页链表中，如果是脏页就转移到冷区的脏页链表中. 当发生脱靶要选择一个驱逐页时，需要先通过概率机制来确定替换干净页还是替换脏页，如果确定替换干净页(脏页)，就直接替换处于干净页链表(脏页链表)的 LRU 位置的页，从而从根本上避免了反向搜索 LRU 链表的开销，降低了算法的时间复杂度.

针对第 2 个问题，我们通过大量实验观察发现：当数据访问的局部性较高时，让热区在缓冲区中占据更大的比例，可以使得 PB-LRU 算法获得更好的性能；当数据访问局部性较低时，让热区在缓冲区中所占比例适当减少，可以使得 PB-LRU 算法获得更好的性能. 由此，我们在 APB-LRU 算法中设计了冷、热区比例动态变化机制，可以根据工作负载的变化动态调整冷、热区所占缓冲区的比例. 冷、热区所占缓冲区比例动态调整的方法如下：事先为冷区和热区设置下界，当冷区中的页命中，就将命中的数据页转移到热区，此时热区的容量增加，冷区的容量减少；当冷区的容量达到了下界，就开始扩展冷区，即将热区中的部分数据页转移到冷区，使得冷区的容量增加，热区的容量减少，如果热区的容量持续减少，最后达到热区的下界，则停止此次扩展过程.

上面扩展冷区的过程，与 PB-LRU 算法中热区中的数据转移到冷区的过程类似，具体如下：首先，获取热区中 LRU 位置的数据页，如果该页是干净页，就直接将该页转移到冷区的 MRU 位置，如果该页是冷标识位为 0 的脏页，则给该页第 2 次机会，将该页的冷标识位设置为 1，并转移到热区的 MRU 位置；然后，再次选择热区中 LRU 位置的数据页，如果该页是干净页，或者是冷标识位为 0 的脏页，就重复上述过程，否则，停止扩展.

4.2　算法设计

APB-LRU 算法细节如算法 3 所示. 在 APB-LRU 算法中，首先判断冷区容量是否达到下界(第 1 行)，如果已经达到下界，则扩展冷区的容量(第 2～11 行). 在选择驱逐页时，与 PB-LRU 算法一样，首先生成一个 0～N 的随机数，然后根据该随机数落入的区间确定替换干净页还是脏页(第 12 行)，在确定替换脏页或是干净页以后，在相应的链表中进行替换(第 13～16 行).

算法 3.　APB-LRU 算法的伪代码.

输入：当前请求页(r)，冷脏页队列(L_{cd})，冷干净页队列(L_{cc})，热队列(L_h)

输出：请求页

```
1.  IF Lcc.size+Lcd.size<LowerBoundOfColdRegion
    THEN
2.    WHILE Lh.size>LowerBoundOfHotRegion DO
3.      p←the LRU page in Lh;
4.      IF cold flag of p is set to 1 THEN
5.        move p to the MRU position of Lcc if p is a
          clean page, or move p to MRU position of
          Lcd if p is a dirty page;
6.        BREAK;
7.      ELSE IF p is clean page THEN
8.        move p to the MRU position of Lcc if p is a
          clean page, or move p to MRU position of
          Lcd if p is a dirty page;
9.      ELSE
10.       set the cold flag of victim;
11.       move victim to the MRU position of Lh;
12. replacePage←rand()% (Costread+Costwrite)<
                    Costread?1:0;
//replacePage 决定替换一个干净页还是替换一个脏页
13. IF (replacePage==0 AND there exists clean page
    in Lcc) OR (there is not dirty page in Lcd) THEN
14.   select a clean page as victim in Lcc;
15. ELSE
16.   select a dirty page from Lcd as victim;
17. remove victim from Lcc or Lcd;
18. RETURN reference to victim.
```

4.3　算法时间复杂度分析

本文的 PB-LRU 算法每次选择驱逐页时都需要反向搜索 LRU 链表，搜索驱逐页的时间复杂度为 $O(n)$，当冷区中的数据页命中时，PB-LRU 算法需要使用 LRU-WSR 算法将热区中的一个数据页转移到冷区，因为 LRU-WSR 算法的时间复杂度为 $O(n)$，所以该转移操作的时间复杂度也为 $O(n)$. 本文的改进算法 APB-LRU 对 PB-LRU 算法进行了改进，不需要反向搜索 LRU 链表来获得驱逐页，而是直接选取处于脏页链表或干净页链表的 LRU 位置的页作为驱逐页，所以，选择驱逐页的代价为 $O(1)$.

