Buffer Replacement Using Online Optimization by Mining

BY

Tung Kum Hoe, Anthony

B.Sc.(Honours), National University of Singapore

A DISSERTATION SUBMITTED
IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR
THE DEGREE OF MASTER OF SCIENCE (COMPUTER
AND INFORMATION SCIENCE)
NATIONAL UNIVERSITY OF SINGAPORE

May 1998

ACKNOWLEDGMENTS

First of all, I want to thank my family for their support and encouragement throughout this period.

I like to thank both my supervisors, Assoc. Prof. Lu Hongjun and Dr Tay Yong Chiang for their advises and guidances. They have been good teachers and mentors who provide me as much help as they can in various ways. Their patience when progress was slow is greatly appreciated.

I would like to thank Aik Hiang, Boon Tun, Chee Wai, Chia Yeow, Chin Yew, Chung Pheng, Joseph, Kian Sing, Kwang Tat, Liang Chye, Seow Yong, Tong Kiat and Wee Yeh for providing company for me during breakfast, lunch, dinner and supper. I have learn a lot from the discussion of your research topics during meal time and beyond.

I would also like to thank Lawrence, Wee Kee and Wei Kai who came down to visit me in the campus whenever they are free from their sales job.

Special thanks to Fan Weiguo for passing me the latex template used in writing this thesis.

Tung Kum Hoe, Anthony

CONTENTS

ACKNOWLEDGMENTS i

CONTENTS ii

LIST OF FIGURES iv

SUMMARY vi

1 Introduction 1

1.1 Related Works 2

1.1.1 Buffer Replacement Algorithms 2

1.1.2 Theoretical Analysis of Paging Algorithms 8

1.1.3 Association Rules in Data Mining 11

1.2 Motivation 13

2 System Model 14

2.1 Derivation of System Model 14

2.1.1 System Architecture 14

2.1.2 Common Properties in Database Access 15

2.2 The Extended Access Graph Model 18

3 Mining the Training Stream 21

3.1 Types of Knowledge 21

3.2 Rules Extraction 23

3.2.1 Mining Advanced Rules: Parameters 24

3.2.2 Mining Advanced Rules: Algorithm 28

4 BROOM: Applying the Rules to Online Stream 30

4.1 Data Structures 30

4.2 Monitoring and Activation of Advanced Rules 32

4.3 Page Replacement 34

5 Simulation and Experiments 38

5.1 Simulation 38

5.2 Experiments 41

5.2.1 Setting the Mining Parameters 41

5.2.2 Small Buffers, Large Buffers 43

5.2.3 The Effect of System Parameters on BROOM's Hit Rate 43

6 Conclusion 58

6.1 Issues of concern 58

6.1.1 BROOM's Overheads in Space and Time 58

6.1.2 Frequency of Mining 60

6.2 When to use BROOM? 60

6.3 Future Work 62

LIST OF FIGURES

1-1 An access graph representing two types of transactions 10

1-2 An extended access graph representing two types of transactions 11

2-1 System Architecture. 15

2-2 Embedded Pattern within Different Transactions. 16

2-3 Extended Access Graph with Multipointers. 20

3-1 BROOM's algorithm for mining basic rule. 24

3-2 A reference stream of length ℓ_{train}=10 has 6 windows of size $W = 5$ 25

3-3 Records generated after chopping 27

3-4 Mining advanced rule. 29

4-1 Structure of a node for storing S_i. 31

4-2 BROOM's data structure for storing advanced rules. 31

4-3 BROOM's algorithms for updating data structures. 35

4-4 BROOM's page replacement algorithm. 37

5-1 Simulator 's algorithm for generating patterns. 39

5-2 Simulator's algorithm for generating reference streams. 40

5-3 Values of parameters in the experiments. 42

5-4 Hit rates for varying buffer sizes 44

5-5 Varying n_{local} with n_{buff}/n_{dbpage} =0.02 47

5-6 Varying n_{local} with n_{buff}/n_{dbpage}=0.100 48

5-7 Varying n_{local} with n_{buff}/n_{dbpage}=0.200 48

5-8 Varying n_{appear} with n_{buff}/n_{dbpage} =0.02 50

5-9 Varying n_{appear} with n_{buff}/n_{dbpage}=0.100 50

5-10 Varying n_{appear} with n_{buff}/n_{dbpage}=0.200 51

5-11 Varying r_{random} with n_{buff}/n_{dbpage} =0.02 52

5-12 Varying r_{random} with n_{buff}/n_{dbpage}=0.100 52

5-13 Varying r_{random} with n_{buff}/n_{dbpage}=0.200 53

5-14 Varying $n_{pattern}$ with n_{buff}/n_{dbpage} =0.02 54

5-15 Varying $n_{pattern}$ with n_{buff}/n_{dbpage}=0.100 55

5-16 Varying $n_{pattern}$ with n_{buff}/n_{dbpage}=0.200 55

5-17 Varying $n_{terminals}$ with n_{buff}/n_{dbpage} =0.02 56

5-18 Varying $n_{terminals}$ with n_{buff}/n_{dbpage}=0.100 57

5-19 Varying $n_{terminals}$ with n_{buff}/n_{dbpage}=0.200 57

6-1 General Framework of BROOM 64

SUMMARY

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 use heuristics to determine which page to replace when there is a page fault, this thesis presents a different approach which uses knowledge mined from historical page references to do buffer replacement. We call this new approach, BROOM (Buffer Replacement using Online Optimization by Mining),

The approach of BROOM 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.

Chapter 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*. It is a classical problem which has been extensively studied.

Data mining on the other hand is a emergent research field which represents the integration of several research fields, including machine learning, database systems, data visualization, statistics and information theory. It is a new technology for discovery of *interesting, implicit, and previously unknown knowledge from large databases* [11]. It has a wide area of application especially in the business and financial world where the knowledge discovered are used by managers and executives to formulate their business policies and strategies.

In this project a new algorithm for buffer replacement is developed. We called this new algorithm BROOM (Buffer Replacement using Online Optimization by Mining). BROOM makes use of knowledge that are mined from historical reference streams to do buffer replacement. The development of BROOM adds a new dimension to the application of data

mining, which is the extraction of knowledge for usage by programs rather than humans. It is hoped that this project could become a motivation for others to look further into this dimension.

1.1 Related Works

Related works in this project can generally be separated into 3 categories:

1. Buffer replacement algorithms.
2. Theoretical analyses of paging algorithms.
3. Association rules in data mining.

We will have a look at these categories in more details.

1.1.1 Buffer Replacement Algorithms

The simplest replacement algorithm is First In First Out (FIFO). As the name implies, FIFO throw out the first page that goes into the buffer. Due to its simplicity, FIFO is usually the worst among all replacement algorithms. Among all the others existing buffer replacement algorithms, the Least Recently Used (LRU) [4, 9, 26] is the most popular among commercial systems. LRU replaces the buffer page that has not been accessed for the longest time. Besides LRU, other improved replacement algorithms are as follow :

i) The CLOCK and GCLOCK Algorithms

In the CLOCK algorithm, an use-bit is added to every buffer page, indicating whether the page was referenced during the recent circulation of the selection pointer. The replacement page is determined by the stepwise examination of the use-bits. If a 1-bit is encountered,

the bit is reset to 0 and the selection pointer will be moved to the next page. The first page that is found with a 0-bit will be replaced by the incoming page. As can be seen, the CLOCK algorithm actually simulates the LRU and empirical study conducted in [10] confirms the similarity of the buffer hit rates between LRU and CLOCK.

