BROOM: Buffer Replacement using Online Optimization by Mining

Anthony K. H. Tung
Dept. of Computer Science
National Univ. of Singapore
tungkh@acm.org

Y.C. Tay
Dept. of Mathematics
National Univ. of Singapore
tay@acm.org

Hongjun Lu
Dept. of Computer Science
National Univ. of Singapore
luhj@comp.nus.edu.sg

Abstract

Buffer replacement is a classic research problem in database management. It has been extensively studied, and new strategies continue to be proposed. While existing strategies (e.g. LRU, LFU) use heuristics to determine which page to replace when there is a page fault, this paper presents a different approach, called BROOM, that is based on two ideas: for robust performance (e.g. regardless of buffer size), the replacement policy must (1) "know" something about the patterns of page references, and (2) imitate the optimum offline policy.

BROOM requires that historical reference streams be mined periodically to extract knowledge about current patterns in page access. It then uses this knowledge for online buffer replacement by imitating the optimum strategy. Simulation results are presented to show that BROOM sometimes has the best hit rates, but seldom the worst, over a wide range of system configurations and reference patterns.

1 Introduction

Since fetching data from disk takes much longer than from RAM, most database management systems (DBMSs) use a main-memory area as a buffer to reduce disk accesses. Generally, the buffer space is subdivided into frames of the same size, and each frame can contain a page from secondary storage. When a page is referenced by a transaction and is not found in the buffer, a *page fault* is said to have occurred, and the referenced page must be read from secondary storage into an unoccupied frame. If all frames are occupied, one of them is selected for replacement by the referenced page. The problem of selecting a frame is called *the buffer replacement problem.*

CIKM 98 Bethesda MD USA

Much research has been done on the buffer replacement problem (one early survey was done by Effelsberg and Haerder [3]). The simplest and most popular buffer replacement policy is *Least Recently Used* (LRU), in which the page in the buffer that is least recently accessed is replaced. Another well-known policy is *Least Frequently Used* (LFU) [10]. With LFU, a reference count is used to monitor the number of times each page is accessed; when a page fault occurs, the page with the lowest reference count is replaced.

More recently, O'Neil *et al* present a variant of the LRU algorithm, called *LRUk* [9]. LRUk keeps track of the times for the last k references to a page, and the page with the least recent last k-th reference will then be replaced. Thus, for $k = 2$, LRU2 replaces the page whose penultimate access is the least recent among all penultimate accesses. Experiments show that LRU2 outperforms LRU significantly, sometimes by as much as 40%. Later, Johnson and Shasha present a *2Q algorithm* [7] that behaves as well as LRU2 but with constant time overhead.

Inspired by the recent development in data mining techniques, and observing that LRUk and $2Q$ outperform other policies because they use more information about the reference stream, Feng *et al* initiated a study on the use of data mining for buffer management [4]. Their basic idea is to mine the database access histories to extract knowledge for the management of buffers in a distributed database. They show that this approach improves the hit rate significantly. Encouraged by this, we develop the idea further and introduce here a new replacement policy — *Buffer Replacement using Online Optimization by Mining* (BROOM). While Feng *et al* focused on a special problem (placement and replacement for *public buffers*), BROOM is developed as a generic replacement policy usable in any DBMS.

The development of BROOM is mainly motivated by two factors. The first factor is the existence of an optimal offline algorithm [2]. If all page references are known in advance, the optimal page replacement policy is to replace the page whose next use is furthest in the

future. The second factor is the existence of reference patterns in the database transactions [8]. If these patterns could be extracted (offline) from the database access histories, we could then match these patterns with current page references to predict (online) when a page will next be referenced; when a page fault occurs, the page that is most unlikely to arrive in the near future will then be replaced.

The remainder of this paper is organized as follows. Section 2 examines the reference patterns that are likely to occur in a database system and describes a system model for these patterns. Section 3 describes how a (training) reference stream is mined for knowledge, while Section 4 explains how the extracted knowledge is used to implement the replacement policy. Section 5 describes the simulation model in our study, and the experimental results are presented in Section 6. Section 7 concludes the paper by discussing some issues on the use BROOM.

2 System Model

The buffer replacement problem can be studied at different levels. For example, operating systems usually maintain buffers for user processes, and some application programs maintain their own buffers as well. This paper focuses on buffer management within the database system.

