Buffer Management in Distributed Database Systems: A Data Mining-Based Approach *

L. Feng[1] H. Lu[2] Y.C. Tay[2] K.H. Tung[2]

[1] The Hong Kong Polytechnic University, Kowloon, Hong Kong
[2] National University of Singapore, Kent Ridge Road, Singapore 119260

Abstract. In this paper, we propose a data mining-based approach to public buffer management in distributed database systems where database buffers are organized into two areas: public and private. While the private buffer areas contain pages to be updated by particular users, the public buffer area contains pages shared among users from different sites. Different from traditional buffer management strategies where limited knowledge of user access patterns is used, the proposed approach discovers knowledge from page access sequences of user transactions and uses it to guide public buffer placement and replacement. The knowledge to be discovered and the discovery algorithms are discussed. The effectiveness of the proposed approach was investigated through a simulation study. The results indicate that with the help of the discovered knowledge, the public buffer hit ratio can be improved significantly.

1 Introduction

Due to the higher cost of fetching data from disk than from RAM, most database management systems (DBMSs) use a main-memory area as a buffer to reduce disk accesses. In a distributed database system, database is spread among several sites over a computer network. In order to support concurrency control as well as reduce the disk I/O time, an approach of using two levels of buffer in distributed systems has been proposed [20, 8, 6]. The basic idea is to have two levels of buffers: a public buffer area to accommodate data pages shared among a number of users and a set of private buffers for individual users from different sites. With two levels of buffers, the private buffer is searched first when a user submits a request for a page. If no such page exists, the public buffer is then searched and if hit, the required page is copied from the public buffer into the corresponding private buffer for use, otherwise, the page is read from disk directly. A user transaction only updates data pages in the private buffers. When the transaction commits, all updated pages in his/her private buffer are written back to the disk. To maintain buffer coherency, copies of these updated pages in the public buffer and other private buffers should be updated accordingly as well.

There are three basic buffer management issues in such a system:

* The first two authors' work is partially supported by NUS academic research fund RP 3950660 and Hughes Research Laboratory Grant 3044GP.95-120

– **public buffer placement problem**: When a page which is not in the public buffer is read from disk into a private buffer, should the page be placed in the public buffer as well? Intuitively, pages that will not or seldom be referenced by other users afterwards should not enter the public buffer.
– **public buffer replacement problem**: When a public buffer frame is needed to bring in a page from disk, and all current buffer frames are in use, which page from the public buffer should be replaced?
– **private buffer replacement problem**: When a private buffer frame is needed to bring in a page from the public buffer or disk, and all current buffer frames are in use, which page from this private buffer should be replaced?

Among the above three issues, buffer replacement problem has been extensively studied. An early survey can be found in [9]. The most popular and simple buffer replacement strategy is LRU, the Least Recently Used [15, 3]: When a new buffer frame is needed, the page in the buffer that has not been accessed for the longest time is replaced. LFU (Least Frequently Used) policy is another simple but effective buffer replacement policy based on the frequency a page is referenced [16]. It relates a reference count to each page, and replaces the one with the smallest reference count when a buffer frame is needed.

In database buffer management, in addition to buffer replacement, *buffer allocation* is also an important and closely related issue and is often studied together. It addresses the problem of how many buffer frames should be allocated to a particular query. Sacco and Schkolnick, after studying the page reference behavior of queries, proposed the well-known hot set model to determine the optimal buffer space allocation for a query [17, 18]. Chou and Dewitt [7] extended this model with the DBMIN algorithm that separates the modeling of the reference behavior from any particular buffer management algorithm.

The above mentioned buffer management strategies for both allocation and replacement do not distinguish the importance of queries. Carey, Jauhari and Livny proposed two priority buffer management algorithms, Priority-LRU and Priority-DBMIN, to deal with the situation where database transactions are of different priority levels [4]. The major difference between the new schemas and the non-priority equivalents is that the priority strategies allow higher priority transactions to steal buffer pages from lower priority transactions. Their approaches were further developed that led to a strategy called Priority-Hints, which utilizes priority hints provided by the database access methods to instruct the buffer management [10]. With the similar idea, Chan, Ooi and Lu suggested to allow an access method to pass replacement hints to the buffer manager by assigning priority values to the buffer pages, and always select the page with the lowest priority value for replacement [5].