In the Generalized CLOCK (GCLOCK) algorithm, a counter is associated with each buffer page. When a page is brought into the buffer, the counter is assigned an initial value(weight) that may be different for different types of data pages. When a page hit occurred on a page, the weight of the page will be increased or reset to a certain value. To find a replacement page during a page fault, a circulating clock pointer sweeps through the buffer pages decrementing their counters until a page with a count of zero is found. This page is selected to be the replacement age. It was shown in [24] that GCLOCK outperforms the CLOCK algorithm with an appropriate choice of weights.

ii) The Least Reference Density Algorithm

The Least Reference Density (LRD) algorithm was derived from GCLOCK, it makes use of 3 variables:

RC(i): The number of times a buffer page i is referenced since it is placed inside the buffer.

GRC: A global reference counter which increases for every page reference.

FC(i): The value of GRC when a buffer page i is first placed.

The reference density of a page i, $RD(i)$ can be calculated using the following formula:

$$RD(i) = \frac{RC(i)}{GRC - FC(i)}$$

When a replacement page is required, the buffer page with the lowest $RD(i)$ will be

replaced.

This approach however has a problem because it assumes equidistance arrival of page references. To see this problem, consider 2 buffer pages, page i and page j which are referenced the same number of times during the same reference interval. The references to page i however are clustered at the beginning of the interval while references to page j are clustered at the end of the interval. In this situation, page i should be preferred over page j if one of them have to be replaced. However, LRD will not exhibit such preference because *RD(i)* is equal to *RD(j)*. To overcome such a problem, the reference counters RC of all buffer pages are reduce by subtraction or division at the end of a certain interval.

Performance studies in [10] show that LRD method is the best policy among LRU, CLOCK and GCLOCK.

iii) The Frequency-Based Algorithm

Like LRD, the Frequency-Based Replacement (FBR) strategy [27] makes replacement choice using a combination of reference frequency and page age. However, the FBR algorithm factored out locality from reference counts. Such an approach is motivated by the fact that certain pages are often repeatedly re-referenced within a short interval, giving them high reference counts and thus causing them to be kept longer then desired in the buffer.

The basis of FBR is the LRU replacement algorithm. In FDR the buffer is seen logically as a stack of pages. The recently referenced pages are kept on top of the stack while the least recently referenced ones are kept at the bottom of the stack. consists of a stack of pages logically, with the most recently reference pages always kept on top of the stack. The stack is separated into 3 sections, new section, middle section and old section. The new section is at the top part of the stack. If a page is in the new section and is referenced, its

reference count will not be increased. Because of this, reference counts for pages that are repeatedly referenced within a short interval will remain unchanged throughout the interval. On the other end of the stack is the old section. When a page fault occurs, the page with the lowest reference count in this section will be selected for replacement. The remaining part of the stack between the new and old section is the middle section. This section gives pages that are moving out of the new section a chance to build up their page count before becoming eligible for replacement.

Simulation results based on traces of file systems and I/O activities from actual systems show that FBR performs consistently better than LRU.

(iv) The LRU-K Algorithm

Although LRU has been the most popular among the commercial systems, it is often pointed out by critics that LRU is making use of too little information to make replacement decisions. In particular, it was criticized that LRU is unable to differentiate between pages that have relatively frequent references and pages that have very infrequent references. Once a page is placed inside the buffer, LRU guarantees it a long buffer life even if the page will never be referenced again.

To solve this problem, [26, 25, 30] proposed a strategy named LRU-K which makes use of more of the access history of the pages to make replacement decision. The basic idea is to keep track of the times of the last K references to the pages and replace the page whose time of access for the K^{th} reference in the past is most distant. Like FBR, LRU-K has a mechanism for factoring out locality. A page will not be dropped immediately after its first reference but will be kept a short period until the possibility of a dependent follow-up reference is minimal. This short period serves the purpose of the new section in FDR.

Meanwhile, repeated references to the same page within a period (called the Correlated Reference Period) are considered as correlated accesses and will be collapsed into one access. Upon dropping a page from memory, LRU-K will retain its information for a period (called the Retained Information Period) in case this page is referenced quickly again.

Both simulation and empirical studies had been done on LRU-2. Results showed that it outperformed conventional algorithms like LRU, sometimes as high as 40%.

(v) The 2Q Algorithm

Although LRU-K can significantly improve on LRU by taking into account the last K references to a page, a great processor overhead is incurred to track the reference history of each page. To solve this problem, [20] presented a new algorithm called 2Q, which behaves as well as LRU-2 but has constant time overhead.

Given a buffer, 2Q divides it into 3 queues: Am, A1in and A1out. Am queue is an LRU queue while both A1in and A1out queues are FIFO queues. When a page is referenced for the first time, it is first put at the end of A1in to factor out locality. If the length of A1in exceeds a certain threshold, the page at the head of A1in is then removed from A1in and put at the end of A1out. Pages in Aout are pages which are accessed once and which are assumed to have no more correlated references. Hence if a page in A1out is referenced again, it is moved to the end of Am as a hot page.

When a page fault occurs, the length of A1out is first checked. If this length exceeds a certain threshold, the page at the head of A1out is dropped because this page has been referenced only once and has not been re-referenced for quite a long time (long enough for the length of A1out to exceed the threshold). If Aout does not exceed the threshold length, then the least recently used page in Am is selected for replacement.

Experiments show that 2Q not only behaves as well as LRU-2. It is also simpler, requires no tuning and executes faster.

(vi)The Priority-Based Algorithms

The replacement algorithms that are discussed in the earlier sections are global algorithms in which the replacement discipline is applied globally to all the buffers in the system. These algorithms have very little or no knowledge of the structure of the database and have no concept of database transactions. Unlike these algorithm, priority-based algorithms tried to makes use of the concept of transaction or database structure to assign priorities to different pages and use the priorities of the page to make replacement decision.

Based on the conventional Global LRU and DBMIN policy, two priority buffer management algorithms, Priority-LRU and Priority-DBMIN were proposed in [7]. The key idea behind these two algorithms is to choose the least recently used page of the lowest priority for replacement. A further strategy Priority-Hint was developed in [19] in which priority hints are provided by the methods used in the transactions to access the database. Instead of using database access methods, [8] used index method to provide priority hint for the buffer manager. Experimental results indicate that priority-based algorithms outperform commonly used, simple LRU-based algorithms.

(vii)The Data Mining-Based Approach for Distributed Database Systems

Inspired by the recent development in data mining and observing that LRU-K and 2Q algorithms outperform other strategies because they use more information about page references than other strategies, Feng *et al* initiated a study on the use of data mining for buffer management in [12]. The basic idea described in [12] is to mine the database access histories to extract knowledge for the management of public buffers in a distributed database. The

buffer manager will then use the extracted knowledge to estimate the *page interest* of a page. When the page interest of a page is high, it means that many sites are "interested" in this page and the page will be placed in the public buffer. To make space for the page, a page with the least interest in the public buffer will be evicted.

Experiments by simulation show that the data mining-based approach outperforms a policy which put all pages into the public buffer and uses LRU as a replacement policy.

1.1.2 Theoretical Analysis of Paging Algorithms

While the term "buffer replacement" is used mainly in the database context, the term "paging" can be used in any context where a faster memory is used to hold frequently accessed pages from a slower memory to improve efficiency. A paging algorithm does exactly the same task of a buffer replacement algorithm.

Much theoretical analyses have been done on paging. Our focus here is not on the result of these analyses but on the way the page references are modeled in these theoretical works. Early work on evaluating paging algorithms focus on average case analysis, assuming that the request sequence is drawn from a probability distribution [5, 29]. Such an approach however is not robust as the results are heavily dependent on the probability distribution.

In [15], a worst case analysis is done on LRU and FIFO for the case where the input is drawn from an arbitrary probability distribution. The "worst-case" input in such a scenario will cause any paging algorithm to fault on every request, a result that is seldom encountered in practice.