APB-LRU 算法除了选择驱逐页的开销以外，还存在扩展冷区的开销，即将热区中的数据转移到冷区的开销. 由于 APB-LRU 算法使用 LRU-WSR 算法扩展冷区，所以扩展操作的时间复杂度为 $O(n)$. 但是，扩展操作只有在冷区容量达到下界时才会发生，假设上一次扩展冷区后，冷区的容量比下界大 N，那么只有在冷区中的数据页命中 N 次后(每次命中都会把冷区中命中页转移到热区，使得冷

区的容量减少),冷区的容量才会再次达到下界,并触发扩展操作,所以 APB-LRU 算法的平均时间复杂度仍为 $O(1)$.

5 实验设计与结果

本节首先介绍了实验的设计(5.1 节);然后通过实验得到了 APB-LRU 获得最优性能时的参数配置(5.2 节);最后,对 LRU、CFLRU、LRU-WSR、CCF-LRU、AD-LRU 和本文的 APB-LRU 算法进行全面详尽的性能比较(5.3~5.6 节).

5.1 实验设计

实验使用 Flash-DBSim[13]① 平台模拟闪存存储系统. Flash-DBSim 是一种高效的、可重用和可配置的闪存存储系统仿真平台,可以根据上层应用的需要模拟出不同特性的固态盘,从而可以方便地为上层应用提供测试环境,很多已有的研究(比如文献[11-12])都采用了该模拟器进行算法性能的比较.

我们采用与文献[11]一样的参数配置模拟器,即模拟一个 128MB 的 NAND 闪存固态盘,该固态盘所采用的 NAND 闪存的数据页大小是 2 KB,每个数据块包含 64 个数据页,详细设备参数请见表 1.

表 1 NAND 闪存的特性参数

参数	值
页容量	2048 B
块容量	64 pages
读代价	25 μs/page
写代价	200 μs/page
擦除代价	1.5 ms/block
擦除次数	100000

在进行算法性能比较时,各个缓冲区替换算法的命中率与算法自身的参数设置密切相关. 本文中,CFLRU 中的参数 w 取值为 0.5(即采用文献[12]在比较 AD-LRU 和 CFLRU 时的 w 值),AD-LRU 算法中的 min_lc 取值为 0.1(与文献[12]的取值相同). 在具体实现时,对于区分冷页和热页的算法(LRU-WSR、CCF-LRU 和 AD-LRU),通过在相关数据结构中添加一个标识位,用于标识该页是热页还是冷页;对于将缓冲区分为多个区的算法(CCF-LRU 和 AD-LRU),在缓冲区链表中插入一个哑节点(dummy node),用以区分冷区和热区. Flash-DBSim 通过统计闪存的读次数、写次数和擦除次数以及这些操作所需要的时间,得到总的运行时间,实现了算法间公平的比较. 每个算法的具体实现,可以参考相关论文.

通过实验发现,在 PB-LRU 算法中,对于所有的测试数据集,热区所占比例多于 0.99 时(即冷区所占比例小于 0.01),算法性能会明显下降,因此 APB-LRU 算法将冷区的下界设置为 0.01;同时,对于所有的测试数据集,热区所占比例少于 0.8 时,算法的整体性能也会明显降低,所以 APB-LRU 算法将热区的下界设置为 0.8.

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[14]所用的测试方式进行测试,生成了 4 种符合 Zipf 分布的测试数据,其统计信息见表 2. 其中"读/写比例"这一列中的"$x\%/y\%$"表示,对于某种测试数据集而言,读操作占所有操作的 $x\%$,写操作占所有操作的 $y\%$;"局部性"列中的"$x\%/y\%$"表示对某种测试数据而言,$x\%$的读写操作集中在 $y\%$的数据页上[15-16]. 我们采用以下标准来评价缓冲区替换算法的性能:(1)命中率;(2)物理读操作(读闪存)的次数;(3)物理写操作(写闪存)的次数;(4)运行时间. 这里不对擦除操作的次数进行比较,因为擦除操作总是与写操作成正比,通过比较各算法写操作的次数就可以反映出各个算法擦除操作的情况.

表 2 测试数据集的统计信息

数据集	请求次数	读写比例	局部性
T_1	3000000	90%/10%	60%/40%
T_2	3000000	30%/70%	70%/30%
T_3	3000000	60%/40%	60%/40%
T_4	3000000	80%/20%	80%/20%

5.2 APB-LRU 算法最佳替换概率的选择

本实验测试不同的替换概率对 APB-LRU 算法性能的影响,从而用来确定算法获得最优性能时的替换概率. 我们设计了 4 种不同的情形(即情形 A、B、C 和 D),表 3 给出了每种情形下的干净页和脏页的替换概率.