More recently, O'Neil and Weikum pointed out that it is difficult to design a query plan-based buffering algorithm that works well in a multitasking systems [15, 19]. They presented a self-reliant full buffer variant algorithm of LRU called LRU-K. The basic idea is to keep track of the times of the last K references to popular database pages and use this information to statistically estimate the inter-arrival times of references on a page by page basis. With K = 2, LRU-2 replaces the page whose penultimate access is the least recent among all penultimate accesses. Their experiments show LRU-K has significant cost/performance advantages over conventional algorithms like LRU, sometimes as high as 40%. To reduce the overhead cost of LRU-K, Johnson and Shasha presented a 2Q algorithm that behaves as well as LRU-2 but has constant time overhead [11]. 2Q follows the same principle as LRU-K, i.e., basing buffer priority on sustained popularity rather than on a single access.

Inspired by the recent development in data mining research and observing that LRU-K and 2Q algorithms outperform other strategies because they use more information about page references than the other strategies, we initiated a study on data mining-based buffer management. The basic idea of such an approach is rather simple: database access histories are collected and knowledge that can guide the buffer management is extracted. At run-time, the buffer manager uses the discovered knowledge to make decisions related to buffer allocation, placement, or replacement. The key issues to the success of the approach include:

1. what type of knowledge should be discovered;
2. how the knowledge discovered can be used effectively in run-time; and
3. whether the knowledge discovered from the access reference history can be generalized.

In this paper, we report some preliminary results on applying the approach to the problems of *public buffer placement and public buffer replacement* described earlier. One of the major reasons to choose them as the subject of our preliminary study is that little related work has been reported and the buffer replacement strategy used in most of the previous work is the simplest one, LRU [6].

In this paper, we propose the form of knowledge to be discovered from user access patterns and demonstrate through simulation that the approach does improve the hit ratio of public buffer significantly. Limited by resources, we did not address the third issue in this paper. However, based on an earlier study by Kearns and DeFazio, who have demonstrated that the database reference patterns exhibited by a transaction change very little over time [12], we assume that users in a database system have relative stable access preferences from a long period of view. Furthermore, data collection and knowledge discovery can be performed periodically so that the knowledge used can best reflect the current user access patterns.

Developing a new and effective approach to solve traditional database buffer management problems is only one of the tangible contributions of our study. The more important contribution of the paper is that, it is a first try to apply data mining technologies to database management itself. It is our fond hope that the work reported here will stimulate the interests to make database management a new application area of data mining technologies.

The remainder of the paper is organized as follows. Section 2 presents a data mining-based approach towards public buffer management in distributed DBMSs. Section 3 describes a simulation model used in our study with the results presented in Section 4. Section 5 concludes the paper with a brief discussion of future work.

2 Data Mining-Based Buffer Management

In this section, we describe our data mining-based approach to database buffer management in distributed DBMSs. In particular, we are interested in the problem of public buffer management in a distributed database system with two levels of buffers. After introducing the framework of such approach, we propose to use two types of knowledge about user access behavior, and describe how such knowledge can be mined. The public buffer placement policy and replacement policy based on mined knowledge are presented at the end of the section.

2.1 The framework

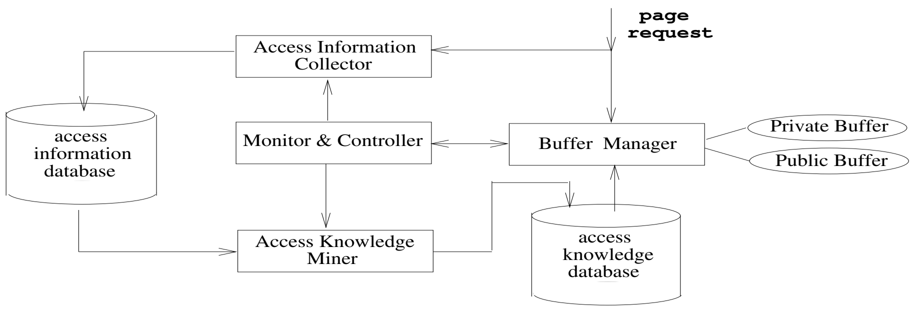


Fig. 1. Data mining-based buffer management

Figure 1 shows the framework of a data mining-based buffer management approach. In addition to the traditional buffer manager in a distributed DBMS, three new components *Access Information Collector* , *Access Knowledge Miner* and *Monitor and Controller* are introduced in the buffer management process. These components cooperate as follows:

- The Buffer Manager is responsible for all the operations on the buffer, including buffer search, buffer placement and buffer replacement, etc. Different from traditional buffer managers where most decision making rules are hard-coded, it uses the access knowledge database in its decision making process.
- The Monitor and Controller monitors the pre-specified performance metrics of buffer management, e.g., hit ratio. When the hit ratio is below certain level, it signifies that the stored access knowledge may be outdated. New access information should be collected and the access knowledge database should be updated by mining from the new data sets.
- When access information is to be collected, the Access Information Collector records the page requests of each transaction. The collected data are pre-processed according to the requirement of data mining process. The results are stored in an access information database.