Sleator and Tarjan [28] proposed the alternative of competitive analysis: let $A(\sigma)$ be the number of faults made by a paging algorithm A request sequence σ and $OPT(\sigma)$ be the number of faults made by an optimal algorithm that have the entire sequence σ in advance.

Algorithm A is said to be *c-competitive algorithm* [21] if for all σ, $A(\sigma) - c.OPT(\sigma)$ remains bounded by a constant. The *competitiveness* of A, c_a, is the infimum of c such that A is *c-competitive*. Such a model of analysis is more robust than probabilistic analysis as it does not assume any underlying distribution in the page references. At the same time, it is also more useful than worst case analysis because paging algorithms can be compared based on their relative performances to the optimal algorithm. With this model Sleator and Tarjan showed that no online paging algorithm can achieve a competitiveness less than k where **k** is the number of pages in the fast memory.

Practitioners voice at least two reservations about the analysis in [28]. Firstly, the analysis does not make a distinction between LRU and FIFO, whereas in practice LRU is better than FIFO most of the time. Secondly, the analysis suggests that the number of faults by LRU on a sequence could be k times the optimal number of faults, whereas in practice, the ratio is often much smaller. Borodin et al in [6] showed that these concerns can be addressed by augmenting the Sleator-Tarjan approach with the *access graph model*.

An access graph G= (V,E) is a graph in which each node corresponds to a page in the slow memory. The generation of a request sequence is subjected to locality imposed by the constraint of the edges of G, that is, following a request to a page(node) u, the next request must be either to u or to a node v such that (u,v) is in E. Thus a sequence σ is a walk in the access graph. Analyses using such a model not only provide a tighter upper bound for c_{LRU} but also show that the c_{LRU} is much smaller than the c_{FIFO} for most access graphs. By assuming that the access graph is known in advance, Borodin et al defined a new algorithm called FAR and prove that it is within a logarithmic factor of the optimal. The main idea behind FAR is to imitate the optimal offline policy in [5] which will replace the page that next use is furthest in the future. [18] went on to prove that FAR is in fact

strongly competitive, that is, within a constant factor of the optimum.

To model a multiprogramming environment, Fiat and Karlin [13] used multiple pointers into the access graph to represent a situation in which there are multiple threads of control. During construction of the sequence, any page that is indicated by a pointer can be selected. A strongly competitive algorithm called MPAlg was defined for such a model. The access graph model however has some troubling aspects. The access graph has to be known in advance to the paging algorithm and the memory required to represent the access graph itself may be very large. It is also unable to deal with changing behavioral patterns. For example, consider a page P_1 that can be accessed by two different types of transaction. The first type of transaction always access P_2 before P_1 and P_3 after P_1. The second type of transaction on the other hand will access P_4 before P_1 and P_5 after P_1. The access graph for the two types of transaction is shown in Figure 1-1. Such a representation however is not correct because two new types of transaction are created: (P_2, P_1, P_5) and (P_4, P_1, P_3).

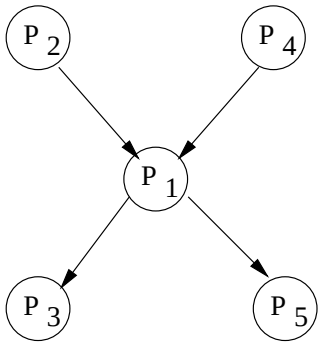

Figure 1-1: An access graph representing two types of transactions

To overcome these problems, Fiat and Medel [14] proposed the *extended access graph* model. In an extended access graph, two different nodes can be used to represent the same page. As such, the two transactions in our earlier example can be represented correctly by an extended access graph shown in Figure 1-2. They also defined a new algorithm

D_{DET} that constructs the extended access graph online and uses the constructed graph for replacement decision. When it is detected that the reference behavior has changed, a new graph will then be constructed. D_{DET} was shown to be strongly competitive for any extended access graph G with $\triangle(G) < k$, $\triangle(G)$ being defined as the minimal number of different pages that are accessed along any path between any two vertices that are mapped to the same page in the graph G. The condition means that the reference behavior must not change so fast that a constructed graph becomes invalid once it has been constructed.

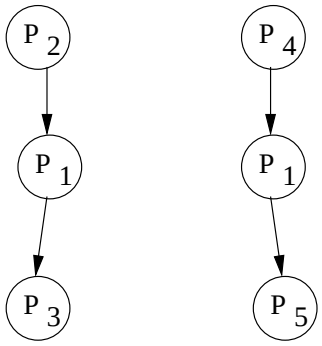

Figure 1-2: An extended access graph representing two types of transactions

1.1.3 Association Rules in Data Mining

In data mining, there are many different types of rules which can be extracted with different techniques. In this report, we are only interested in a technique used for discovering association rules. The problem of mining association rules was introduced in [3] using an example in the supermarket. In the supermarket, bar-code technology has made it possible to collect the so called basket data that store items purchased on a per-transaction basis. An example of an association rule that can be mined from these basket data will be "30% of the transactions contain bread and butter, out of these transactions, 90% purchase milk".

A formal definition of the problem is as follow: Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items

such that $T \subseteq I$. We say that a transaction T contains X, a set of some items in I, if $X \subseteq T$. An association rule is an implication of the form $X \Longrightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = 0$. The rule $X \Longrightarrow Y$ holds in the transaction set D with *confidence* level c if $c\%$ of transactions in D that contain X also contain Y. The rule $X \Longrightarrow Y$ has a *support* s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$. The problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called $minsup$) and minimum confidence (called $minconf$). The problem of discovering association rules can be decomposed into two subproblems [3]:

1. Find all sets of items (itemsets) that have transaction support above mincof. The support for an itemset is the number of transactions that contain the itemset. Itemsets with minimum support are called frequent itemset. A fast algorithm called *Apriori* algorithm was developed by Agrawal and Srikant in [1] for solving this task.

2. Use the frequent itemsets to generate the desired rules. A straightforward algorithm for this task is as follow: For every frequent itemset l, find all subsets of l. For every such subset a, output a rule of the form $a \Longrightarrow (l - a)$ if the ratio of support$(l - a)$ to support(a) is higher than $minconf$. Such an approach however is slow and a faster algorithm by Agrawal and Srikant was described in [2].

Although the Apriori algorithm was developed for doing market basket analysis, its application is not limited to that. We will show how it can be used for mining page reference sequence in this thesis.

1.2 Motivation

The development of BROOM is mainly motivated by two factors.

The first factor is the existence of optimal offline algorithm [5]. By imitating this optimal algorithm, theoreticians had derived algorithms which are proven to be strongly competitive and robust [6, 18, 14]. The algorithm described in [6] however, assumed too much knowledge of the access graph and thus has a very large memory requirement. On the other hand, the algorithm derived in [14] assumed it has no knowledge of the access graph at all. As a result, the algorithm is unable to cater to reference sequences which have fast changing behavior. Guided by these observations, our approach in designing BROOM tried to avoid these two extremes. In BROOM, past reference sequences are mined to extract knowledge about the frequently visited portion of the extended access graph. BROOM will then try to imitate the optimal offline algorithm by using this knowledge.

The second motivation factor for developing BROOM is the existence of stable reference patterns in databases transactions over a long period of time [22]. This mean that the knowledge which is extracted by mining will remain valid for future accesses.

The remainder of this thesis is organized as follows. Chapter 2 examines the reference patterns that are likely to occur in a database and describes a system model for devloping BROOM. An explanation will then be given on how the system model is actually related to the extended access graph. Chapter 3 describes how knowledge is mined from historical accesses, while Chapter 4 explains how the extracted knowledge is used to implement the replacement policy. Chapter 5 describes a simulation model for studying BROOM's performance, and show some results obtained by experiments that are done on the simulator. Chapter 6 concludes the thesis by discussing some issues on using BROOM, stating the conditions under which BROOM works well, and giving some direction for future work.