表 3 4 种不同的情形

情形	替换比例	描述
A	1∶1	同等概率替换干净页和脏页
B	写代价∶读代价	干净页与脏页的替换概率与替换它们的代价成反比
C	(写代价+擦除代价)∶读代价	以更大的概率替换干净页
D	1∶0	总是优先替换干净页

① Flash-DBSim. http://kdelab.ustc.edu.cn/flash-dbsim/index_en.html, retrieved, May 2009

图 6 展示了缓冲区容量为 3 MB 时，4 个测试数据集在不同情形下的运行时间. 可以看出，对于 4 个测试数据集而言，采用情形 C 的替换概率设置时，APB-LRU 算法在其中的 3 个数据集（即 T_1、T_2 和 T_3）都表现出了较好的性能，其中，在数据集 T_1 和 T_2 中，算法在情形 C 下表现最优，在数据集 T_3 中，算法在情形 C 下表现次优. 相反，采用其它情形（即情形 A、B 和 D）的概率设置时，算法都不能很好地适应不同的数据集. 因此，本文采用情形 C 的替换概率来运行 APB-LRU 算法.

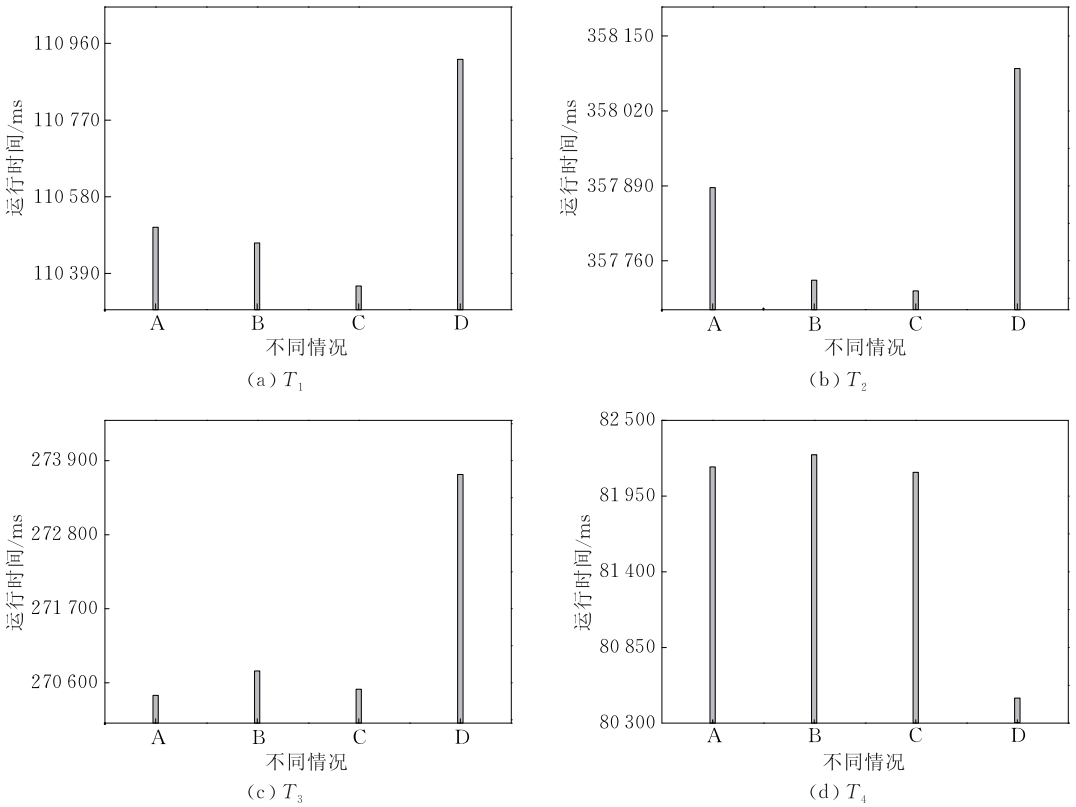


图 6　不同替换概率对性能的影响

需要强调指出的是，本文后面所有的实验中，APB-LRU 算法都采用了情形 C 的替换概率设置.