– After enough data are collected, the Access Knowledge Miner will apply the mining algorithms to the collected data. The access knowledge discovered is used to update the access knowledge database. This process runs off-line.

2.2 Mining knowledge about user access behavior

As in every data mining application, we need identify what to be mined. In the public buffer management problem, we identify two types of knowledge, *access probability* and *associated access,* motivated by our understanding of the characteristics of database accesses.

Access probability In distributed database systems, users from different sites are usually interested in different portion of the database. For example, in a university database, users at the personnel department mainly access the data related to staff members while users at the registrar's office access the data related to students and courses. Thus, knowledge about the data portions that are of interests to users at different sites may be used to guide the buffer placement and replacement. To represent such knowledge, we introduce the notion of *access probability* to relations and data pages that reflects how often a relation or a page is accessed.

Definition 2.1 Relation Access Probability $ProbRel(s,r)$. Given a set of transactions T presented by users at site s, the relation access probability of site s to relation r is the fraction of transactions that access relation r to the total transactions of site s, i.e., $ProbRel(s,r) = \frac{|T_r|}{|T|}$, where $T_r = \{t_r \mid t_r \in T \wedge t_r \mapsto r\}$, and $t_r \mapsto r$ represents that transaction t_r accesses relation r.

Definition 2.2 Page Access Probability $ProbPage(s,p_r)$. Given a set of transactions T presented by users at site s, the page access probability of site s to relation r's page p is the fraction of transactions that access relation r's page p to the transactions that access relation r, i.e., $ProbPage(s,p_r) = \frac{|T_{p_r}|}{|T_r|}$, where $T_r = \{t_r \mid t_r \in T \wedge t_r \mapsto r\}$, $T_{p_r} = \{t_{p_r} \mid t_{p_r} \in T_r \wedge t_{p_r} \mapsto p\}$.

Associated access Another observation from the user access pattern is that, there usually exists some coherent relationship among successive accesses within a database transaction. In other words, some *associations* among pages referenced may exist in page reference sequences of user transactions. That is, *if certain pages have been accessed, then there are some other pages will be most likely accessed as well*, Such association results from several factors. One is due to semantic relationship among the data. For example, after a student record is accessed, the related pages containing his/her course information are more likely to be accessed also. Other reasons resulting in association among pages include physical data organization, query processing methods, etc. For example, a nested loops join will repeatedly scan a set of pages a number of times. The same set of index pages are always accessed before a data page is accessed. Apparently, if we can discover such associations among page references, we may be able to predict future page requests from the pages that have already been accessed by a transaction.

Mining association rules from transactional data is a classic data mining problem [1]. The main applications reported are related to the market ananlysis of sales data. We here apply the similar concept and define associated page access rules as follows.

Definition 2.3 Associated Access $AssoAccess(s,\, prepages \Rightarrow aftpages)$. Given a set of transactions T presented by users at site s, the associated access $prepages \Rightarrow aftpages$ of site s is that, access to page set $prepages$ imply the accesses to page set $aftpages$, with support s and confidence c defined as: $s = \frac{|T_{p+a}|}{|T|}$, $c = \frac{|T_{p+a}|}{|T_p|}$, where

$$\begin{aligned} T_{p+a} &= \{t_{p+a} \mid t_{p+a} \in T \ \wedge\ \forall p_i \in prepages\ (t_{p+a} \mapsto p_i) \ \wedge \\ &\quad\ \forall p_j \in aftpages\ (t_{p+a} \mapsto p_j)\}, \\ T_p &= \{t_p \mid t_p \in T \ \wedge\ \forall p_i \in prepages\ (t_p \mapsto p_i)\} \end{aligned}$$

Mining access probability and associated access rules To mine the above defined knowledge about user access behavior, the Access Information Collector collects and preprocesses the page access information (e.g., removing locality from page references, etc.), and stores the access information into the access information database, which contains records with the following form: *(transaction_id, site_id, list_of_relations, list_of_pages for each accessed relation).*

Mining access probabilities is a simple counting process: both $ProbRel(s, r)$ and and $ProbPage(s, p_r)$ can be computed by scanning the access information data. The mining of associated access rules consists of two steps:

- Finding all sets of pages that have support greater than the minimum support. Such page sets are called frequent sets, or largesets. Here, we adopted the Apriori algorithm [2]. It first finds the largesets of one page. Using the largesets of $k-1$ pages, the algorithm finds largesets of k pages recursively.
- Using the found largesets to generate the desired rules. That is, for every largeset l, find all non-empty subsets of l. For every such subset a, output a rule of the form $a \Rightarrow (l - a)$ if the ratio of support(l) to support(a) achieves the desired confidence level.