Most DBMSs do not rely on the buffer management provided by the underlying operating systems. Instead, a buffer pool is maintained for database users. From the viewpoint of buffer management, we simplify the environment as follows: Users issue transactions from some terminals; two terminals may run similar transactions at the same time, but each terminal has only one active transaction at any time. Transactions are processed by a database server that uses a main memory buffer (shared by all transactions) to keep frequently referenced pages and thus reduce disk accesses.

2.1 Assumptions

One basic assumption of our work is that, although there are random differences among transactions run from different terminals, database accesses do have certain patterns. For instance, there are so-called *hot* pages that are more frequently accessed than others. Examples of hot pages include the system catalog pages and the top levels of B-tree index pages.

In an early study, Kearns and DeFazio [8] demonstrated that the database reference patterns exhibited by a transaction change very little over time. Another assumption of our work is that users in a database system have relatively stable access patterns. That is, the access patterns presented in two traces of page accesses obtained at different times would have similar characteristics.

With the above assumptions, we formulate the mining approach to buffer replacement problem as follows: Given a (training) reference stream of database accesses from concurrent transactions that is recorded over a period of time, our task is to discover (offline) some knowledge from this stream, so that an online policy can use this knowledge to determine the page to be replaced in the buffer when a page fault occurs.

We now analyze in greater detail reference patterns in DBMSs. Specifically, we point out (1) the sources for the patterns — namely, the database system and user behavior, and (2) the two types of patterns — namely, *sequential* and *locality* patterns [8]. In a sequential pattern, page accesses are not repeated; these pages need not be in ascending (numerical) order (unlike the model by Kearns and Defazio). On the other hand, a locality pattern contains pages that are repeated a number of times.

In our model of the multiuser environment, these two types of patterns are mixed with random accesses to form transactions that are, in addition, interleaved.

2.2 Reference Patterns Induced by Database System

A database system affects reference patterns in many ways; For example, if a sequential scan is often done on a relation, pages that are in that relation will often be accessed close to each other and hence come together to form a reference pattern. The following are some additional examples:

1. Index scan using clustered index:
2. Index scan using non-clustered index:
3. Page accesses during joining operation:
4. Execution plans of query language:

While reference patterns that are induced by database system can be detected by carefully analysing the structure of the database. Such analysis are often tedious and data mining provide a way to automate this task.

2.3 Reference Patterns Induced by User Behavior

While database objects induce patterns within a transaction, different transactions may share patterns or form patterns. For example, a bank officer who needs to approve a loan may first access a client's financial information to make her decision, then proceed to enter her decision into another relation. These two transactions may then form one reference pattern between the indices of the two relations (whether it is sequential or locality depends on the behavior of the application).

Furthermore, the frequency of the reference patterns is also affected by user access behavior. For example, if a user's routine task is to generate report from a set

of selected data records, a certain data page (and index pages) will be accessed repeatedly.

Generally, reference patterns induced by user behavior are much harder to detect by human observation; they therefore form a natural target for data mining.

3 BROOM: Mining the Training Stream (Offline)

Based on the observations described above, we develop a new approach to buffer replacement for database environments: Buffer Replacement using Online Optimization by Mining (BROOM). As the name suggests, the approach has two characteristics: (1) using data mining techniques to discover patterns in page references from transactions; and (2) imitating the offline optimal solution to the problem.

The approach consists of two components: (1) mining (offline) a historical reference trace to obtain rules that reflect the relationships among database objects and transactions; and (2) applying the rules online to determine the pages that should be replaced. The mining process should be carried out periodically to make sure that the rules reflect changes in database structure and transactions. In this section we describe BROOM's first component (mining).

3.1 Mining Basic Rules

Given an offline reference stream, the optimum page replacement strategy is to swap out the page that occurs furthest down the stream when there is a page fault [2]. If we use a rule $P_i \xrightarrow{D_{ii}} P_i$ to denote the fact that there are D_{ii} expected page references between two consecutive occurrences of a physical page P_i, the optimum strategy suggests the following heuristic: swap out the page P_k in the buffer with the largest D_{kk}.

Existing replacement policies such as LFU, LRU and LRU2 more or less use such a strategy. Because future references are unknown, these strategies estimate D_{ii} in various ways. In LRU, D_{ii} is estimated by the number of references between when P_i was last accessed and when a pageout is required; thus, the page that has the largest D_{ii} (i.e. the least recently used) is chosen for replacement. LRU2 and LFU can similarly be viewed as strategies that are based on (different) estimates of D_{ii}.