Chapter 2

System Model

In this chapter, we will have a look at the system model on which BROOM is developed. We will first discuss how the model is derived and then present an overview of the system model by using an extended access graph.

2.1 Derivation of System Model

2.1.1 System Architecture

DBMSs can be used in many different environments and system architectures. Here, we choose the common client-server architecture for the development of BROOM. The architecture is simplified as shown in Figure 2-1.

In such a system, we have a database server that receives page requests from a number of terminals. Upon receiving a page request, the buffer manager, which resides at the database server, will first check for the page in a shared buffer. If the page is not found, it will be fetched from the disk and put into the buffer. The page will then be send to the terminal that request for it. It is assumed that the communication between the server and the terminals is handled by another software module and that the buffer manager has no way of identifying the terminal that issued a particular page request. Terminals are assumed to be homogeneous, that is, they have the same speed, priority and access patterns. At any one time, only one transaction is active at a terminal. Page requests are send to the server on a single-page basis.

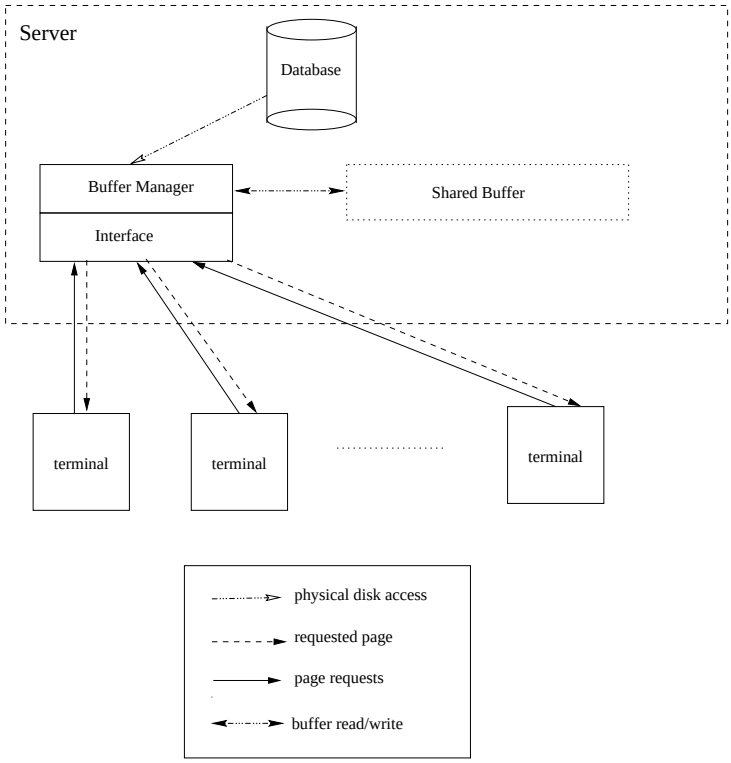

Figure 2-1: System Architecture.

2.1.2 Common Properties in Database Access

During the derivation of the system model, certain common properties in database access are assumed.

The first property that we assumed is the existence of the so-called *hot* pages that are more frequently accessed than others. Examples of hot pages include the system category pages and the top levels of B-tree index pages. Due to their high access frequency, an increase in the hit-rates of hot pages will cause a significant increase in the overall hit rate.

The second property we assumed is the existence of reference patterns within database access sequence. These reference patterns can be divide into 2 categories. The first category is common access patterns that are embedded within different transactions. For example, consider the three transactions that are shown in Figure 2-2, a common pattern that is embedded within them will be the page access sequence $\{P_1, P_3, P_7, P_9, P_{15}\}$. The second

category is patterns which occurred between transactions. An example will be a reference pattern in which a transaction A always occur after a transaction B

Transaction 1: $\{\mathbf{P_1}, \mathbf{P_3}, \mathbf{P_7}, \mathbf{P_9}, \mathbf{P_{15}}, P_{20}, P_{15}, P_3, P_6, P_{19}, P_{14}, P_5, P_8, P_{23}\}$

Transaction 2: $\{P_{10}, P_2, P_8, P_9, P_{12}, P_{21}, \mathbf{P_1}, \mathbf{P_3}, \mathbf{P_7}, \mathbf{P_9}, \mathbf{P_{15}}\}$

Transaction 3: $\{P_9, P_{25}, P_{10}, P_2, \mathbf{P_1}, \mathbf{P_3}, \mathbf{P_7}, \mathbf{P_9}, \mathbf{P_{15}}, P_{18}, P_4, P_8\}$

Figure 2-2: Embedded Pattern within Different Transactions.

As an early study by Kearns and DeFazio [22] shows that database reference patterns exhibited by a transaction change very little over time, we assumed that these common reference patterns are stable over a long period of time.

We now analyze in greater detail reference patterns in DBMSs. Our analysis focus on two points:

- The sources for the patterns which are namely the database system and user behavior
- The two types of patterns: *sequential* and *locality* patterns [22]. 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.

i) Reference Patterns Induced by Database Systems

A database system affects reference patterns in many ways; the following are some of them:

Sequential scan of a relation: One relation's pages will often be accessed close to each other and hence they come together to form a reference pattern. If no index or a clustered index is used to search the relation, the pattern generated is likely to be sequential.

Index scan using clustered index: When a portion of a relation is accessed via a clustered index, index and data pages are often accessed in a sequential fashion: a number of index pages (from the root to the leave along a branch) followed by a sequence of data pages.

Index scan using non-clustered index: When a portion of a relation is accessed using a non-clustered index, it will result a locality pattern that is determined by the distribution of records over the data pages of the relation. That is, a data page will be repeatedly accessed a number of times, corresponding to the number of entries in the index that refer to the data page.

Page accesses during joining operation: Pages of relations that are often joined will also induce some reference patterns. For example, a nested loop join may result in patterns that depend on the length of the inner loop and the size of the buffer. If the length of a loop is much bigger than the buffer size, the reference pattern can be treated as a concatenation of sequential patterns that repeat some number of times; if it is much smaller, one can consider the reference pattern as a locality pattern.

Execution plans of query language: Most DBMSs support non-procedural query languages. The execution plans that determine the sequence of page references are generated by the DBMS itself rather than coded by programmers. Therefore, the page access sequence for a query will be the same even if it is submitted from different terminals.

ii) 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 accesses a client's financial information to make her decision, then proceeds 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.

2.2 The Extended Access Graph Model

To give an overview of the system model, we use an extended access graph with multiple pointers [14] which is shown in Figure 2-3. In the figure, we have seven reference patterns which occurred frequently. These patterns are *frequently accessed region* of the extended access graph. Two types of reference patterns are present in the graph, locality and sequential patterns. In the figure, pattern 1,3 and 6 are locality patterns while pattern 2,4,5 and 7 are sequential patterns.

The *random region* in the center represents page accesses outside the frequently access region and can be infinitely large. To model concurrency, we use multiple *terminal pointers* that represent the terminals. These pointers will move along the graph, generating a page request for each node that they visit. Because we assumed that these terminals have equal priority and speed, the probability of a terminal pointer moving is the same for all terminals.