The output of the Access Knowledge Miner is stored in an access knowledge database. The access probabilities are stored in two tables, $ProbRel$ and $ProbPage$, with the following schema:

ProbRel(**relation_id, site**, probability)
ProbPage(**page_id, site**, probability)

For a large database system with a number of sites, the size of table $ProbPage$ could be reasonably large. To be able to retrieve the probability of certain page efficiently, index could be built on page_id. The associated access rules can also be stored in tables. However, as the number of pages in the left and right hand side of the rules varies, such rules can be simply stored in text files and a run-time data structure is used to hold the rules in memory. By controlling the thresholds of support and confidence levels, the number of such rules can be adjusted so that the space and computational overhead are tolerable.

2.3 Applying access knowledge in public buffer management

After obtaining the data mining results, we can use them to guide public buffer placement and replacement. To do so, we use $interest(p_r)$ to measure how likely page p_r tends to be accessed by users from all sites.

Definition 2.4 Page Interest $interest(p_r)$. The access interest that users from N sites show on page p_r is defined as:

$$interest(p_r) = \sum_{s=1}^{N} SiteAccessProb_s(p_r)$$

where values of both $interest(p_r)$ and $SiteAccessProb_s(p_r)$ are changing at run-time as follows:

Case 1 From the definition, the probability that users from site s access page p_r is equal to $ProbRel(s,r) * ProbPage(s,p_r)$.
Case 2 Once relation r is accessed, $ProbRel(s,r)$ becomes 1, and $SiteAccessProb_s(p_r) = ProbPage(s,p_r)$.
Case 3 When users from site s accessed all *prepages* of certain association rule(s), and page p_r belongs to *aftpages* of the rule(s), then $SiteAccessProb_s(p_r)$ becomes c, where c is the maximum confidence value of all these rules.
Case 4 After page p_r has been referenced and retrieved into the private buffer of site s, these users will no longer search p_r from the public buffer. That is, $SiteAccessProb_s(p_r)$ becomes 0.

As a summary, we have

$$SiteAccessProb_s(p_r) = \begin{cases} ProbRel(s,r) * ProbPage(s,p_r) & \text{(Case 1)} \\ ProbPage(s,p_r) & \text{(Case 2)} \\ c & \text{(Case 3)} \\ 0 & \text{(Case 4)} \end{cases}$$

With the *interest* of each page, the public buffer placement and replacement policies can be simply described as follows:

- placing those pages whose *interest*s are greater than a pre-determined value into the public buffer; and
- replacing the page with the least *interest* in the public buffer when a buffer frame is needed.

Astute readers may argue that the updating of the interest measure of pages will incur high overhead, especially when the database is large, as the number of pages in the database are very large, and the number of associated access rules are also large. To make a compromise, access probability thresholds are introduced to discretize the relation or page access probability: for a relation(page) whose access probability is below the threshold value, its access probability will be set to 0, otherwise to 1. With such simplification, we maintain two bit arrays $RelBit_r[N]$ and $PageBit_{p_r}[N]$ for each relation r and each page p_r of r, where N is the number of total sites in the whole system. $RelBit_r[s] = 1$ means that users from site s have high possibility to access relation r, and 0 otherwise. So is $PageBit_{p_r}[s]$. As the Page Access Probability is calculated on the premise that its corresponding relation has been accessed, users at site s are supposed to access page p_r afterwards if and only if $(RelBit_r[s] = 1) \ \wedge \ (PageBit_{p_r}[s] = 1)$. The page *interest* that users from all sites show on certain page can thus be redefined as follows:

Definition 2.5. Refined Page Interest $interest'(p_r)$. The access interest that users from N sites show on page p_r is defined as:

$$interest'(p_r) = \sum_{s=1}^{N} SiteAccessProb'_s(p_r)$$

where

$$SiteAccessProb'_s(p_r) = \begin{cases} 1 \text{ if } (RelBit_r[s] = 1) \ \wedge \ (PageBit_{p_r}[s] = 1), \\ 0 \text{ otherwise.} \end{cases}$$

With the refined page interest measure, we can have the following public buffer placement policy *PIG0 (Page Interest Greater than 0),* and public buffer replacement policy *LPI (Least Page Interest).*

PIG0 Public Buffer Placement Policy

When a private buffer requests a page p_r that does not exist in the public buffer, page p_r should be placed into the public buffer if and only if there exists at least one site s user, such that both $RelBit_r[s]$ and $PageBit_{p_r}[s]$ are 1, in other words, the refined page *interest* of p_r is greater than 0, i.e.,

$$\exists s \ (RelBit_r[s] = 1 \ \wedge \ PageBit_{p_r}[s] = 1) \ \Leftrightarrow \ (interest'(p_r) > 0)$$