For an online strategy to imitate the optimum behavior, we can first extract from the historical reference trace some knowledge about when a page will appear further down a reference stream. This leads to the first set of rules mined in BROOM — *Basic Rules* of the form

$$P_i \xrightarrow{D_{ii}} P_i .$$

Basic rules are mined and stored for every data pages and the mining process is straightforward.

3.2 The Idea Behind Advanced Rules

Using basic rules to predict the future is too simplistic, as we are only relying on an occurrence of P_i to provide hints on the next occurrence of P_i. For example, when several transactions run concurrently, the next occurrence of P_i is not only determined by the access pattern of a transaction itself, but also the number of interleaving active transactions. Thus, an occurrence of P_i may not provide a good hint on the next occurrence of P_i. To overcome this problem, BROOM uses *Advanced Rules*, of the form:

$$S_j \xrightarrow{D_{ji}} P_i$$

where S_j is a set of pages $\{P_{i_1}, ..., P_{i_k}\}$ and D_{ji} is the expected number of page accesses between $\{P_{i_1}, ..., P_{i_k}\}$ and the next occurrence of P_i. (We explain what "between" means in Section 3.3.1.) To see the idea behind advanced rules, consider the reference stream

$$P_9,\ P_{20},\ P_4,\ P_7,\ P_9,\ P_{11},\ P_{33},\ P_{11},\ P_{19},\ P_{50}.$$

Intuitively, we have $\{P_9, P_{20}\} \xrightarrow{3} P_9$, $\{P_9, P_4, P_{11}\} \xrightarrow{1} P_{33}$ and $\{P_9\} \xrightarrow{4} P_{33}$; where the distance (4) in the last rule is the average of two distances (6 and 2). If S_j consists of h pages, we call S_j an *h-item set.*

3.3 Mining Advanced Rules: Parameters

Since a reference stream may contain many thousands of page accesses, the number of possible rules is prohibitive. In BROOM, we tackle the problem in three ways, with the help of some parameters:

3.3.1 Window Size W

First, we use the concept of a *window.* By moving the window over the reference stream, the stream is transformed into a set of records, each of which contains the pages covered by the window. Thus, with a window of size W, a reference stream of length ℓ_{train} is transformed into $\ell_{\text{train}} - W + 1$ records of W accesses each, and item sets have maximum size W. By using windows, we restrict advanced rules to only those whose S_j appears in windows. Clearly, a W that is too large would admit spurious S_j's that are irrelevant to the underlying reference patterns. On the other hand, if W is too small to contain meaningful S_j's, we may lose some useful advanced rules.

If n_{terminal} is the number of terminals and $h_{\max}$ is the maximum size of frequent sets, then we set $W = n_{\text{terminal}} \times h_{\max}$ This choice of W is large enough that — with the interleaving of references from the transactions — there are, on average, $h_{\max}$ references from each terminal in every window. The choice of $h_{\max}$ is very much like the choice of k in LRUk algorithm. We increase it as long as it causes a significant improvement in performance.

We can now explain what we mean by the distance between S_j and P_i in an advanced rule: it is the number of references between P_i and a window containing S_j. We will however ignore repeated occurrance due to the same pages.

3.3.2 Minimum Support Level $MinSup$

Second, while basic rules are kept for all pages, advanced rules are only kept for S_j and P_i with a *support level* higher than a predetermined minimum support level $MinSup$. The support level of S_j and P_i are defined as follows:

$$support(S_j) = \frac{\text{number of windows containing } S_j}{\text{number of windows}}$$

$$support(P_i) = \frac{\text{number of windows containing } P_i}{\text{number of windows}}$$

We use different support levels for frequent sets of different sizes. Hence if $h_{\max} = 3$, then we must determine three values $MinSup_1$, $MinSup_2$ and $MinSup_3$ which are used as minimum support levels for 1-item sets, 2-item sets and 3-item sets respectively. These values are obain by generating a (uniformly) random reference stream, extracting the records, and taking the average frequency of a randomly generated h-item set;

To further reduce the number of rules, note that there is no point in trying to predict the arrival of a page that is rarely referenced. We therefore restrict our rules to only those whose P_i's satisfy a minimum support level $MinSupZ$.

3.3.3 Significance Level α

Third, to remove redundancy between basic and advanced rules (and thus reduce the number of rules), an advanced rule that hints at the arrival of a page P_i is kept only if the hint is significantly stronger than that provided by a basic rule; in other words, an advanced rule $S_j \xrightarrow{D_{ji}} P_i$ is kept only if D_{ji} is significantly less than D_{ii} in the basic rule $P_i \xrightarrow{D_{ii}} P_i$; the decision is made with a standard statistical test with significance level $\alpha = 0.99$.