While the page request sequence is generated using such a model, it is assumed that BROOM does not have this detail knowledge of the extended access graph. Instead, BROOM will try to mine out knowledge of the frequently access pattern from the page

request sequence. In the next chapter, we will have a look at the type of knowledge that BROOM will extract from the page request sequence. Then we will discuss the algorithm for mining this required knowledge

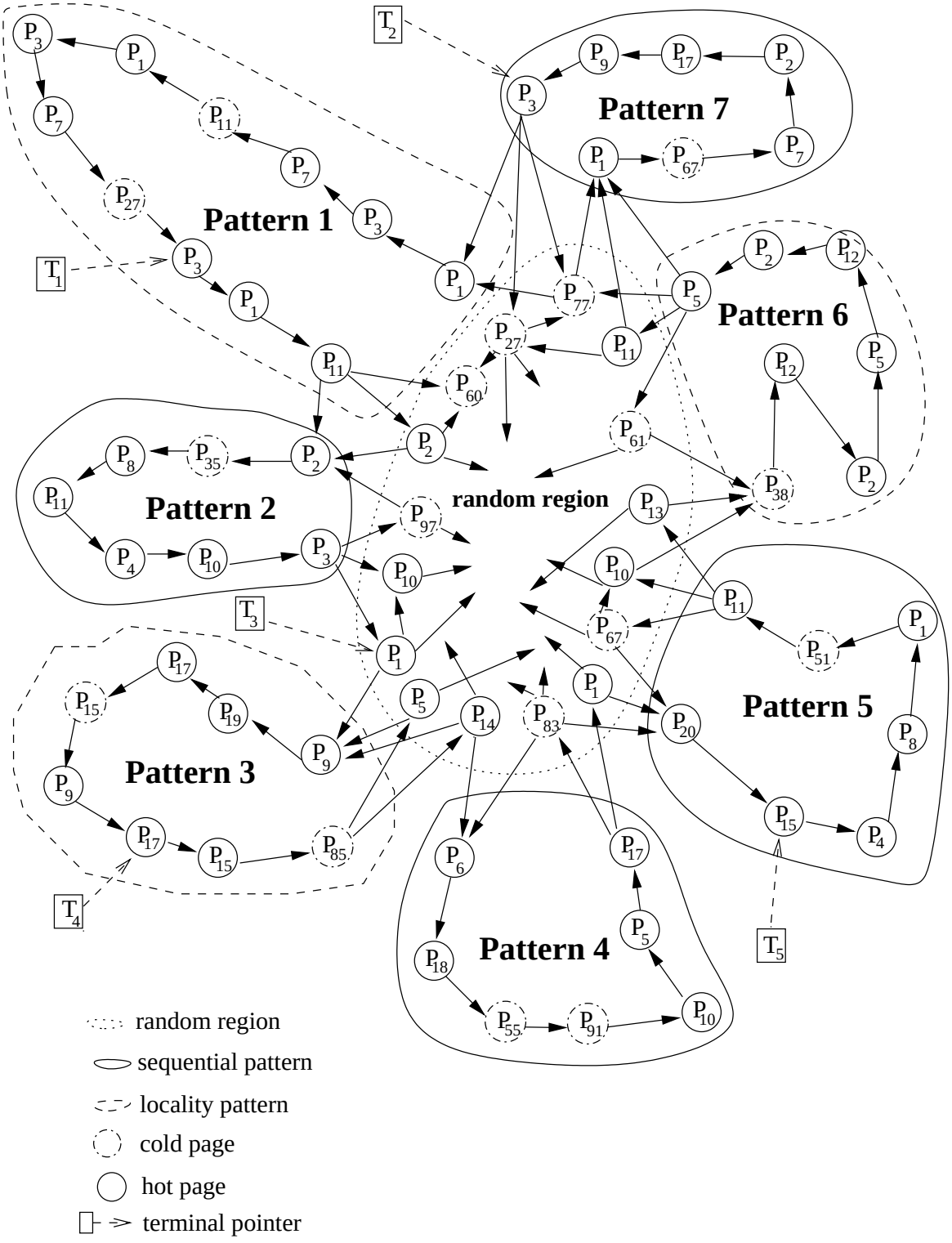

Figure 2-3: Extended Access Graph with Multipointers.

Chapter 3

Mining the Training Stream

In this chapter, we will have a look at the types of knowledge that BROOM uses and how such knowledge is extracted from historical page access sequence or *training stream*.

3.1 Types of Knowledge

As mentioned earlier, the basic idea behind BROOM is to imitate the offline optimal strategy which replaces a page that will be accessed furthest down the reference sequence during a page fault. To do so, BROOM makes use of rules extracted from the training stream to predict the next request to a page in the buffer. It then replaces the page that is predicted to be accessed furthest down the reference sequence. Two types of rules are used in BROOM: *basic rules* and *advanced rules*.

Basic rules are of the form $P_i \xrightarrow{D_{ii}} P_i$ where D_{ii} denotes the expected *distance* between two consecutive occurrences of a physical page P_i. Here, we define *distance* between two page requests as the number of references that occurred between the two requests. If only basic rules are used, 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 page fault occurs; thus, the page that has the largest D_{ii} (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}.

However, Using only 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 average distance between $\{P_{i_1}, ..., P_{i_k}\}$ and the next occurrence of P_i.

To see some examples of advanced rules, consider the reference stream

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

We will have the following three advanced rules,

$$\{P_9, P_{20}\} \xrightarrow{3} P_9, \qquad \{P_9, P_4, P_{11}\} \xrightarrow{1} P_{33} \qquad \text{and} \qquad \{P_9\} \xrightarrow{4} P_{33};$$

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*. Thus, $\{P_9, P_{20}\}$ and $\{P_9, P_{20}, P_4\}$ are 2-item and 3-item sets, respectively.

To see the idea behind advanced rules, let us go back to Figure 2-3. Consider the case in which we only have the graph but do not have any idea on where the terminal pointers are. To do prediction, we will have to first guess the position of the terminal pointers by looking at the reference stream. For example, if we see a request for P_1, we might guess

that we are in pattern 1 or pattern 7. However, making such a deduction from one page request is usually incorrect since P_1 may also come from the random region. To make a better prediction, we have to observe the past few page requests instead. For example, if we see P_1 and P_3 in the past few page requests, we can conclude that we are in pattern 1. Astute readers will point out that this is also not a valid conclusion because of the following cases:

Case 1: Requests for P_1 and P_3 may be generated by different terminal pointers.

Case 2: Requests for P_1 and P_3 may be generated from the random region.

Case 3: P_1 and P_3 may occur in many other frequent patterns.

The probability of case 1 occurring however will be very low since the page requests send by 2 terminals are independent of each other. The occurrence of case 2 is by definition unlikely because frequent occurrence of case 2 will move the sequence $\{ P_1, P_3 \}$ into the frequently access region. Case 3 is a valid concern which means that we must either take more page requests into consideration or choose a different "hint" completely.

Following the above discussion, we can see that the antecedent of an advance rule is actually a "hint" for the arrival of the page at the consequent. As such when we are mining for advance rules, we are actually looking for sets of pages which provide a good "hint" for the arrival of other pages.

3.2 Rules Extraction

The algorithm for the extraction of basic rules is trivial and is shown in Figure 3-1. We shall focus on the extraction of advanced rules in this section.

```
MineBasic()                              // how basic rules are mined
{
  for each database page P_i {
    scan the reference stream to find distances (#pages) between consecutive accesses to P_i;
    D_ii ← average distance;
    σ_ii ← standard deviation of distances;      // used for advanced rules
    output basic rule: P_i --D_ii--> P_i;
  }
}
```

Figure 3-1: BROOM's algorithm for mining basic rule.

3.2.1 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:

i) Windows Size

As mention earlier, to identify the positions of the terminal pointers in the access graph, BROOM needs to analyze the past few page requests. However, the number of past page requests being analyzed must not be too large. This gives rise to the concept of a *window* in which the window size W correspond to the number of past requests being taken into consideration. During mining, we moved this window over the reference stream, transformed the stream into a set of records, each of which contains the pages covered by the window. For example, with a window of size W, a reference stream of length $\ell_{\text{train}}$is transformed into $\ell_{\text{train}} - W + 1$ records of W accesses each, and item sets have maximum size W. In Figure 3-2, a window of size 5 will divide the reference stream of length 10 into 6 records, W_1 to W_6.

By using windows, we restrict advanced rules to only those whose S_j appears in windows.