5.3　命中率

图 7 展示了各个缓冲区替换算法在运行不同的测试数据集时的命中率情况. 从图中可以看出，在不同的测试数据集和不同的缓冲区容量下，充分考虑数据访问频度的算法（CCF-LRU、AD-LRU、PB-LRU 和 APB-LRU），其命中率明显高于没有考虑数据访问频度的算法（LRU、CFLRU 和 LRU-WSR），而本文提出的 PB-LRU 算法命中率在大多数情况下都比 CCF-LRU 和 AD-LRU 高，这充分说明基于概率的替换机制是一种行之有效的方法. 本文提出的 APB-LRU 算法在所有测试数据集和缓冲区容量下，都取得了最好的命中率. 例如，在缓冲区容量为 5 MB，测试数据集为 T_2 时，没有考虑数据访问频度的算法命中率在 42%左右，考虑了数据访问频度的算法命中率达到了 53%，而本文提出的 APB-LRU 算法命中率为 58%，高于其它算法. 对于考虑了数据访问频度的算法，命中率高于其它算法是因为它们在选择驱逐页时，总是会优先选择冷页，使得热页可以在缓冲区中停留更长的时间，所以提高了命中率. APB-LRU 算法能够获得更好的命中率是因为它不仅充分考虑了数据访问的频度，而且使用了基于概率的替换机制，能够做到优先替换冷页，让热页有更多的时间驻留缓冲区，同时避免了“干净页刚读入缓冲区就被选择为驱逐页”和“冷脏页长期驻留缓冲区”的情况，充分利用了缓冲区资源，所以获得了较高的命中率.

5.4　物理读操作次数

图 8 展示了各个缓冲区替换算法在运行不同的测试数据集时的物理读操作次数. 从图中可以看出，考虑了数据访问频度的算法，其读操作次数远少于没有考虑数据访问频度的算法. 本文提出的基于概率的替换算法（PB-LRU 和 APB-LRU）的读操作次数也少于总是优先替换干净页的算法（CCF-LRU 和 AD-LRU）. 从结果中还可以看出，本文提出的改