3.4 Mining Advanced Rules: Algorithm

The process of mining advanced rules can be separated into two parts:

(a) Identifying Frequent Sets
This can be further divide into two steps:

i) Divide the training stream into records, then remove repetitions from these records.

ii) Use Apriori algorithm [1] to extract frequent sets from these records.

```
MineAdvanced()   // how advanced rules are mined
{
for each frequent set S_i {
 for each page P_j {
  if (#occurrences of P_j in reference stream > MinSupZ){
   scan reference stream {
    if (incoming page result in window containing S_i ) {
     find distance between window and following P_j occurrence;
     m ← number of such S_i-P_j distances;
     D_ij ← average distance;
     σ_ij ← standard deviation of distances; } } }
  m_j ← #occurences of P_j in reference stream;
  if ((D_ij − D_jj)/√(σ²_ij/m + σ²_jj/m_j) < −z_0.01 )
   output advanced rule: S_i --D_ij--> P_j;
  /* z_0.01 is the 99-percentile for standard normal variate
  the test here checks whether distance between S_i and P_j
  is significantly smaller than distance between consecutive P_j's
  */
} } }
```

Figure 1: Mining advanced rules.

(b) Identifying Closely Related Pages
After identifying the frequent sets, we use **MineAdvanced()** in Figure 1 to identify those pages P_j that follow these frequent sets at a distance which is significantly closer than the distance D_{jj} hinted by a previous occurrence of P_j.

4 BROOM: Applying the Rules to Online Stream

We now describe BROOM's second component (replacement policy).

4.1 Data Structures

BROOM uses a variable *StreamPtr* to count the number of arrivals; thus, *StreamPtr* is incremented every time a page reference arrives, and is reset to zero when it overflows. The distances D_{ii} for basic rules are stored in an array, while advanced rules are stored in a two-dimensional list structure — space constraint prevents us from describing this structure in detail.

In addition, BROOM uses four arrays to predict page arrivals. When a frequent set is detected inside a window, it is a hint that certain pages will be arriving at certain later times. To give room for error, we use two arrays $Early1[1..n_{\text{dbpage}}]$ and $Late1[1..n_{\text{dbpage}}]$ to store each page's earliest and latest predicted arrivals. When a page P_i arrives, the *StreamPtr* will be checked to see whether its value is between $Early1[P_i]$ and $Late1[P_i]$. If this is so, P_i is treated as the page that is predicted to have arrived. We must then update $Early1[P_i]$ and $Late1[P_i]$ to predict the next arrival of P_i. This update is straightforward if we predict two arrivals at a time, i.e. the next arrival is already predicted with another two arrays, $Early2[P_i]$ and $Late2[P_i]$ — we then

```
BROOM(P_in )
// updates predicted arrivals for buffer pages
// and selects buffer page for replacement
{
 if (P_in not in buffer) {
  // update estimates only when page fault occurs
  for each page B_i in buffer {
   // update estimates for pages in buffer
   if (StreamPtr > Late2[B_i]) {
    // both estimates obsolete, revise estimates
    // using D_{B_i} from B_i --D_{B_i}--> B_i
    Early1[B_i] ← StreamPtr +0.5D_{B_i};
    Late1[B_i] ← StreamPtr +1.5D_{B_i};
    Early2[B_i] ← Late1[B_i] +0.5D_{B_i};
    Late2[B_i] ← Late1[B_i] +1.5D_{B_i};
   } else if (StreamPtr > Late1[B_i])
    // 1st arrival estimate wrong and obsolete
    PushUp(B_i); // replace by 2nd arrival estimate }
 find page B_k in buffer with maximum Late1[B_k];
 replace B_k by P_in ; }
 if (StreamPtr > Early2[P_in ]) {
  // both estimates obsolete
  // update estimates for incoming page
  Early1[P_in ] ← StreamPtr +0.5D_{P_in} ;
  Late1[P_in ] ← StreamPtr +1.5D_{P_in} ;
  Early2[P_in ] ← Late1[P_in ] +0.5D_{P_in} ;
  Late2[P_in ] ← Late1[P_in ] +1.5D_{P_in} ;
 } else if (StreamPtr > Early1[P_in ])
  // 1st arrival estimate obsolete
  // replace by 2nd arrival estimate
  PushUp(P_in );
}
```

Figure 2: BROOM's page replacement policy.

simply move this second prediction into $Early1[P_i]$ and $Late1[P_i]$ and calculate the new values for $Early2[P_i]$ and $Late2[P_2]$.