Reference Stream:	$P_9,\ P_{20},\ P_4,\ P_7,\ P_9,\ P_{11},\ P_{33},\ P_{11},\ P_{19},\ P_{50}$
W_1:	$\{P_9,\ P_{20},\ P_4,\ P_7,\ P_9\}$
W_2:	$\{P_{20},\ P_4,\ P_7,\ P_9,\ P_{11}\}$
W_3:	$\{P_4,\ P_7,\ P_9,\ P_{11},\ P_{33}\}$
W_4:	$\{P_7,\ P_9,\ P_{11},\ P_{33},\ P_{11}\}$
W_5:	$\{P_9,\ P_{11},\ P_{33},\ P_{11},\ P_{19}\}$
W_6:	$\{P_{11},P_{33},P_{11},\ P_{19},\ P_{50}\}$

Figure 3-2: A reference stream of length ℓ_{train}=10 has 6 windows of size $W = 5$

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 $n_{\text{terminal}}$is the number of terminals and $h_{\max}$ is the maximum size of antecedent, then we set W to a value of $n_{\text{terminal}} \times h_{\max}$. This choice of W is large enough such that with the interleaving of references from the transactions there are, on average, $h_{\max}$ references from each terminal in every window. It is however not possible to set one value for h_{max} because its value is heavily dependent on the nature of the access graph. Thus, we took the same approach in selecting a value for k in the LRU-k algorithm, which is to increase h_{max} until there is no significant improvement in performance. We also note that it is possible for some terminals to remain idle for a period of time (due to user think time etc.), but it will just mean that more pages are taken into consideration from the remaining active terminals. This will not bring down the accuracy of our mining.

ii) Minimum Support Level $MinSup$

The second parameter for mining is the Minimum Support Level, $MinSup$. 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*. Intuitively, it means that

these rules must occur frequent enough so that they are likely to be from the frequently access region of the access graph.

The support level of S_j and P_i are defined as follows:

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

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

In Figure 3-2, the support level of item set $\{P_4, P_7\}$ is 3/6 or 50%. In mining terminology, item sets with support levels greater than *MinSup* are called *frequent sets*. Therefore, in this case, we say that advanced rules are only kept for frequent sets.

We use different support levels for frequent sets of different sizes. Hence if $h_{\max}$ has a value of 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.

The idea behind support level is that the appearance of the set in the reference stream is more than random. One could in theory, work out the formula for the expected number of occurrences of an h-item set in all the records of size W obtained from a stream of random references, and use this as the support level. Instead, we simply generate a (uniformly) random reference stream, extract the records, and take the average frequency of a randomly generated h-item set; this frequency is then used as $MinSup_h$.

The advanced rules must pass a statistical test before they are used. This test is based on the Central Limit Theorem and requires that the sample size for a frequent set must be "large"; most textbooks consider 30 to be large enough for the normal approximation to hold [23]. Hence, if $MinSup_h$ is so low that $MinSup_h \times \ell_{\text{train}} < 30$, then either ℓ_{train} must be increased or $MinSup_h$ should be set to $30/\ell_{\text{train}}$.

Record 1:	$P_9,\ P_{20},\ P_4,\ P_7$
Record 2:	$P_{20},\ P_4,\ P_7,\ P_9,\ P_{11}$
Record 3:	$P_4,\ P_7,\ P_9,\ P_{11},\ P_{33}$
Record 4:	$P_7,\ P_9,\ P_{11},\ P_{33}$
Record 5:	$P_9,\ P_{11},\ P_{33},\ P_{19}$
Record 6:	$P_{11},\ P_{33},\ P_{19},\ P_{50}$

Figure 3-3: Records generated after chopping

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 $MinSupZ$.

We set $MinSupZ$ by considering the skew distribution of references over the database pages. Suppose we have a database in which 80% of the database references are to 10% of the database pages. To determine $MinSupZ$, we can first measure the frequency of each page in the stream. We then add up the frequencies of the hottest pages until we get 80% of the accesses, and set $MinSupZ$ to be the frequency for the last page that is added.

iii) Significance Level α

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 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 $\alpha = 0.99$.

3.2.2 Mining Advanced Rules: Algorithm

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

i) Identifying Frequent Sets

This can be further divided into two steps:

Step 1: Divide the training stream into records, then remove repetitions from these records. For Figure 3-2, the records generated are shown in Figure 3-3.

Step 2 Use Apriori algorithm [3] to extract frequent sets from these records. The support level should be set to the minimum of $MinSup_i$ for $1 \leq i \leq h_{\max}$. After the frequent sets are generated, each i-item set is compared to $MinSup_i$ to further remove frequent sets that have support level less than $MinSup_i$.