LPI Public Buffer Replacement Policy

When a new public buffer frame is needed for a page about to be read in from disk, and all current buffer frames are in use: the page p'_r to be replaced is the one whose refined page *interest* is less than any page p_r in the public buffer, i.e.,

$$interest'(p'_r) = Min(interest'(p_r))$$

3 A Simulation Model

In this section, we describe our simulation model of a distributed DBMS concentrating on the buffer management functions and the synthetic data generated for studying the performance.

The model consists of three major components: the distributed database itself; a *Source* which generates the workload of the system; and a *buffer manager* which implements the public buffer management algorithm. The major parameters and their values are summarized in Table 1

The distributed database is modeled as a collection of *relations*, and each relation is modeled as a collection of pages, including index pages and data pages. The buffer manager component encapsulates the details of database buffer management schema. It consists of two parts, the private buffer manager and the public buffer manager. In this study we are only concerned with the public buffer management. pub_buf_size is the public buffer size, measured in the percent of total number of database pages num_pages, and $private\_buf\_size_s$ is the site s's private buffer size ($1 \leq s \leq num\_site$). $threshold$ is the access probability threshold defined to filter and refine the original relation and page access probabilities.

The source module is the component responsible for modeling the workload in the distributed DBMS. Users access the distributed database through transactions. A transaction is a sequence of read and write requests. The system at each site can execute multiple requests coming from all sites concurrently on the premise that data consistency is not violated. There are totally num_sites number of sites in the system. The number of transactions presented by users from site s ($1 \leq s \leq num\_sites$) is num_trans_s. $max\_rels\_per\_trans$ is the maximum number of relations that can be accessed by one transaction. Each request can reference at most $max\_pages\_per\_request$ number of pages.

Each transaction is supposed (but not unrealistic [13]) to access database pages with a Zipfian distribution, that is, the probability for referencing a page with page number less than or equal to i is $(i/num\_pages)^{loga/logb}$ with a and b between 0 and 1. The meaning of a and b is that a fraction a of the references accesses a fraction b of the num_pages pages (and the same relationship holds recursively by within the fraction b of hotter pages and the fraction $1-b$ of colder pages) [11, 15, 19]. In addition, observing that users from different sites tend to be interested in different relations and pages, i.e., have different hot relations and pages, we use $hot\_rel\_overlap$ and $hot\_page\_overlap$ to simulate the overlapped hot relations and pages for users from two successive sites. To model the coherent associated accesses in one transaction, we determine a potentially largeset number $num\_largesets$. In each largeset, the maximum number of relations is $max\_rels\_per\_largeset$. The maximum and minimum number of pages per relation in a largeset is $max\_pages\_per\_largeset\_rel$ and $min\_pages\_per\_largeset\_rel$ respectively. $support$ and $confidence$ are support and confidence levels regarding the association rules.

The values given in Table 1 are used in our simulation. The system consists of 8 sites. There are totally 20 relations and 30000 pages in a database. Each re-

The Distributed Database		
num_sites	no. of sites in the system	8
num_rels	no. of relations in database	20
num_pages	no. of pages in database	30,000
rel_size_r	no. of pages in relation r	1,500
The Buffer Management		
pub_buf_size	public buffer size relative to the DB size	2.5-12.5%
$private\_buf\_size_s$	site s's private buffer size	
$threshold$	relation/page access probability threshold	0-70%
The Workload		
Transaction Parameters		
num_trans_s	no. of transactions posed by site s user	100
$max\_rels\_per\_trans$	max. no. of relations per transaction	5
$hot\_rel\_overlap$	overlap of hot relation among sites	0-100%
$hot\_page\_overlap$	overlap of hot page per relation among sites	0-100%
Request Parameters		
$max\_pages\_per\_request$	max. no. of pages per request	20
$num\_requests$	no. of requests executed by one site	100,000
Associated Access Parameters		
$num\_largesets$	no. of potentially largesets	5
$max\_rels\_per\_largeset$	max. no. of relations in a largeset	3
$max\_pages\_per\_largeset\_rel$	max. no. of pages per relation in a largeset	10
$min\_pages\_per\_largeset\_rel$	min. no. of pages per relation in a largeset	30
$support$	support level of associated access rules	40-100%
$confidence$	confidence level of associated access rules	98 %