4.2 Replacement Policy

BROOM's online actions can be divided into two separate parts. The first part monitors and triggers the advanced rules. When an advanced rule $S \xrightarrow{d} P$ is activated, the values of $Early1[P]$, $Late1[P]$, $Early2[P]$ and $Late2[P]$ are updated by a function Update() that will adjusts the predicted arrival time of page P accordingly.

The second part of BROOM decides which page to remove from the buffer and updates the expected arrival times by activating basic rules. The algorithm for this part is described in Figure 2.

5 Simulation

We experimented with BROOM by generating synthetic patterns and reference streams. As described in Section 2, a reference stream may have a mixture of sequential or locality patterns. For both types of patterns, 80% of the pages are hot and 20% of the pages are cold. Thus, if pages 0 to 19 are hot and 20 to 99 are cold, and pattern length is ℓ_{train}=10, then an example of a sequential pattern is { 14, 12, 18, 28, 11, 17, 16, 15, 90, 13} (recall that a sequential pattern has no repetitions) and an example of a locality pattern for $g = 2$ is { 14, 28, 14, 16, 90, 11, 16, 17, 17, 14}.

To generate the reference streams (for both training and test streams) recall first that patterns are embedded in transactions with other random accesses or other patterns that occur infrequently. We model this by introducing a random mode for terminals — it sends a random page request while in this mode. When a terminal is not in random mode, it will have an active pattern.

6 Experiments

We now use our simulator to compare BROOM's hit rate against LFU, LRU and LRU2. (It has been shown that the hit rate for LRUk, where $k > 2$, is only marginally higher than that for LRU2 [9].) Along the way, we will comment on the relative performance of the four policies and draw conclusions on BROOM's behavior.

Figure 3 shows the range of values used for the parameters in our experiments. Space and time constraint prevent us from fully exploring all combinations of all parameters, so some parameters are fixed at default values during the experiments.

BROOM needs some space in main memory to store its data structures (e.g. for the rules); therefore, to be fair to the other policies, we give them a number of extra buffer frames equal to the space needed by BROOM. Thus, in an experiment that compares the policies for a buffer size of n_{buff} pages, if BROOM needs s pages for data structure, then LFU, LRU and LRU2 are each given $n_{\text{buff}}+s$ pages of buffer space. The value of s is measured in each experiment after BROOM's mining phase, and it is usually about 5 (8Kbyte pages).

6.1 Small Buffers, Large Buffers

We first make one general observation: our experiments show that the hit rate for LRU2 is lower than for LFU when $n_{\text{buff}}/n_{\text{dbpage}}$ is small, but higher when the ratio is large. We use this observation to give an operational definition of buffer size: A buffer is *small* if LRU2 has a significantly higher hit rate than LFU; it is *medium* if their hit rates are about the same; and it is *large* if LFU has significantly higher hit rate than LRU2.

The idea behind this definition is that, while one might speak informally about a buffer being "too small" or "too large" when explaining a policy's hit rate, one cannot give a formal definition in terms of the many parameters (n_{dbpage}, n_{buff}, n_{terminal}, ...). Our operational definition avoids this difficulty, and we shall see that BROOM's performance can be described in terms of whether the buffer is small, medium or large.

With our operational definition, one cannot tell — in practice — whether the buffer for a system is small or large unless LRU2 and LFU are implemented for the system and their hit rates measured. However, there is

parameters for the database	
n_{dbpage}	number of physical pages in the database, fixed at 1000
n_{buff}	number of pages in BROOM's buffer, varies (20–300), default=100
n_{hot}	number of hot pages, fixed at 200 (so there are 800 cold pages)
$PageSize$	page size, fixed as 8Kbytes
parameters for reference patterns	
ℓ_{pattern}	length of a pattern, uniformly distributed (60–100)
n_{pattern}	number of patterns, varies (40–160), default=100
n_{seq}	number of sequential patterns, varies (0–100), default=50
n_{local}	number of locality patterns, $n_{\text{local}}=n_{\text{pattern}}-n_{\text{seq}}$
n_{appear}	average number of times a page appears in a locality pattern, varies (2–10), default=8
parameters for reference streams	
ℓ_{train}	length of training stream, varies (10000–100000), default=100000
ℓ_{test}	length of testing stream, fixed at 150000
n_{terminal}	number of terminals, fixed at 5
r_{random}	ratio of number of random pages to length of reference stream, varies (0.1–0.8), default=0.5

Figure 3: Values of parameters in the experiments.