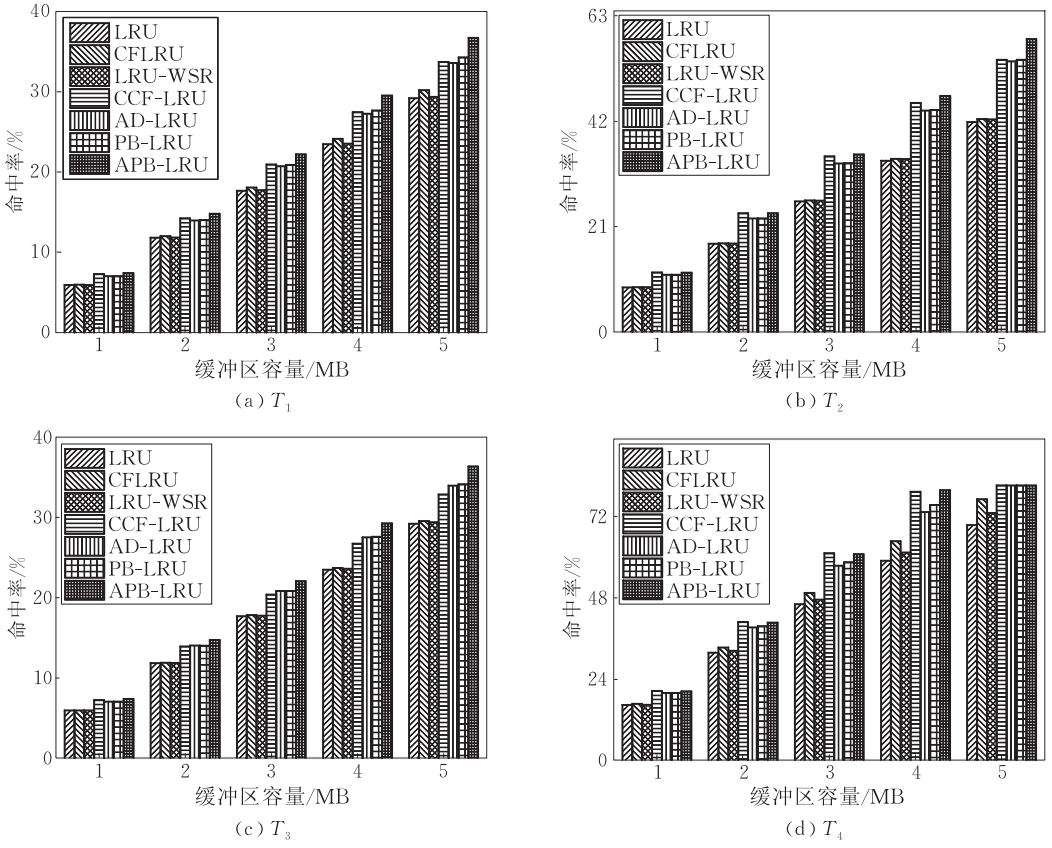


图 7 在不同测试数据集上的命中率比较

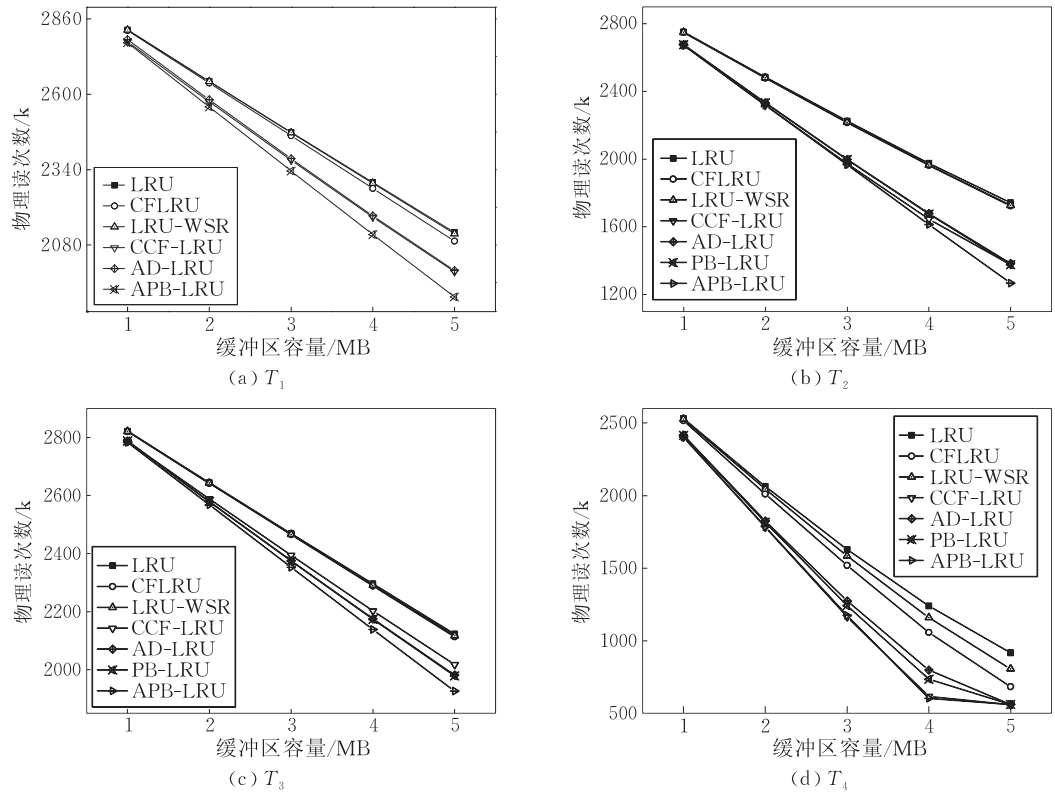


图 8 在不同测试数据集上的物理读操作比较

进算法(APB-LRU)总是优于朴素算法(PB-LRU). 本文提出的采用了自适应机制的 APB-LRU 算法不仅明显减少了物理读操作的次数,而且在所有测试数据集上都表现出了比较稳定的性能. 相反地,

AD-LRU 和 CCF-LRU 算法在不同的测试数据集上的性能表现不够稳定.例如,AD-LRU 在测试数据集 T_4 上的物理读操作次数明显多于 CCF-LRU 和 APB-LRU,在缓冲区容量为 4 MB,测试数据集为 T_4 时,CCF-LRU 和 APB-LRU 只需要 61 万次读操作,而 AD-LRU 需要 79 万次读操作.CCF-LRU 也存在表现不稳定的情况,例如在缓冲区容量为 4 MB,测试数据集为 T_3 时,APB-LRU 算法只需要 214 万次读操作,AD-LRU 需要 217 万次读操作,而 CCF-LRU 需要 220 万次读操作,多于 AD-LRU 和 APB-LRU.

5.5　物理写操作次数

图 9 展示了各个缓冲区替换算法在执行不同的测试数据集时物理写操作的次数.从图中可以看出,考虑了数据访问频度的算法(CCF-LRU、AD-LRU、PB-LRU 和 APB-LRU),其物理写操作的次数明显少于没有考虑数据访问频度的算法(LRU、CFLRU 和 LRU-WSR).此外,本文的 APB-LRU 算法在所有测试数据集上的物理写操作次数都比其它算法少.这是因为,虽然 APB-LRU 相对于其它算法(CF-LRU、CCF-LRU 和 AD-LRU)而言,脏页有更大的机会被选择为驱逐页,但是 APB-LRU 替换的是冷脏页,将冷脏页尽早替换出缓冲区,充分利用了宝贵的缓冲区资源,从而使得 APB-LRU 在提高命中率的同时,没有增加物理写操作的次数.