Table 1. The parameters and values

lation has 1500 pages. At least 100 transactions are presented by users from each site. The maximum number of relations referenced by a transaction is 5. Each request of a transaction accesses at most 20 pages. There are totally 100,000 requests coming from 8 sites. For each user, the page reference frequency distribution is skewed, in that a percent of transactions accesses b percent of database relations and pages. Here, we assign 80% to a and 20% to b. To simulate different hot spots for users at different sites, we use the term *overlap*, ranging from 0% to 100%, to express the overlapped hot portions between two successive sites. For associated accesses, we allocate 5 potential largesets, with each containing 1-3 relations. Each relation in a largeset refers to 3-10 pages. The confidence level for association rules is set to 98%, and the support level is selected from 40% to 100% according to different data sets being generated. Public buffer size is varied from 2.5% to 12.5% of the total pages. As we do not consider private buffer management issues in the paper, we do not limit the size of private, and simply assume that private buffer frames will be available when it is requested.

The relation/page access probability threshold is set from 0 to 70%.

To conduct the performance study using the model, two types of workload were generated. One is to simulate the historical data (training data) and the others are used as testing data to evaluate algorithms. Different random seeds are used in generating different data sets. Each data set, a steam of page accesses, was generated in three steps. The first step is to mark hot relations and hot pages of each relation for users from different sites. The 80-20 rule is used here: 20% of pages were marked as hot pages. The second step is to generate a sequence of page accesses for users at each individual site. Again, using the 80-20 rule, 80% of accesses will be hot pages. The associated access within a transaction were obtained in the similar manner as generating synthetic data sets for general association rules. To form a single stream, each user's access stream was broke into *requests*. The number of pages of each request was determined randomly. Requests from different sites were randomly picked to form a single access stream, that are used either as training data or test data.

4 Experiments and Results

In this section, we present the performance results using the simulation model defined in the previous section. Since the purpose of introducing public buffer is to reduce the number of disk I/Os, the major performance measure used in the simulation is the public buffer hit ratio, i.e., the ratio between the number of pages found in the public buffer and the total number of pages requested. The three buffer management strategies studied are:

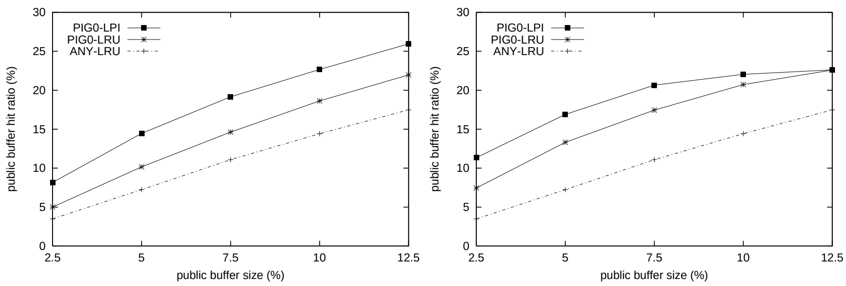
PGI0-LPI Using PIG0 and LPI described in Section 2 as the placement and replacement policies, respectively.

PGI0-LRU Similar to PIG0-LPI except that LRU is used as the replacement policy.

ANY-LRU No data mining results are used. While LRU is used as its replacement policy, every page requested from the database is placed into the public buffer.

4.1 Experiment 1: the effect of public buffer size

This experiment investigated the impact of buffer size when it was varied from 2.5% to 12.5% of the total number of database pages. The results obtained with *threshold* equal to 15 and 30% are shown in Figure 2. *hot_rel_overlap* and *hot_page_overlap* are both set to 75%. For each setting, five testing data sets with different random seeds were used and the average is used in the curve. It is evident that the buffer hit ratio increases with the size of public buffer under all three policies, as more pages can be kept inside the buffer pool. Two algorithms PIG0-LPI and PIG0-LRU, which utilize data-mining results more or less, are superior over algorithm ANY-LRU which does not take historical access knowledge into consideration. For instance, at buffer size 2.5% and *threshold* 30%, the