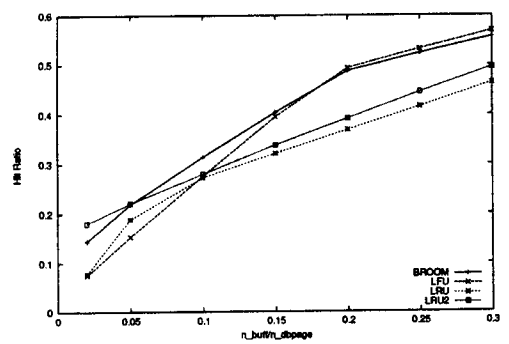

Figure 4: Hit rates for varying buffer sizes

no need to: We shall see that, unlike LRU2 and LFU, BROOM's hit rate is seldom the worst whatever the buffer size, and sometimes the best. BROOM is therefore robust in the following sense: Although we may not know in practice whether the buffer is "too small" today or "too large" tomorrow (because of changing circumstances), we are assured that BROOM will give a hit rate that is either the best, or better than the worst.

6.2 The Effect of System Parameters on BROOM's Hit Rate

We now study how BROOM's hit rate is affected by changes in the various parameters. Due to space constraint, we are only able to present the graphs that are obtained from varying n_{buff} and n_{pattern}. For the other parameters, we will only give the observations here and present the graphs in [11].

6.2.1 The effect of buffer size: Changing $n_{\text{buff}}/n_{\text{dbpage}}$

Figure 4 shows the hit rates for BROOM, LFU, LRU and LRU2 when $n_{\text{buff}}/n_{\text{dbpage}}$ varies (and other parameters are kept at default values). This graph illustrates three observations that hold true for the rest of our experiments:

(B1) For small buffers, LRU2 has a higher hit rate than BROOM, and BROOM has a higher hit rate than LFU.

(B2) For medium buffers, BROOM has the highest hit rate.

(B3) For large buffers, BROOM and LFU have similar hit rates (which are higher than LRU2).

To understand these three observations, we must first understand the relative performance of LRU2 and LFU. LFU relies entirely on frequencies, and has no concept of "position" in the online stream. For example, two pages P_i and P_j may have the same frequencies, but P_i's may be evenly spaced out, whereas P_j's are clustered together. LFU treats P_i and P_j equally, whereas LRU2's hit rate for P_j will be higher than for P_i. When the buffer is so small that not all hot pages can be kept in main memory, this causes LFU's hit rate to be worse than LRU2's. On the other hand, when the buffer is large enough to keep all hot pages, LFU outperforms LRU2, because it avoids replacing hot pages.

As for BROOM, when buffers are small, BROOM uses the distances in its rules to keep track of position in the online stream, and thus (like LRU2) outperforms LFU — this explains (B1). For large buffers, BROOM's basic rules contain similar knowledge as frequencies, so BROOM performs like LFU — we thus get (B3). These strengths that the rules give to BROOM also hold up its hit rate for medium buffers as LRU2 and LFU both begin to suffer from their weaknesses; hence (B2).

These observations suggest that BROOM's performance should be studied in three separate regions (viz. small, medium and large buffers), and the following experiments are presented accordingly.

6.2.2 The effect of number of patterns: Changing n_{pattern}

One expects that, as the number of patterns (n_{pattern}) increases, BROOM would fail to distinguish among the patterns and therefore have a lower hit rate. We can check this with a simple calculation: With $r_{\text{random}} \times \ell_{\text{train}}$ pages from the training stream generated by patterns, and n_{pattern} patterns of length ℓ_{pattern} each, there are $r_{\text{random}} \times \ell_{\text{train}}/(n_{\text{pattern}} \times \ell_{\text{pattern}})$ activations of each pattern in the training stream.

