一种对 LRFU 置换策略的自适应改进

李占胜[1],毕会娟[1],李艳平[2],张立松[1]
LI Zhan-sheng[1],BI Hui-juan[1],LI Yan-ping[2],ZHANG Li-song[1]

1.华北计算技术研究所 软件平台研究室,北京 100083
2.北京邮电大学 计算机科学与技术学院,北京 100876
1.Department of Software Platform,North China Institute of Computing Technology,Beijing 100083,China
2.Department of Computer Science and Technology,Beijing University of Posts and Telecommunications,Beijing 100876,China
E-mail:diann10101@126.com

LI Zhan-sheng,BI Hui-juan,LI Yan-ping,et al.Improved LRFU algorithm with adaptive replacement.Computer Engineering and Applications,2008,44(17):153-157.

Abstract: The database buffer page replacement algorithm has an important impact in the performance of the database system. Caching is one of the most fundamental metaphor in modern computing,Cache replacement policies play a crucial part in different aspects of today's high performance computing environments.It studies some of the page replacement algorithms,including the recency based algorithms,the frequency based algorithms and the adaptive algorithms.The LRFU algorithms isn't a self-tuning algorithm,although it combines the recency and frequency.This paper proposes an improved LRFU algorithm,that can adaptively and dynamically modify the λ value of LRFU.Conducting simulations with a variety of file access patterns and a wide range of buffer size,it shows that the improved LRFU algorithm outperforms a LRFU algorithm in many cases.
Key words: database buffer;page replacement algorithm;LRFU;self-tuning;improved LRFU

摘 要:数据库缓冲区页面置换算法对磁盘数据库的性能有着重要的影响,页面置换算法主要有基于访问时间的置换策略、基于访问次数的置换策略、两者结合的置换策略等。LRFU 算法是一系列结合 LRU 和 LFU 置换策略的置换算法,很好地实现了两种置换策略的结合,但却没有给出一种在不同的应用场景中进行动态调整的机制。提出了一种对 LRFU 算法进行动态调整的方法,模拟测试发现改进的 LRFU 算法都不同程度地提高了缓冲区命中率。
关键词:数据库缓冲区;页面置换算法;LRFU;动态调整;改进的 LRFU 算法
DOI:10.3778/j.issn.1002-8331.2008.17.046 **文章编号**:1002-8331(2008)17-0153-05 **文献标识码**:A **中图分类号**:TP311

1 引言

直接从磁盘读取数据要比从内存读数据慢至上千倍,但是由于磁盘数据库的数据是存放在外部磁盘上的,于是基于预取数据和减少与磁盘的交互的缓冲区(Buffer)技术成为数据库管理系统中一项必不可少的提高性能的技术。缓冲区的基本思想是数据提前读取和滞后存储;数据页式(Page)管理、页式存取,"打包"成页以后进行磁盘读写。其基本目的是减少磁盘 I/O 的次数和减少一次磁盘 I/O 的开销。为达到这一目的其主要的实现方法是读取合适的磁盘页面,写出合适的缓冲区页面。读取磁盘的数据页面的方法主要有预取(Prepaging/Prefetching)和按需读取(Demand Paging),如何选择需写出的缓冲区页面就涉及页面置换算法,高效的页面置换算法就是将最不需要的页面置换出缓冲区,页面置换算法的好坏对缓冲区管理器的性能有着重要的影响。缓冲区页面置换算法的研究已出现多年,但是由于现实中数据库应用的复杂多变并且缓冲区页面置换技术在操作系统、存储系统、Web 应用、中间件、文件系统等领域都有广泛应用[1],现有的页面置换算法仍需要进行优化,页面置换算法的研究至今仍是一个研究热点。本文主要从数据库管理系统的角度描述缓冲区技术。

衡量页面置换算法的性能的主要指标是页面命中率和页面缺失率;当数据库发起页面访问请求时,如果所请求的页面在缓冲区中,称为页面命中(Hit);否则如果所请求的页面不在缓冲区,就产生了页面缺失(Miss)。

现有的页面置换算法有很多种,大都是基于近期的页面访问信息查找一个合适的页面,同时大都要维护一个页面状态表或页面队列,大都要记录近期的页面访问序列等等,这些信息或操作都需要一定的开销,而且有的还涉及页面队列中页面的移动,带来的开销就会相当大。同时这些算法大都没有根据实际应用进行动态的调整。