(a) threshold=15%   (b) threshold=30%

Fig. 2. Public buffer hit ratio versus public buffer size

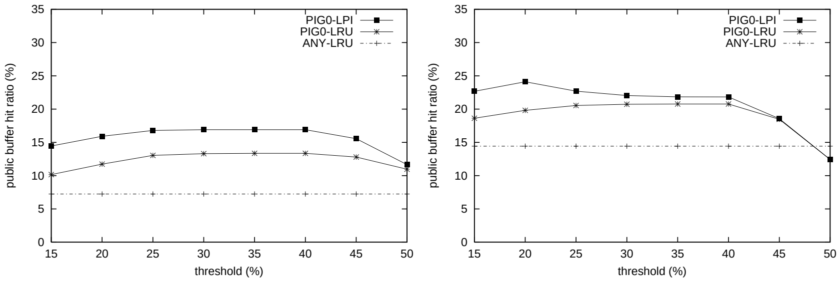
hit ratio of ANY-LRU is 3.49%, while those of PIG0-LRU and PIG0-LPI reach 7.45% and 11.34% respectively, about 53% and 69% more. Comparing PIG0-LRU and PIG0-LPI, we may find that, with suitable access probability threshold, PIG0-LPI can achieve higher hit ratio than PIG0-LRU. This is because the former algorithm replaces pages according to the estimation of their future usage from historical transactions, while the latter simply uses LRU replacement policy, where too little information is considered in making replacement decision.

From this basic experiment, we can see that both PIG0-LRU and PIG0-LPI outperform traditional ANY-LRU. PIG0-LPI, which applys data mining results as much as possible to solve both public buffer placement and replacement problems, comes out the best of the three.

4.2 Experiment 2: the effect of access probability threshold

Experiment 2 studied the impact of access probability threshold by varying the threshold for different buffer sizes. Two sets of the results are presented in Figure 3 where the buffer sizes are 5 and 10% of the database size, respectively. Since ANY-LRU algorithm does not use information related to the threshold, its curves are flat as expected. For PIG0-LRU and PIG0-LPI, there exists a range of thresholds during which the hit ratio is higher. This is also expected: when the threshold is set too lower, pages with less access probabilities are mistakenly assumed as frequently accessed pages, and thus they are treated equally the same as those actually popular ones. On the other hand, when choosing a higher threshold, more pages with relatively reasonable access probabilities are exempted from consideration and given lower priorities in using the buffer.

One interesting observation is that, with high threshold, the performance of two data mining based strategies performs similarly. When buffer size is large and threshold is high, they even perform worse than LRU policy. The reason could be that, when the threshold is high enough, more site's access probabilities to certain pages have simply been abstracted into 0. For each page resident in



(a) public buffer size=5% (b) public buffer size=10%

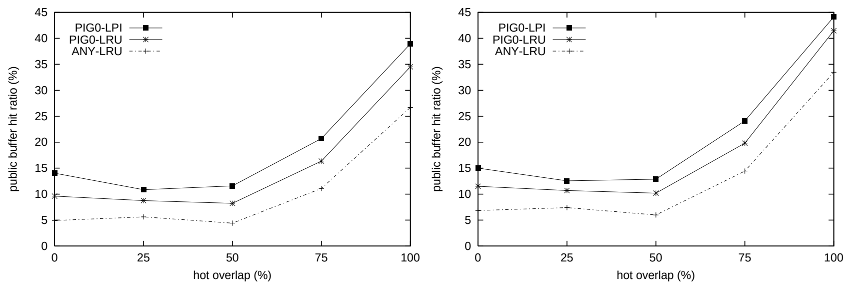
Fig. 3. Public buffer hit ratio versus threshold

the buffer, its total page *interest* can be quickly reduced to 0 along with the access evolution (LPG0-LPI algorithm also sets the corresponding page access probability for a site as 0 once the site accesses this page). On this situation, the LRU replacement policy of PIG0-LPI and ANY-LRU plays a major role. Therefore, the behavior of PIG0-LRU and PIG0-LPI are quite similar at higher thresholds. When buffer size enlarges, since pages passing the threshold can stay longer inside the buffer, they have the chance to endure more references coming from all the sites before being kicked out. So even lower thresholds are adopted at this time, *interest*s of the pages in the buffer may still be decreased into 0 by setting the access probabilities to 0 when database accesses evolve, which thus makes LRU dominate LPG0-LPI's replacement decision as well, just like that in LPG0-LRU.

4.3 Experiment 3: the effect of hot overlap

Experiment 3 examines the impact of access locality by varying *hot_rel_overlap* and *hot_page_overlap* from 0% to 100%. Here, hot overlap=100% means that all users have exactly the same hot relations and pages, and 0% indicates that access from two consecutive sites have totally different hot relations and pages. Sample results for buffer size 5 and 10% are shown in Figure 4.