ii) Identifying Closely Related Pages

After identifying the frequent sets, we use **MineAdvanced()** in Figure 3-4 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.

```
MineAdvanced()                          // how advanced rules are mined
{
 for each frequent set S_i {
  for each page P_j {
   if (#ooccurrences 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 ← #ooccurrences 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 3-4: Mining advanced rule.

Chapter 4

BROOM: Applying the Rules to Online Stream

In this chapter, we will have a look at how the extracted knowledge is used by BROOM to do buffer replacement. In the first section, the data structures that are used by BROOM for doing buffer replacement will be discussed. BROOM's online actions can be divided into two separate parts. The first part monitors and triggers the advanced rules while the second part does the actual page replacement. These two parts are discussed in section 2 and section 3 of this chapter respectively.

4.1 Data Structures

BROOM uses three data structures to store the basic rules, the advanced rules, and the predicted arrival of a page. In addition, a system variable *StreamPtr* is used to count the number of arrivals; thus, *StreamPtr* is incremented every time a page reference arrives, and is reset to zero when it overflows. We now examine the three data structures in detail.

i) Structure for Storing Basic Rules

An array $D[1 \ldots n_{\text{dbpage}}]$ will be sufficient for storing these rules. The array element $D[i]$ contains the value of D_{ii} .

ii) Structure for Storing Advanced Rules

Each frequent set S_i has a node with the structure shown in Figure 4-1.

Symbol	Description	Sample value
h	size of frequent set	2
S_i	frequent set with pages sorted in order	$\{P_1, P_7\}$
$F_1, \ldots, F_h$	each F_i keeps track of the number of times a particular page in the frequent set occurs in the window	2,0
active	number of F_i with value greater than 0	1
g	number of pages that are hinted by S_i	3
$H_1, \ldots, H_g$	pages that are hinted by S_i	P_2, P_5, P_8
$D_1, \ldots, D_g$	distance between S_i and hinted pages	12, 30, 2

Figure 4-1: Structure of a node for storing S_i.

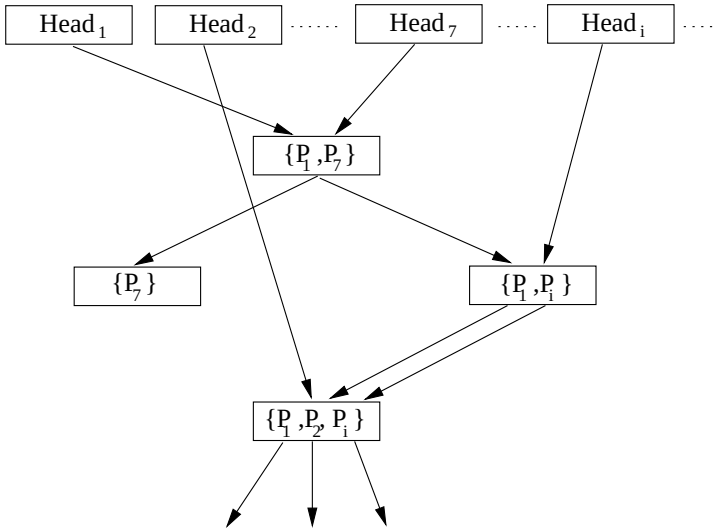

Figure 4-2: BROOM's data structure for storing advanced rules.

For example, if the window contains $\{P_1, P_3, P_{11}, P_9, P_1\}$, then the three advanced rules

$$P_1, P_7 \xrightarrow{12} P_2, \qquad P_1, P_7 \xrightarrow{30} P_5, \qquad P_1, P_7 \xrightarrow{2} P_8$$

are stored in a record shown in Figure 4-1.

These nodes are then linked up to form multiple linked lists that are intertwined with each other as shown in Figure 4-2. A node for a 3-item set will be in 3 linked lists. In the figure, only the frequent set in each node is shown while other details are omitted.

With such a data structure, all nodes with frequent sets containing P_i can be accessed by following $Head_i$ and the corresponding pointers in each node that contains P_i in their frequent set. This greatly reduces the time taken to detect frequent sets that are present in the window.

iii) Structures for Storing Predicted Arrival of Pages

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 arrive. 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, that is, the next arrival is already predicted with another two arrays, $Early2[P_i]$ and $Late2[P_i]$. We will then 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 Monitoring and Activation of Advanced Rules

Figure 4-3 shown BROOM's algorithm for updating the data structures that are described in the earlier section. When a page request for page P_{in} arrived, **Update** is called and P_{in} will be passed to it as a parameter. In **Update**, the window will be shifted which result in a page, P_{out} leaving the window. To update this fact, we first go to $Head_{P_{out}}$ and from there access all nodes that contain P_{out} in their frequent set. For each of these nodes, the corresponding value of F_i (we called it F_{out}) for P_{out} will be decreased and if F_{out} becomes zero, the value of *active* in the node will also be decreased.

After that, the same procedure will be done using P_{in} to access all the nodes that contain P_{in}. However, the corresponding F_i for P_{in} (which we defined as F_{in}) will be increased instead. If F_{in} transit from zero to one, the value of *active* in the node will be increase and compared to the value of h. If *active* is equaled to h, it means that the node's frequent set is found in the window. When this happened, BROOM will proceeds to update the expected arrival time of page H_1 to page H_g in the node.

Consider a page P that is "hinted" by the node's frequent set. Before updating the expected arrival time of P, BROOM will first check for the validity of $Late1[P]$ and $Late2[P]$. There are three possible outcomes:

i) $StreamPtr < Late1[P] < Late2[P]$

ii) $Late1[P] \leq StreamPtr < Late2[P]$

iii) $Late1[P] < Late2[P] \leq StreamPtr$

In the case of (i), both $Late1[P]$ and $Late2[P]$ are valid and no adjustment is needed. For case (ii), it means that Early1[P] and Late1[P] have become invalid. As such, the procedure **PushUp** is called to replace Early1[P] and Late1[P] with Early2[P] and Late2[P] respectively. New estimations for $Early2[P]$ and $Late2[P]$ are then calculated by adding $0.5D_p$ and $1.5D_p$ to the new value of $Late1[P]$. D_p here is the average distance between two consecutive pages P in the basic rules. In case (iii), all the estimated arrival has become invalid. BROOM will then use d, the average distance between the frequent set and P to do an estimation for the next arrival time. The time of second arrival is estimated using basic rules like in case (ii).

Having ensured that the values of the estimated arrivals are valid, the next step of

BROOM is to update the projected arrival time of P. For case (iii), the projected arrival time will have already been updated. For case (i) and (ii), the actions of BROOM differed among the following three cases:

a) $StreamPtr + 1.5d \leq$Late1[P]

b) $Late1[P] < StreamPtr + 1.5d \leq$Late2[P]

c) $Late2[P] < StreamPtr + 1.5d$

For case (a), the activated rule predicts that the next request for page P will arrive sooner than the time predicted by $Early1[P]$ and $Late1[P]$. When such case occurred, $Early1[P]$ and $Late1[P]$ are updated to the values of $StreamPtr$+0.5d and $StreamPtr$+1.5d respectively. In the case of (b), the second next request for page P is assumed to come sooner than the time predicted by $Early2[P]$ and $Late2[P]$. The prediction for the second arrival is drawn nearer by setting $Early2[P]$ and $Late2[P]$ to $StreamPtr$+0.5d and $StreamPtr$+1.5d. For case (c), because the predicted arrival time of the activated rule is later than both $Late1[P]$ and $Late2[P]$, no update will be done.

4.3 Page Replacement

The algorithm for doing page replacement is shown in Figure 4-4. After updating the data structures when P_{in} arrive, P_{in} is then passed to procedure **BROOM** for page replacement.

If P_{in} is not found in the buffer, a page in the buffer must be selected for replacement. Before doing so, BROOM will first check for the validity of the predicted arrival time for all the pages in the buffer. The cases to check for are similar to cases (i), (ii) and (iii) in the previous section. The updates done in each case are also the same except that basic rules

```
Update(P_in )        // updates data structure for rule activation
{                    // and use activated rules to update arrival predictions
 put P_in  at one end of window;
 let P_out  be page at other end (in the past) of window in reference stream;
 for each S --d--> P {
  if (S contains P_in ) update F_in and active for the rule;
  if (S contains P_out ) update F_out and active for the rule;
 }
 for each activated rule S --d--> P {      // rule is activated if S is in window
  if (StreamPtr < Late1[P]) {          // 1st arrival estimate still valid
    SetEarlyLate(StreamPtr ,P,d);            // sharpen the estimate
  } else if (StreamPtr < Late2[P]) {   // 1st arrival estimate obsolete
        PushUp(P);                         // replace by 2nd estimate
        SetEarlyLate(StreamPtr ,P,d);      // sharpen the estimates
  } else {                           // both estimates obsolete
    Early1[P] ← StreamPtr +0.5D_P;// where D_P is the distance in the basic rule P --D_P--> P
    Late1[P] ← StreamPtr +1.5D_P;
    Early2[P] ← Late1[P] +0.5D_P;
    Late2[P] ← Late1[P] +1.5D_P;
  }
 }
}

PushUp(P)
{
 Early1[P] ← Early2[P];              // 1st arrival estimate obsolete
 Late1[P] ← Late2[P];               // replace by 2nd arrival estimate
 Early2[P] ← Late1[P] +0.5D_P;      // calculate new 2nd arrival estimate
 Late2[P] ← Late2[P] +1.5D_P;
}

SetEarlyLate(StreamPtr , P, d)
{
 if (StreamPtr +1.5d < Late1[P]) {            // sharpen estimate for 1st arrival
   Early1[P] ← StreamPtr +0.5d;
   Late1[P] ← StreamPtr +1.5d;
 } else if (StreamPtr +1.5d < Late2[P]) {     // sharpen estimate for 2nd arrival
        Early2[P] ← StreamPtr +0.5d;
        Late2[P] ← StreamPtr +1.5d;
 }
}
```

Figure 4-3: BROOM's algorithms for updating data structures.

are used instead of advanced rules. After the check for validity, the page which is expected to be accessed furthest into the future is replaced.

While the validity of the predicted arrival time for pages in the buffer is checked during a page fault, the validity of the predicted arrival time for P_{in} is checked for both page hit and page miss. The three cases that we check for are as follow:

I) $StreamPtr < Early1[P_{in}]$

II) $Early1[P_{in}] \leq StreamPtr < Early2[P_{in}]$

III) $Early2[P_{in}] \leq$StreamPtr

For case (I), the predicted arrival time of P_{in} is still valid and thus no update is needed. In the case of (II), there are two sub-cases. It could be that the predicted arrival has materialized (that is, this request for P_{in} is the predicted one) or that both $Early1[P_{in}]$ and $Late1[P_{in}]$ have become obsolete. However, the action to be done is the same for both sub-cases; call procedure **Pushup** to update the appropriate values like in case (ii) of previous section. When case (III) occurred, there are again 2 sub-cases. It could be that the predicted arrival has materialized (that is, this request for P_{in} is the one predicted to arrive between $Early2[P_{in}]$ and $Late2[P_{in}]$) or that both $Early2[P_{in}]$ and $Late2[P_{in}]$ have become obsolete. In either sub-cases, all the four values of $Early1[P_{in}]$, $Late1[P_{in}]$, $Early2[P_{in}]$ and $Late2[P_{in}]$ are updated using basic rules.

After page replacement and the update of predicted arrival time are done, BROOM will be ready to handle the next page request.