LRFU 缓冲区页面置换算法[2]是近年出现的一种算法;结

基金项目:国家部委预研基金资助项目(the Pre-Research Foundation of China Ministries and Commissions)。
作者简介:李占胜,男,硕士研究生,主研领域数据库实现技术与理论;毕会娟,女,博士,研究员;李艳平,女,硕士研究生;张立松,男,工程师。
收稿日期:2007-09-14 **修回日期**:2007-11-28

合其他页面置换算法，引入了一种动态调整的机制对LRFU算法进行了改进，使得LRFU算法能更好的适应多种应用环境。

2 相关算法

当数据库发生缺页中断时，缓冲区管理器就扫描缓冲区以查找空位置或可以替换出去的页面。缓冲区管理器可以随机选择一个页面进行置换，也可以根据一定的算法结合页面的逻辑访问序列选择置换页面，它们带来的性能有很大的差别；如果一个经常使用的页面被置换出去，很快又要被调入内存，就带来了不必要的额外开销，必然影响整个系统的性能。针对页面置换算法的设计，工作主要集中在以下几个方面：基于访问时间的算法（Recency Based Algorithm）、基于访问频率的算法（Frequency Based Algorithm）及访问时间频率结合算法（Recency/Frequency Based Algorithm）。另外，还有先进先出算法（FIFO），随机算法（Random）等。下面将对各种页面置换算法进行介绍。

最近最少使用算法（Least Recently Used，LRU）：LRU算法是使用较多的一种算法，LRU算法基于长时间没有使用的页面比那些最近访问过的页面有更小的最近访问的可能性的考虑，置换出那些最长时间没有访问过的页面。LRU算法中Cache容易被一次扫描式访问污染，对页面的访问频率不做区分，使得其无法适应需要。还有一种和LRU恰好相反的置换策略，最近使用替换算法（Most Recently Used，MRU），MRU算法置换出那些最近刚刚访问过的页面。另外时钟算法（Clock）是LRU的一个常用的有效的近似，并且比LRU算法更易实现。时钟算法比LRU算法实现简单，但是却无法弥补LRU算法的不足，在许多应用场景中仍然无法适应需要。

最少使用次数算法（Least Frequently Used，LFU）：LFU算法置换缓冲区中访问次数最少的页面。对于短时间内访问次数很多的页面，可能会在其不再使用时却很难被置换出缓冲区；同时LFU的运行开销比较大；于是实际应用中会使用LFU的改进算法。如LFU-Aging，一个页面被访问时，其之前的访问次数乘一个小于1的参数，使得不同的访问时间的访问次数具有不同的权重。

基于使用次数的算法（Frequency Based Replacement，FBR）[3]：这种策略结合了访问时间置换机制，将缓冲区分为三个区域：新、中间、旧；每个页面都记录访问计数；三个区域的大小都保持不变，页面根据需要在这三个区域之间移动。新区域保存最近访问页面，发生页面缺失时，选择旧区域中具有最小访问次数的页面替换。中间区域是为了避免新页面在第一次出新区域时马上被置换，有一个过渡期。FBR算法避免了缓冲区污染（Cache Pollution）问题，即具有较大的访问次数的页面长久地占用缓冲区的问题。

最短最近使用间隔算法（Low Inter-Reference Recency Set，LIRS）[4]：LIRS算法是一种基于LRU算法的弱点而改进的算法，使用了页面的最近使用间隔（Inter-Reference Recency，IRR）来决定要替换的页面。IRR用来表示一个页面的最近两次访问的间隔中的其它无重复页面的个数。LIRS还定义了一种不同的最近访问时间R（Recency），R用来表示一个页面的最近访问至当前时间之间的其它无重复页面的个数。如图1，页面1的IRR为3，R为2。LIRS算法置换具有较大的IRR值的页面；对页面最近访问的可能性进行了预测利用，并且实现与开销都不大。LIRS适合具有频繁访问特征的模式，而不适合某些具有时间局部性特征的访问模式。结合LIRS的思想还有一种Clock-Pro[5]算法。