For our default values and average ℓ_{pattern}=80, each pattern is activated $0.5 \times 100000/(160 \times 80) < 4$ times when ℓ_{pattern}=160. This is hardly enough for pattern discov-

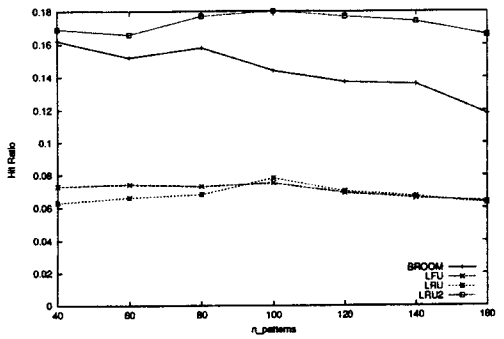

Figure 5: Varying $n_{\text{pattern}}$with n_{buff}=20

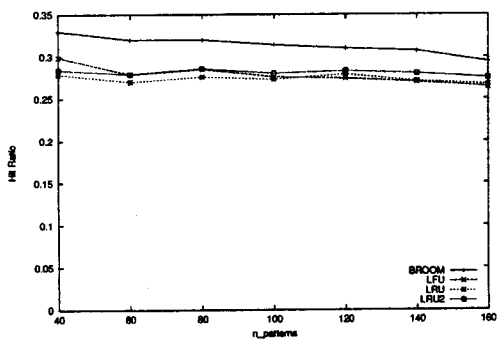

Figure 6: Varying $n_{\text{pattern}}$with n_{buff}=100

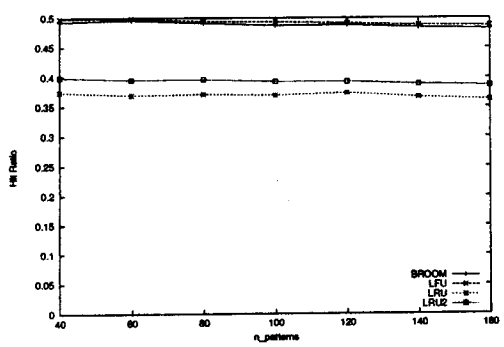

Figure 7: Varying $n_{\text{pattern}}$with n_{buff}=200

ery, and we can see the deterioration of BROOM's hit rate for small and medium buffers in Figures 5 and 6. We hence make the following observation:

(B4) For small and medium buffers, BROOM's hit rate decreases when there are more patterns.

Of course, another way of looking at this is that, if BROOM is to maintain its performance, then ℓ_{train}must be longer if n_{pattern}increases. However, there is a subtler reason for the negative effect of increasing n_{pattern}, and this effect is independent of ℓ_{train}. If a page occurs only in one pattern, then it provides a very strong hint for other pages that follow it in the pattern, even if the training stream contains only a few activations of this pattern. When n_{pattern}increases, the hot pages appear in more and more patterns, and the hints grow weaker, and this is another reason for BROOM's dropping hit rate when buffers are small and medium. For large buffers, BROOM is insensitive to the number of patterns (see Figure 7) since it — in effect — keeps all the hot pages in the buffer. Since LRU, LRU2 and LFU have no way of keeping track of patterns, their hit rates are insensitive to the number of patterns for any buffer size.

6.2.3 The effect of locality: Changing $n_{\text{local}}/n_{\text{pattern}}$

The mixture of sequential and locality patterns plays an important part in determining the hit rate of a policy.

For a small buffer, if there is little locality in the patterns ($n_{\text{local}}/n_{\text{pattern}}$is small), then LRU and LRU2 have little advantage over BROOM and LFU. This is because they rely on the past one or two accesses to estimate the hotness of a page, and there is too little locality for this estimate to be useful. As the locality increases, however, the hit rates for LRU, LRU2 and BROOM improve — with BROOM staying close to the leader LRU2.

For a medium buffer, BROOM is comparable to the best of the other three policies. We have following observation when we varied $n_{\text{local}}/n_{\text{pattern}}$:

(B5) For small and medium buffers, BROOM has higher hit rates when there is more locality.

6.2.4 The effect of repetitions: Changing n_{appear}

Another factor that affects hit rates is the number of times (n_{appear}) a page appears in a locality pattern. For small buffers , LRU2 begins to break away from the other policies as n_{appear}exceeds 4. When n_{appear}= 8, for example, LRU2 needs just 2 accesses to detect a hot page and thus get a hit for the other 6 accesses. With the knowledge mined, BROOM is able to stay close to LRU2 as n_{appear}increases, unlike LRU and LFU.

For medium buffers , BROOM has enough space to hold the pages that it knows are coming in, so it outperforms the other policies for a wide range of n_{appear}values. For large buffers , n_{appear}has negligible effect. We thus have the following observation:

(B6) For small and medium buffers, BROOM has higher hit rates when there are more repetitions in the patterns.