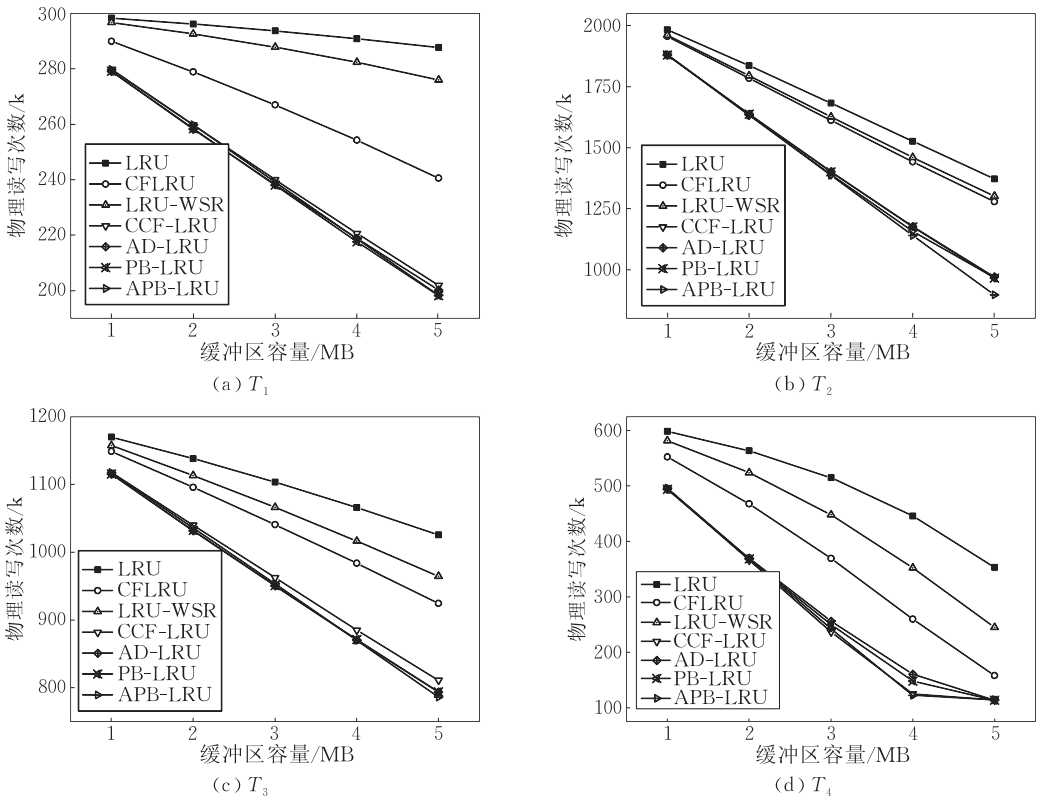


图 9　不同测试数据集上的物理写操作比较

5.6　运行时间

图 10 展示了各个缓冲区替换算在执行不同的测试数据集时的运行时间的比较,包括物理读操作时间、物理写操作时间和闪存中擦除操作的时间.本实验中采用了 1 种面向磁盘的缓冲区替换算法(LRU)和 6 种面向闪存的缓冲区替换算法(CFLRU、LRU-WSR、CCF-LRU、AD-LRU、PB-LRU 和 APB-LRU).从图中可以看出,面向闪存的缓冲区替换算法所需要的运行时间远少于面向磁盘的缓冲区替换算法 LRU,这是因为,面向闪存的缓冲区替换算法会赋予干净页更高的替换优先级,这样做虽然增加闪存读操作的次数,但是减少了闪存的写操作和擦除操作的次数,而闪存的读代价要小于写代价,因此可以获得整体性能的提升.在所有的面向闪存的缓冲区替换算法中,考虑了数据访问频度的算法(CCF-LRU、AD-LRU、PB-LRU 和 APB-LRU)的运行时间,明显少于没有考虑数据访问频度的算法(CFLRU 和 LRU-WSR),其中本文的 APB-LRU 算法性能表现最好.这是因为:(1)该算法使用了基于概率的替换机制,提高了命中率,减少了物理读操作的次数和物理写操作的次数;(2)该算法使用了冷、热区所占缓冲区比例动态调整机制,能够适应不同访问模式的工作负载.