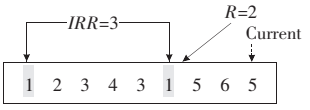

图1 LIRS算法的IRR示意

LRU-K算法[6]：LRU-K算法记录页面的前K次的访问时间，如果一个页面的访问次数不足K次，则其前第K次访问的时间记录为一个很大的值。发生页面缺失时，选择访问次数最小的且LRU值最大的页面进行替换。首先查找只访问了1次的页面，选择其中具有最大LRU值的页面进行替换，如果没有只访问了1次的页面，就查找访问了2次的页面，依次类推。LRU-K算法能够将访问次数少而长时间不使用的页面置换出缓冲区，但是LRU-K算法实际上是一种LFU算法的近似，仍然无法避免缓冲区污染的问题。

2Q算法[7]和MQ（Multi-Queue）算法[8]：2Q算法维护一个LRU队列和两个FIFO队列：Q_{in}、Q_{out}。Q_{in}保存被访问一次的缓冲区中的页面，Q_{out}保存被置换出的页面。LRU队列保存被访问次数超过1的页面。请求页面在LRU队列中时将其访问次数加1，在Q_{in}或Q_{out}中时将其移动到LRU队列，否则置换Q_{in}中的页面，将被置换的页面添加到Q_{out}。MQ算法维护了多个LRU队列$Q_0,Q_1,Q_2,\cdots,Q_{n-1}$；对于$i<j$，$Q_j$的页面比$Q_i$的页面具有更长的生命周期，还维护一个FIFO队列$Q_{out}$，用以记录被置换出的页面的访问次数。随时间变化，将不活跃的块从高一级队列移动到低一级的队列中，用以对队列进行调整，去除那些被访问很多次，但最近没有被访问的页面，防止它一直存在高一级的队列中，从而兼顾访问次数和访问时间。

ARC（Adaptive Replacement Cache）[1]算法：ARC将缓冲区分为两部分，一部分保存仅访问过一次的页面，另一部分保存访问次数超过一次的页面。每一部分还有一个相应的表用来记录从其置换出的页面。当进行逻辑读写或外存读写时页面记录在这四个记录区域（缓冲区的两部分和两个表）间进行移动，兼顾了访问次数和访问时间。CAR（Clock with Adaptive Replacement）和CART（CAR with Temporal filtering）[9]是将ARC的机制用到CLOCK算法中，维护两个Clock和两个LRU队列，运行机制与ARC相似。

使用时间与使用次数结合的算法（Least Recently Used/Least Frequently Used，LRFU）：缓冲区中每个页面都保存一个CRF（Combined Recency and Frequency）值，用以表示该块被继续访问的可能性；每个页面还用LAST记录上次被访问的时间。缓冲区管理器维护一个以CRF值排序的页面堆，根节点具有最小CRF值。通过权值函数计算CRF值，页面读入缓冲区时赋予一个CRF值，通过权值函数的调整可以使LRFU算法和LRU算法或LFU算法一致。当发生页面缺失时，选择具有最小CRF值的页面，即堆的根节点进行置换；新读入的页面插入堆中通过CRF值重新排序。LRFU的页面的CRF值只是在被访问时才改变，而有替换请求时却不变。该算法通过计算权值函数，兼顾访问时间（Recency）和访问次数（Frequency）两个值，其原则是最近的一次访问所占的权值最大，越往后其权值越小。

此外，还有一些借助分析预测文件的访问规律进行改进的页面置换算法，如UBM[10]、SEQ[11]等。借助于对文件访问规律的预测的方法实现复杂，并且适应范围有限。

这些算法主要的参考参数是访问时间或访问次数或两者的结合,LRFU 置换策略提出了一种很好的访问时间和访问次数结合的方法,但是其结合方法无法动态调整,不能很好的适应于实际应用环境,本文结合已有的动态调整的思想对 LRFU 进行了改进,为 LRFU 置换策略引进了自适应的机制。

3　改进方法

LRFU 算法对 LRU 和 LFU 两种置换策略的结合使用是通过权值函数,兼顾访问时间(Recency)和访问次数(Frequency)两个值,其原则是最近的一次访问所占的权值最大,越往后其权值越小。通过权值函数计算出 CRF 值,通过 CRF 值确定要置换出的页面。

CRF 值的计算公式为:

$$C_{t_{base}}(b)=\sum_{i=1}^{k}F(t_{base}-t_{bi})$$

$C_{t_{base}}(b)$即块 b 在第 base 时间点的 CRF 值。

$F(x)$是权值函数,$\{t_{b1},t_{b2},t_{b3},\cdots,t_{bk}\}$是块 b 被访问的时间点,$t_{b1}<t_{b2}<\cdots<t_{base}$。

$$F(x)=\left(\frac{1}{p}\right)^{\lambda x},(P\geqslant 2)$$

该权值函数有如下特性:

$C_{t_{b_k}}(b)=F(0)+F(\delta)C_{t_{b_{k-1}}}(b)$,其中 $\delta=t_{b_k}-t_{b_{k-1}}$

那么在实现时,可以通过记录上次访问的时间和上次访问时的 CRF 值即可算出当前时间的 CRF 值。

另外,该权值函数还有一个重要的特性如下:

(1)如果 λ 为 1 则它和 LRU 算法一致,根据 CRF 替换的和 LRU 算法是同一个块;

(2)如果 $F(x)=C$,C 为常数,而 λ 为 0 时 $F(x)$为常数,则算法和 LFU 一致,替换的是同一个块。

于是,在 λ 从 0 到 1 的变化过程中,LRFU 置换策略将从 LFU 置换策略过渡到 LRU 置换策略。但是在一个特定的 LRFU 算法中 λ 值是固定的,没有在实际的运行过程中动态的调整。于是,给定一个 λ 值以后,LRFU 算法其实是一种 LRU 和 LFU 两种置换策略的混合折衷方案,而无法在不同的场景中使得 LRU 或 LFU 置换策略发挥各自的长处。在有的应用场景中 LRFU 算法的命中率还不如 LRU 或 LFU 算法。在文[12]中给出了一种 λ 值动态调整的方法,但是这种方法实现复杂,而且并不见得能够起到积极的作用。

本文的改进主要在不同的应用场景应用不同的置换策略,通过动态调整 λ 值使 LRFU 倾向于不同的置换策略。2Q、MQ、ARC、CAR 等算法都保存了近期置换出的页面,这些历史页面信息对调整算法的性能起到了极为重要的作用。结合这些算法,将 LRFU 算法改进,使得缓冲区管理器保存一个替换出的页面的 LRU 队列 Q_{out} 用以分析历史信息,用以调整 λ 值。该 LRU 队列不保存实际的数据,只是记录近期置换出的页面的记录号和 CRF 值。

LRU 队列中的页面按照被置换出的时间顺序排列,刚被置换出的页面放在队列的首部,当 Q_{out} 已满时,清除队列尾部的数据,将刚置换出的页面放至页面首部。Q_{out} 用以分析文件的访问方式,实现动态地调整置换策略。

读取新的页面时,先在 Q_{out} 中查找新的页面的记录,如果找到相应的记录,在将新页面读取到缓冲区的同时将新页面的 CRF 值设为 Q_{out} 中的相应记录的 CRF 值;将 Q_{out} 中的相应记录删除。如果没有找到相应的记录,直接从外存读取数据并将新页面的 CRF 值通过权值函数设为 $F(0)$。另外将置换出的页面添加到 Q_{out} 的首部,如果 Q_{out} 已满,将队列尾部的记录删除。

在读取新的页面时,如果连续多次在 Q_{out} 中找到了相应的页面记录,或者多次在 Q_{out} 中找到了相应的页面记录,并且新的页面在 Q_{out} 中的次数比不在 Q_{out} 中的次数多的话,说明此时的应用模型倾向于这样一种模型:访问次数较多的页面再次被访问的可能性更大;于是就减小 λ 值,使得置换策略更倾向于 LFU 算法。如果新读取的页面,连续多次不在 Q_{out} 中,或者多次不在 Q_{out} 中,并且新的页面不在 Q_{out} 中的次数比在 Q_{out} 中的次数多的话,说明此时的应用模型倾向于这样一种模型:最早被访问过的页面再次被访问的可能性最小;于是就增大 λ 值,使得置换策略更倾向于 LRU 算法。另外,如果请求访问的页面在缓冲区中找到,也说明次数的应用模型倾向于访问次数较多的页面再次被访问的可能性更大;将请求访问的页面在缓冲区中命中的次数,与新页面在 Q_{out} 中找到的次数按同样的方式处理。