From the results, we can see that, while the hot overlap between two successive sites is below 50%, the public buffer hit ratios under the three policies hardly change, but from above 50%, all three policies improve their hit ratios greatly. This invariance and hereafter improvement can be explained as follows: With higher locality, accesses from all sites start to concentrate on a smaller number of relations and pages. The reference frequencies to these hot spots are much higher than when the locality is low. For ANY-LRU which places any page into the public buffer as long as private buffers request, the page referenced by users at one site at larger overlaps are more likely to belong to the hot portions of other sites, and it may be accessed by those sites later as well. Therefore, the



(a) public buffer size=7.5% (b) public buffer size=10.0%

Fig. 4. Public buffer hit ratio versus hot relation/page overlap

placement decision made by ANY-LRU tends to be reasonable at larger overlaps. Meanwhile, the LRU replacement policy that ANY-LRU adopts also seems to behave correctly as well, since LRU always chooses the least referenced page to replace and keeps those frequently referenced pages (hot portions) resident in the buffer. This is also the reason why the performance gains of LPG0-LPI and LPG0-LRU over ANY-LRU decrease from 69%, 56% (under hot overlap 50%) to 41%, 34% (under hot overlap 100%) when the buffer size is 5.0%.

5 Conclusion

The work reported here is a preliminary step towards data mining based database buffer management. While the initial results seem promising, there remain a number of interesting problems and opportunities for future work. As we are only concerned with public buffer management issue here, one important future work is to investigate how data mining results can be applied to private buffer management. Another aspect we plan to study is to utilize discovered knowledge to solve both public and private buffer allocation problems. Incremental maintenance of the discovered knowledge is also a critical point worth studying in the future. Finally, as the cross-site buffer invalidation effect is ignored in this paper, it should be interesting to see how it will affect the performance of the system.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 ACM SIGMOD Int'l Conf. on management of data*, pages 207–216, Washington D.C., USA, May 1993.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Conf. on Very Large Data Bases*, pages 478–499, Santiago, Chile, September 1994.

3. Ö. Babaoğlu and D. Ferrari. Two-level replacement decisions in paging stores. *IEEE Transactions on Computers*, 32(12):1151–1159, December 1983.
4. M.J. Carey, R. Jauhari, and M. Livny. Priority in DBMS resource scheduling. In *Proc. of the 15th Conf. on Very Large Data Bases*, pages 397–410, Amsterdam, August 1989.
5. C.Y. Chan, B.C Ooi, and H. Lu. Extensible buffer management of indexes. In *Proc. of the 18th Conf. on Very Large Data Bases*, pages 444–454, British Columbia, Canada, August 1992.
6. M.S. Chen, P.S. Yu, and T.H. Yang. On coupling multiple systems with a global buffer. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):339–344, April 1996.
7. H. Chou and D. DeWitt. An evaluation of buffer management strategies for relational database systems. In *Proc. of the 11th Conf. on Very Large Data Bases*, pages 127–141, Stickhilm, Sweden, August 1985.
8. A. Dan, D.M. Dias, and P.S. Yu. Analytical modelling of a hierarchical buffer for a data sharing environment. In *Proc. of the ACM SIGMETRICS*, pages 156–167, San Diego, CA, May 1991.
9. W. Effelsberg and T. Haerder. Principles of database buffer management. *ACM Transaction on Database Systems*, 9(4):560–595, December 1984.
10. R. Jauhari, M.J. Carey, and M. Livny. Priority-hints: an algorithm for priority-based buffer management. In *Proc. of the 16th Conf. on Very Large Data Bases*, pages 708–721, Brisbane, Australia, August 1990.
11. T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proc. of the 20th Conf. on Very Large Data Bases*, pages 439–450, Santiago, Chile, September 1994.
12. J.P. Kearns and S. DeFazio. Diversity in database reference behavior. *Performance Evaluation Review*, 17(1):11–19, May 1989.
13. D.E. Knuth. *The Art of Computer Programming, Vol.3: Sorting and Searching.* Addison-Wesley, 1973.
14. V.F. Nicola, A. Dan, and D.M. Dias. Analysis of the generalized clock buffer replacement scheme for database transaction processing. *ACM SIGMETRICS and PERFORMANCE*, 20(1):35–46, June 1992.
15. 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.
16. 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, August 1990.
17. G.M. Sacco and M. Schkolnick. A mechanism for managing the buffer pool in a relational database system using the hot set model. In *Proc. of the 8th Conf. on Very Large Data Bases*, pages 257–262, Mexcio City, September 1982.
18. G.M. Sacco and M. Schkolnick. Buffer management in relational database systems. *ACM Transactions on Database Systems*, 11(4):473–498, December 1986.
19. G. Weikum, C. Hasse, A. Mönkeberg, and P. Zabback. The comfort automatic tuning project. *Information Systems*, 19(5):381–432, May 1994.
20. P.S. Yu, D.M. Dias, J.T. Robinson, B.R. Iyer, and D.W. Cornell. On coupling multi-systems through data sharing. *Proc. of the IEEE*, 75(5):573–587, May 1987.

This article was processed using the LaTeXmacro package with LLNCS style