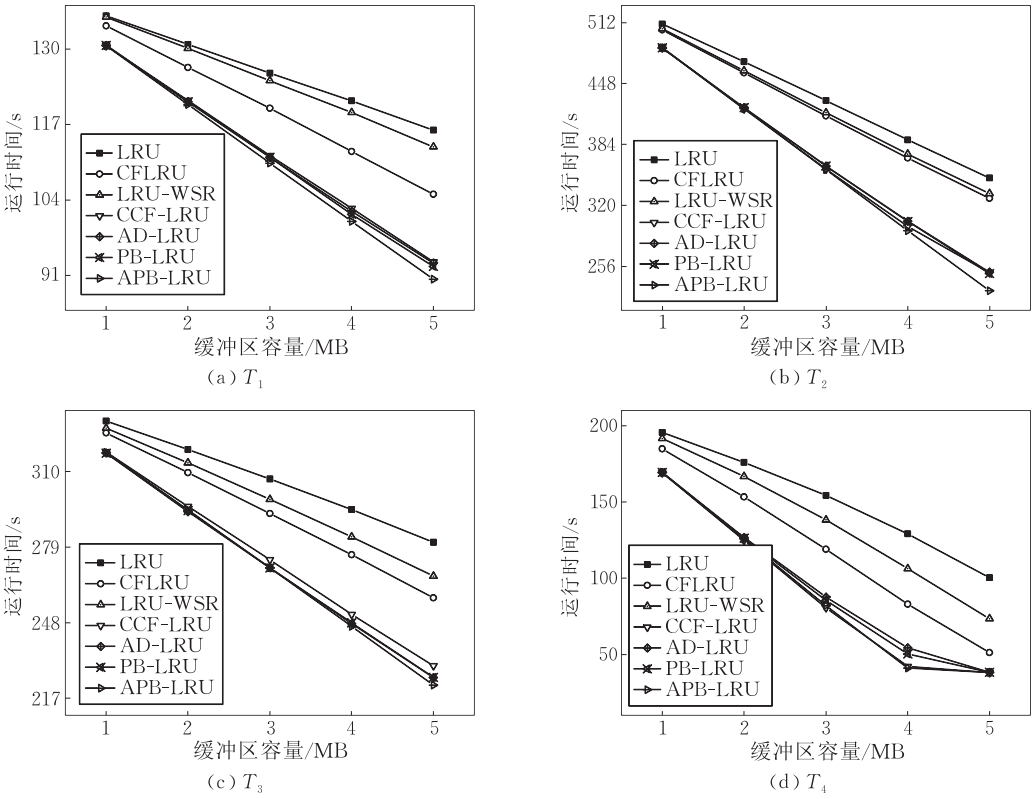


图 10 不同测试数据集上的运行时间比较

6 总 结

随着闪存技术的发展,闪存数据库系统会越来越普及.采用面向闪存的缓冲区替换算法可以有效提高闪存数据库系统的整体性能.已有的面向闪存数据库系统的缓冲区替换算法,或者没有充分考虑闪存的特性,或者没有充分利用数据访问的局部性等特征,因此还存在较大的性能提升空间.本文首先提出了一种朴素的缓冲区替换算法,即 PB-LRU 算法,该算法不仅考虑了闪存读写非对称的特性,而且充分利用了数据访问的频度特征,通过以较大的概率替换缓冲区中的干净页,以较小的概率替换缓冲区中的脏页,避免了冷脏页长期驻留缓冲区的情况,从而提高了命中率,获得了较好的整体性能.此外,我们通过大量实验发现,对于局部性较高的工作负载,应该赋予热区更大的空间,对于局部性较低的工作负载,热区所占缓冲区比例应该地相应减少,才能够获得更好的性能.因此,我们提出了针对朴素算法 PB-LRU 的改进算法 APB-LRU,它能够根据不同的工作负载模式,动态调整冷、热区大小,从而使得替换算法在不同的负载模式下都可以取得较好的性能.我们进行了大量的实验,实验结果显示 APB-LRU 算法具有比其它已有的算法更好的性能.

本文在研究过程中,和其它已有的替换算法(比如 CFLRU 和 AD-LRU)类似,采用了固定的闪存读写代价比.然而,不同的闪存设备的读写代价比是不一样的,我们将在未来的研究工作中,在不同闪存设备上充分测试 APB-LRU 算法的不同性能表现,从而为不同读写代价比的设备确定最优的参数组合.