λ 值动态调整的过程可用如下伪代码说明:

```
MODIFY_TIMES=5; /* 改变的时机 */
MODIFY_RATIO=1.0; /* 改变的可能 */
MODIFY_VALUE=5.0; /* 改变量 */

InTimes=0; /* 缓冲区命中和新页面在 Qout 中的次数 */
OutTimes=0; /* 新页面不在 Qout 中的次数 */
IsPreIn;/* 上次的新页面是否在 Qout 中,为了统计连续出现的次数 */
SuccTimes=0; /* 记录新页面连续在(或不在)Qout 中出现的次数 */
IsInQout=0; /* 新页面是否在 Qout 中或请求页面是否命中 */

if ((新页面在 Qout 中有记录) OR(请求页面在缓冲区中命中))
{
    /* 删除在 Qout 中的记录 */
    InTimes++;
    IsInQout=1;
}
else
 {
    OutTimes++;
}

if (IsPreIn==IsInQout)  /* 更新 SuccTimes */
 {
    SuccTimes++;
}
else
 {
    SuccTimes=0;
}
IsPreIn=IsInQout;

 if ((InTimes >MODIFY_TIMES AND InTimes/OutTimes >MODIFY_RATIO)
        OR(SuccTimes>MODIFY_TIMES AND IsPreIn==1))
{
```

```
        λ/=MODIFY_VALUE;
        InTimes=0;
        OutTimes=0;
        SuccTimes=0;
    }

    if ((OutTimes>MODIFY_TIMES AND OutTimes/InTimes>MODIFY_RATIO)
        OR (SuccTimes>MODIFY_TIMES AND IsPreIn==0))
    {
        λ*=MODIFY_VALUE;
        if (λ>=1)
        {
            λ=0.999 9;
        }
        InTimes=0;
        OutTimes=0;
        SuccTimes=0;
    }
```

在实际应用中,MODIFY_TIMES,MODIFY_RATIO 及 MODIFY_VALUE 可以根据实际情况进行调整,不同的 MODIFY_TIMES、MODIFY_RATIO 及 MODIFY_VALUE 对算法的性能有一定的影响,经过测试发现 MODIFY_TIMES 的值为 5 时动态调整的效果最好。对于 MODIFY_VALUE,测试发现大于 2.0 的值之间的差异不明显;本文第 4 章的测试工作基于的 MODIFY_VALUE 值为 5.0。

通过本文第 4 章的测试数据可以看到应用这样的动态调整方法,实现了不同的置换策略的动态切换,可以使 LRFU 算法的命中率得到提高,从而可以使 LRFU 算法具有更广泛的应用范围。

4 性能测试

实际应用中,不同的置换策略在不同的应用环境中会有不同的性能表现,由于现实应用的复杂多变,很难有一种算法在各种应用环境中都有优异的性能表现。为了设计合适的测试用例,需要对数据库的应用进行分析,在数据库管理系统中,一般的文件访问方式有以下几种:

(1)顺序访问:文件扫描式的顺序向前访问;

(2)循环访问:对文件的某部分或整个文件进行多次顺序访问;

(3)聚簇访问:较多地访问文件的有些部分;

(4)概率访问:对文件的各部分数据的访问概率是相互独立的且固定的;

(5)随机访问:对文件的访问方式是不定的,不可预测的,纯随机的;这种情况下任何置换策略都失去意义,最简单的算法就会成为最好的算法。

在实际应用中,单纯的某种文件访问方式是不多的,经常是多种访问方式的混合体现。各种文件访问方式下都有最高效的页面置换策略,基于 MRU 的策略适合于文件的顺序访问和循环访问;基于 LFU 的策略适合于时间性聚簇访问;而基于 LFU 的策略在有的概率访问中能表现出较好的性能。理想的页面置换算法是不同的文件访问方式采用不同的页面置换策略,使不同的页面置换算法在系统运行期间动态的调整更换。LRFU 算法虽然对页面的访问时间和访问次数进行了综合的考虑,但是并没有提出动态调整的方法来没有根据实际情况进行及时的调整,在实际使用过程中其实是几种置换策略的混合使用,而没有充分发挥不同的置换策略在不同的应用场景中的高效率。本文对 LRFU 算法的改进实现了动态调整,当文件的访问方式发生改变时,置换策略也根据应用进行及时地切换,使得不同的应用场景使用最合适的置换策略;本文设计了几种不同的文件访问方式,从测试数据可以看到对 λ 进行动态调整带来的改进效果。

为了测试改进的效果,同时接近实际应用,在实际数据库系统应用环境中进行了测试。修改 Postgresql 的共享缓冲区的代码,所使用的 Postgresql 的版本为 7.4.2,在 Linux 环境中进行了测试。测试环境为 NeoShine Linux 桌面 3.0 系统,Intel Pentium 4 CPU 3.0 GHz,内存 1 G 的 PC 机。