6.2.5 The effect of random references: Changing r_{random}

Recall from Section 5 that patterns may be interspersed with random references, and we model this by a probability r_{random}, which specifies the fraction of references

that are random. The following observation from is not surprising:

(B7) BROOM's hit rate decreases when there are more random references in the transactions.

The same is true for LRU, LRU2 and LFU because they also assume certain rules hold for the references, and these rules hold less and less as randomness increases. What is interesting is that the relative performance of the four policies remains unchanged for most of the range up to 75% random accesses.

7 Conclusions

We conclude with some discussion on the overheads of BROOM.

BROOM has two types of overheads: costs for mining and for usage of rules in buffer replacement.

Since mining is done offline, memory requirement is not an issue, as long as the time needed (including disk accesses) to extract the knowledge is short. For all of the tests in Section 6, mining the training stream took less than 5 minutes on a SunSparc Ultra-1 processor. An analysis of BROOM's data structures shows that mining takes $O(n_{\text{freq}} \times \ell_{\text{train}})$ time, where $n_{\text{freq}}$is the number of frequent sets discovered. Thus, even with a 10-fold increase in n_{freq}or ℓ_{train}, the mining time will be less than an hour; this seems acceptable even if mining is done everyday.

A more important issue lies in the cost of using the rules. We have already addressed (in Section 6) the issue of extra runtime memory requirements by adding extra buffers for LRU, LRU2 and LFU. BROOM's performance would be even better in the experiments if the other policies do not have these extra buffers. In practice, the situation could be better or worse.

If the number of patterns is huge, BROOM may need much more memory to store the rules, thus making BROOM less viable. On the other hand, we are conservative in using 8Kbyte pages in our experiments. This is expected to be too small for index page sizes by the year 2005 [5], when 64 Kbytes pages may be required for optimum performance. A larger page size means that more rules can be stored inside one buffer page, so BROOM will not need as many extra buffer frames, and therefore perform better than indicated in our experiments.

As for the extra time needed by BROOM to monitor and activate rules during buffer management, we measured this in our experiments and found it to be (on average) 1.5 times that of the other policies. Assuming a 10.8ms disk access time [5], our calculations show that this increase in processing time is still worth the increase in hit rate, especially where (B2) is concerned.

References

[1] R. Agrawal, R. Srikant. *Fast Algorithm for Mining Association Rule.* In Proc. of the 20th Conf. on Very Large Databases, pages 487-499, Santiago, Chile, September 1994.

[2] L.A. Belady. *A study of replacement algorithms for virtual storage computers.* IBM Systems Journal, 5:78-101, 1966.

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

[4] L. Feng, H. Lu, Y.C. Tay, K.H. Tung. *Buffer Management in Distributed Database Systems: A Data Mining-Based Approach.* In Proc. of VI Int'l on Extending Database Technology, Valencia, Spain, March 1998.

[5] J. Gray and G. Graefe *The Five-Minute Rule ten years later, and other computer storage rules of thumb.* SIGMOD Record, Vol. 26, No. 4, Dec 1997, pp. 63-68.

[6] J. Gray and G. F. Putzolu. *The Five-Minute Rule for trading memory for disc accesses, and the 10 Byte Rule for trading memory for CPU times.* In Proc. of the 1987 ACM SIGMOD Int'l Conf. on management of data, June 1987, pp. 395-398.

[7] T.Johnson and D.Shasha. *2Q: A low overhead high performance buffer management replacement algorithm.* In Proc. of the 20th Conf. on Very Large Databases, pages 439-450, Santiago, Chile, September 1994.

[8] J.P. Kearns and S. Defazio. *Diversity in database reference behavior.* Performance Evaluation Review, 17(1):11-19, May 1989.

[9] E.J. O'Neil, P.E. O'Neil and G. Weikum. *The LRU-K page replacement algorithm for database disk buffering.* In Proc of the 1993 ACM SIGMOD Int'l Conf. on management of data, pages 297-306, Washington D.C., USA, August 1993.

[10] J.T. Robinson and M.V. Devarakonda. *Data cache management using frequency-based replacement.* In Proc. of the 1990 ACM SIGMOD Int'l Conf. on management of data pages 134-142, Brisbane, August 1990.

[11] A. K. H. Tung, Y.C. Tay and H. Lu. *Mining Based Buffer Management Using BROOM.* Submitted for publication.