参 考 文 献

[1] Chiang M L, Lee P C H, Chang R C. Managing flash memory in personal communication devices//Proceedings of the International Symposium on Consumer Electronics. Seattle, WA, USA, 1997: 177-182

[2] Jung Hoyong, Yoon Kyunghoon, Shim Hyoki, et al. LIRS-WSR. Integration of LIRS and writes sequence reordering for flash memory//Proceedings of the ICCSA. Kuala Lumpur, Malaysia, 2007: 224-237

[3] Kim Hyojun, Lee Ki Yong. A new transactional flash translation layer for embedded database systems based on MLC NAND flash memory//Proceedings of the International Conference on Consumer Electronics. Las Vegas, NV, 2008: 1-2

[4] On Sai Tung, Xu Jianliang, Choi Byron, et al. Flag commit: Supporting efficient transaction recovery in flash-based DBMSs. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(9): 1624-1639

[5] Nath Suman, Kansal Aman. FlashDB: Dynamic self-tuning database for NAND flash//Proceedings of the IPSN. Cambridge, Massachusetts, USA, 2007: 410-419

[6] O'Neil Elizabeth J, O'Neil Patrick E, Gerhard Weikum. The LRU-K page replacement algorithm for database disk buffering//Proceedings of the SIGMOD. Washington, USA, 1993: 297-306

[7] Gupta P, Bhattacharjee G P. An efficient algorithm for random sampling without replacement//Proceedings of the FSTTCS. Bangalore, India, 1984: 435-465

[8] Altman E R, Agarwal V K, Gao G R. A novel methodology using genetic algorithms for the design of caches and cache replacement policy//Proceedings of the ICGA. Urbana Champaign, IL, USA, 1993: 392-399

[9] Li Zhi, Jin Peiquan, Su Xuan, et al. CCF-LRU: A new buffer replacement algorithm for flash memory. IEEE Transactions on Consumer Electronics, 2009, 55(3): 1351-1359

[10] Jin Peiquan, Ou Yi, Härder Theo, Li Zhi. AD-LRU: An efficient buffer replacement algorithm for flash-based databases. Data & Knowledge Engineering, 2012, 72: 73-102

[11] Park Seon-Yeong, Jung Dawoon, Kang Jeong-Uk, et al. CFLRU: A replacement algorithm for flash memory//Proceedings of the CASES'06. Seoul, Korea, 2006: 234-241

[12] Jung H, Shim H, Park S, et al. LRU-WSR: Integration of LRU and writes sequence reordering for flash memory. IEEE Transactions on Consumer Electronics, 2007, 54(3): 1215-1223

[13] Su X, Jin P, Xiang X, et al. Flash-DBSim: A simulation tool for evaluating flash-based database algorithm//Proceedings of the ICCSIT'09. Beijing, China, 2009: 185-189

[14] Johnson Theodore, Shasha Dennis. 2Q: A low overhead high performance buffer management replacement algorithm//Proceedings of the VLDB'94. Santiago de Chile, Chile, 1994: 439-450

[15] Tang X, Meng X F. ACR: An adaptive cost-aware buffer replacement algorithm for flash storage devices//Proceedings of the MDM2010. Kansas City, Missouri, USA, 2010: 33-42

[16] Tang Xian, Meng Xiao-Feng, Liang Zhi-Chao, Lu Ze-Ping. Cost-based buffer management algorithm for flash database systems. Journal of Software, 2011, 22(12): 2951-2964(in Chinese)
(汤显，孟小峰，梁智超，卢泽萍. 基于代价的闪存数据库缓冲区置换算法. 软件学报，2011，22(12)：2951-2964)

LIN Zi-Yu, born in 1978, Ph. D., assistant professor. His main research interests include database, data warehouse, data mining, etc.

LAI Ming-Xing, born in 1989, M. S. candidate. His main research interest is flash-based database.

ZOU Quan, born in 1982, Ph. D., assistant professor. His main research interests include bioinformatics and data mining.

XUE Yong-Sheng, born in 1946, professor. His main research interest is database theory.

YANG Si-Ying, born in 1992, undergraduate student. Her main research interest is flash-based database.

Background

With the development of real-time database and real-time data warehouse, flash-based database system may play a more and more important role in enterprise information system. Flash-based databases are able to provide higher performance than traditional hard-disk-based databases, because flash can achieve much faster speed in both read and write operations than hard disk. We have studied flash-based database system since 2009, and focused on buffer replacement policy, storage management, indexing, query processing, transaction management, etc. On the aspect of buffer replacement policy, there has been much existing research work, but they all have some deficiencies. First, they all select clean pages in the buffer as the victim when replacement occurs, which may lead to that the buffer is totally occupied by dirty pages in the end. Second, cold dirty pages will stay in the buffer for a long time, which wastes the delicious resource of buffer. With the proposed method in this work, the above problem can be effectively resolved. This work is partly supported by the Fundamental Research Funds for the Central Universities (Nos. 2011121049 and 2012121030), the Natural Science Foundation of China (Nos. 61001013, 61102136, 61202012), and the Natural Science Foundation of Fujian Province (Nos. 2011J05156, 2011J05158, 2013J05099).