测试发现,LRFU 算法虽然在多种情况下都有较好的性能表现,但并不是在各种情况下都有比 LRU 或 LFU 更优的性能。测试用例通过 Postgresql 的存储过程模拟了文件的顺序访问和聚簇访问,另外还测试了两种 OLTP(On-Line Transaction Processing)工作量(Workload),在多种缓冲区大小的情况下进行了数据对比。

几种工作量的文件逻辑访问请求(Request)次数如表 1 所示,这里 P1~P6 所有的页面的大小都是 2 048 字节,且都是只读访问。两种 OLTP 工作量的页面大小是 8 192 字节,是读写访问。测试中 LRFU 的 λ 值为 0.001,改进算法(以下用 pLRFU 表示)中 λ 的初始值是 0.001。

表 1 几种工作量的文件逻辑访问请求信息

Trace Name	Number of Requests
P1	13 343 734
P2	53 353 722
P3	333 383 722
P4	602 160
P5	602 184
P6	6 016 031
OLTP1	2 553 000~2 560 000
OLTP2	3 064 000~3 072 000

P1、P2、P3 模拟的是文件的顺序访问;P4、P5、P6 模拟的是文件的概率访问。对于两者访问方式都有的文件访问方式中,LRFU 及 pLRFU 算法的性能将会比 LRU 和 LFU 算法的性能都更优。表 2 列出了几种置换算法在几种情况下的命中率,可以看出,pLRFU 算法在各种情况下都比 LRFU 算法的命中率更高。一个 λ 值会使 LRFU 算法的性能固定在一个范围,而动态调整 λ 值则有可能调整 LRFU 算法的命中率,从几组不同文件访问方式不同工作量的数据中可以看出本文提出的 λ 值的动态调整方法很大程度地提高了 LRFU 算法的命中率。

由图 2~图 6 可以看出改进的 LRFU 算法在几种情况下都不同程度的提高了缓冲区命中率,说明对 λ 值的动态改变起到了积极的作用。本文主要是针对 LRFU 的改进,改进算法与 LRFU 算法进行对比已经能够说明改进的效果了;由于改进的算法不会比 LRFU 算法的性能差,所以较全面的对比情况可以在[2]中看到。通过以上测试数据可以看出本文的 λ 值动态调整方法,能够实现置换策略的动态调整,使得 LRFU 算法的命中率得到了提高,可以使 LRFU 算法具有更广泛的应用范围。

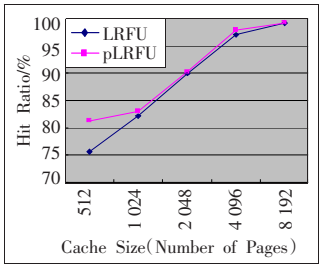

图 2　P4 中改进的 LRFU 和 LRFU 的命中率对比

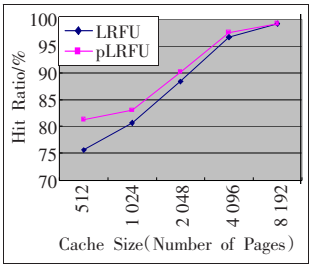

图 3　P5 中改进的 LRFU 和 LRFU 的命中率对比

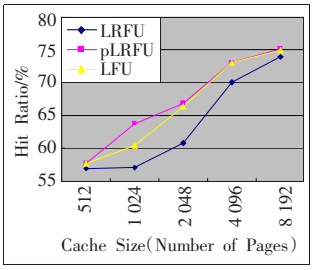

图 4　P6 中改进的 LRFU、LRFU 及 LFU 的命中率对比

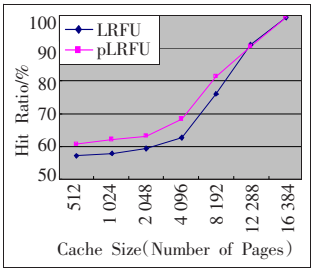

图 5　OLTP1 中改进的 LRFU 和 LRFU 的命中率对比

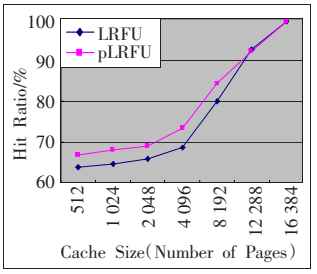

图 6　OLTP2 中改进的 LRFU 和 LRFU 的命中率对比

表 2　四种置换算法在几个工作量下的命中率对比　%

Trace	LRU	LFU	LRFU	pLRFU	Cache Size
P1	14.76	61.85	60.16	61.85	512
P1	59.00	94.53	93.27	94.53	1 024
P2	3.69	34.59	33.49	34.59	512
P2	14.75	61.95	60.36	61.95	1024
P3	0.59	14.71	14.21	14.71	512
P3	2.36	28.31	27.44	28.31	1 024

5　总结

LRFU 置换策略提出了一种不同置换策略结合的思想,在多种应用场景很好地提高了性能。LRFU 置换策略给出了一种改进的方向;而实际上,在有的情况下,其缓冲区命中率还不如 LRU 或 LFU。而 ARC 置换策略给出的动态调整的方法是一种比 LRFU 算法这种 LRU 和 LFU 的混合体更能适应实际应用的改进方法。本文结合 ARC 的动态调整方法对 LRFU 算法进行了自适应改进,提出了一种不同的置换策略动态调整的方法,这种调整方法实现了置换策略的切换,使得不同的应用场景使用不同的最适宜的置换策略。并且这种改进在带来缓冲区命中率的提高的同时并没有带来很大的额外开销。

实际应用中 λ 值变化的方式可以有多种形式,在不同的情况下有不同的表现,这里只实现了一种调整方式,而其他的一些动态调整方式也是有可能提高 LRFU 算法的性能的,这里不再进行介绍。

λ 值的动态变化带来了置换策略的改变,但是 CRF 值记录的是一种历史信息,有可能不是对当前的置换策略有用的信息,于是使用该信息可能会产生不正确的置换结果,这是一个应该改进的地方。不过本文的改进方法,可以使 CRF 值随着 λ 值的变化而变化,能够从记录访问时间(或次数)的历史信息变化到记录访问次数(或时间)的历史信息;测试结果显示,这种情况并没有带来负面影响。

致谢:中国人民大学信息学院博士生刘大为对测试工作给予了热心的指导和支持,在此表示衷心感谢。

参考文献:

[1] Megiddo N,Modha D.ARC:a self-tuning,low overhead replacement cache[C]//Proceedings of the 2nd USENIX Symposium on File and Storage Technologies,2003.

[2] Lee D,Choi J,Kim J H,et al.LRFU:a spectrum of policies that subsumes the least recently used and least frequently Used Policies[J].IEEE Trans Computers,2001,50(12).

[3] Robinson J T,Devarakonda M V.Data cache management using frequency-based replacement[C]//Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems,1990.

[4] Jiang Song,Zhang Xiao-dong.LIRS:an efficient low inter-reference recency set replacement policy to improve buffer cache performance[C]//Proceedings of the 2002 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems,2002.

[5] Jiang Song,Chen Feng,Zhang Xiao-dong.CLOCK-Pro:an effective improvement of the CLOCK replacement[C]//Proceedings of 2005 USENIX Annual Technical Conference,2005.

[6] O'Neil E J,O'Neil P E,Weikum G.The LRU-K Page replacement algorithm for database disk buffering [C]//Proceedings of the 1993 ACM SIGMOD Conference,1993.

[7] Johnson T,Shasha D.2Q:a low overhead high performance buffer management replacement algorithm [C]//Proceedings of the 20th VLDB Conference,1994.

[8] Zhou Y,Philbin J F.The multi-queue replacement algorithm for second level buffer caches[C]//Proc USENIX Annual Tech Conference,2001.

[9] Bansal S,Modha D.CAR:clock with adaptive replacement[C]//Proceedings of the 3nd USENIX Symposium on File and Storage Technologies,2004.

[10] Kim J M,Choi J,Kim J,et al.A low-overhead high-performance unified buffer management scheme that exploits sequential and looping references[C]//Proceedings of the 4th Symposium on Operating Systems Design and Implementation,2000.

[11] Glass G,Cao Pei.Adaptive page replacement based on memory reference behavior[C]//Proceedings of the 1997 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems,1997.

[12] Lee D,Choi J,Kim J H.On the existence of a spectrum of policies that subsumes the Least Recently Used (LRU) and Least Frequently Used(LFU) policies[C]//Proceedings of the 1999 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems,1999.