```
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
    Early1[B_i] ← StreamPtr +0.5D_Bi;      // revise estimates
    Late1[B_i] ← StreamPtr +1.5D_Bi;       // using D_Bi from B_i --D_Bi--> B_i
    Early2[B_i] ← Late1[B_i] +0.5D_Bi;
    Late2[B_i] ← Late1[B_i] +1.5D_Bi;
   } 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];     // imitating optimum offline policy
  replace B_k by P_in ;
 }
 if (StreamPtr > Early2[P_in ]) {        // update estimates for incoming page
  Early1[P_in ] ← StreamPtr +0.5D_Pin ;          // both estimates obsolete
  Late1[P_in ] ← StreamPtr +1.5D_Pin ;
  Early2[P_in ] ← Late1[P_in ] +0.5D_Pin ;
  Late2[P_in ] ← Late1[P_in ] +1.5D_Pin ;
 } else if (StreamPtr > Early1[P_in ])           // 1st arrival estimate obsolete
       PushUp(P_in );                            // replace by 2nd arrival estimate
```

Figure 4-4: BROOM's page replacement algorithm.

Chapter 5

Simulation and Experiments

In this chapter, we will first discuss the simulation that is used to evaluate BROOM's performance. Then we will have look at the results of experiments that are done with the simulator.

5.1 Simulation

As described in Chapter 2, reference patterns can either be sequential or locality patterns. For both types of patterns, 80% of the pages are hot and 20% of the pages are cold. Thus, if page 0 to 19 are hot and 20 to 99 are cold, and the length of the pattern is 10, then an example of a sequential pattern is

14, 12, 18, 28, 11, 17, 16, 15, 90, 13

and an example of a locality pattern for $g = 2$ is

14, 28, 14, 16, 90, 11, 16, 17, 17, 14.

The algorithms for generating patterns are shown in Figure 5-1.

To generate the reference streams (for both training and test streams) recall first that there are two regions in the extended access graph; frequently accessed region and random region. We model the random region by introducing a *random mode* for terminals. When a terminal is in random mode, it sends a random page request. A terminal in random mode has a certain probability of switching into a *pattern mode*. Such a switch represents

```
GeneratePattern()        // generates access patterns of transactions
{
 for (i = 1 to n_pattern){                    // n_pattern=#patterns
  if (i/(n_pattern−i) <n_seq/n_local) GenSequential(i);
  else GenRepeating(i,n_appear);  // n_appear=ave #repetitions per page in pattern
 }
}


GenSequential(i)        // sequential access interspersed with random accesses
{                      // 80% of accesses go to 20% of database pages
 for (j = 1 to ℓ_pattern){            // ℓ_pattern=length of pattern
  if (rand() < 0.2)                   // rand() gives random r, 0 < r < 1
       P_ij ← random cold page;  // P_ij  is j-th page in pattern i
  else  P_ij ← next page in circular list of hot pages;  // sequential scan is over hot pages
 }
}


GenRepeating(i,n_appear)          // generates a pattern with repetitions
{                                // interspersed with random accesses
 H = 0.8ℓ_pattern/n_appear;                  // H = hot pages that are repeated
 Designate H pages as hot pages;
 for (j = 1 to ℓ_pattern){
  if (rand() < 0.2)                  // 80% of accesses go to hot pages
       P_ij ← random cold page;
  else  P_ij ← random page from H hot pages;
 }
}
```

Figure 5-1: Simulator 's algorithm for generating patterns.

the movement from random region into frequently accessed region. A terminal in pattern mode will have an active pattern associated with it. This pattern corresponds to one of the frequently access patterns in the extended access graph. At the end of the active pattern, the terminal will either move into random mode or be associated with another active pattern. The algorithms for generating reference streams are shown in Figure 5-2.

```
GenerateStream()                          // generates a reference stream
{
 for each terminal i                      // initialization
  PickMode(terminal i);
 for (k = 1 to L) {                       // L=length of the reference stream
  terminal ← randomly pick terminal;
  if (terminal in random mode){
   A_k ← randomly pick from n_dbpage pages;     // A_k is k-th access in a reference stream
   PickMode(terminal );
  }
  else { A_k ← next page of pattern in terminal ;
        if (end of pattern) PickMode(terminal );
  }
 }
}


PickMode(terminal )          // sets terminal mode to random or pattern
{
 if (rand() ≥ p) {           // p can be calculated from r_random
  put terminal in pattern mode;
  randomly pick pattern for terminal from n_pattern patterns;
 } else put terminal in random mode;
}
```

Figure 5-2: Simulator's algorithm for generating reference streams.

5.2 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 [26].) Along the way, we will comment on the relative performance of the four policies and draw conclusions on BROOM's behavior.

Figure 5-3 shows the range of values used for the parameters in our experiments. As the number of combinations generated by varying these parameters is rather large, some parameters are fixed at default values during the experiments.

BROOM needs some space in main memory to store its data structures; 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.

5.2.1 Setting the Mining Parameters

As described in Chapter 3.2.1, the mining process is controlled by several parameters, and we now describe how their values are chosen for our experiments.

To show that even the simplest advanced rules mined from the training stream are helpful, we set the value of $h_{\max}$to 2; thus, the only advanced rules used by BROOM are of the form $\{P_i\} \longrightarrow P_k$ and $\{P_i, P_j\} \longrightarrow P_k$. It follows (from Section 3.3.1) that the window size is W has a value of $2n_{\text{terminal}}$.

An important parameter that determines the quality of the rules mined by BROOM is ℓ_{train}, the length of the training stream. Since we use the normal approximation in the

parameters for the database

n_{dbpage}	number of physical pages in the database, fixed at 3000
n_{buff}	number of pages in BROOM's buffer, varies (60–900)
n_{hot}	number of hot pages, fixed at 600 (so there are 2400 cold pages)
$PageSize$	page size, fixed as 8Kbytes

parameters for reference patterns

ℓ_{pattern}	length of a pattern, uniformly distributed (180–300)
n_{pattern}	number of patterns, varies (100–400), default=300
n_{seq}	number of sequential patterns, varies (0–300), default=150
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 (100000–400000), default=300000
ℓ_{test}	length of testing stream, fixed at 400000
n_{terminal}	number of terminals, varies (2–15), default=5
r_{random}	ratio of number of random pages to length of reference stream, varies (0.1–0.8), default=0.5

Figure 5-3: Values of parameters in the experiments.

statistical test in **MineAdvanced()**, we should have $\ell_{\text{train}}>30n_{\text{dbpage}}$ to ensure that frequent sets have sample sizes large enough for the approximation to apply. We experimented with ℓ_{train} from 100000 to 400000 (for n_{dbpage} of value 3000) and find a gradual increase in BROOM's hit rate from the improvement in the quality of the rules; this increase is marginal for values of ℓ_{train} that are greater than 300000, which we therefore adopt 300000 as the default value ℓ_{train}.

Once ℓ_{train} is set, the support levels $MinSup_1$, $MinSup_2$ and $MinSupZ$ are measured from the random stream. Finally, the significance level α for testing advanced rules against basic rules in **MineAdvanced()** is fixed at 0.99.

5.2.2 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 no need to do so. 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.

5.2.3 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.

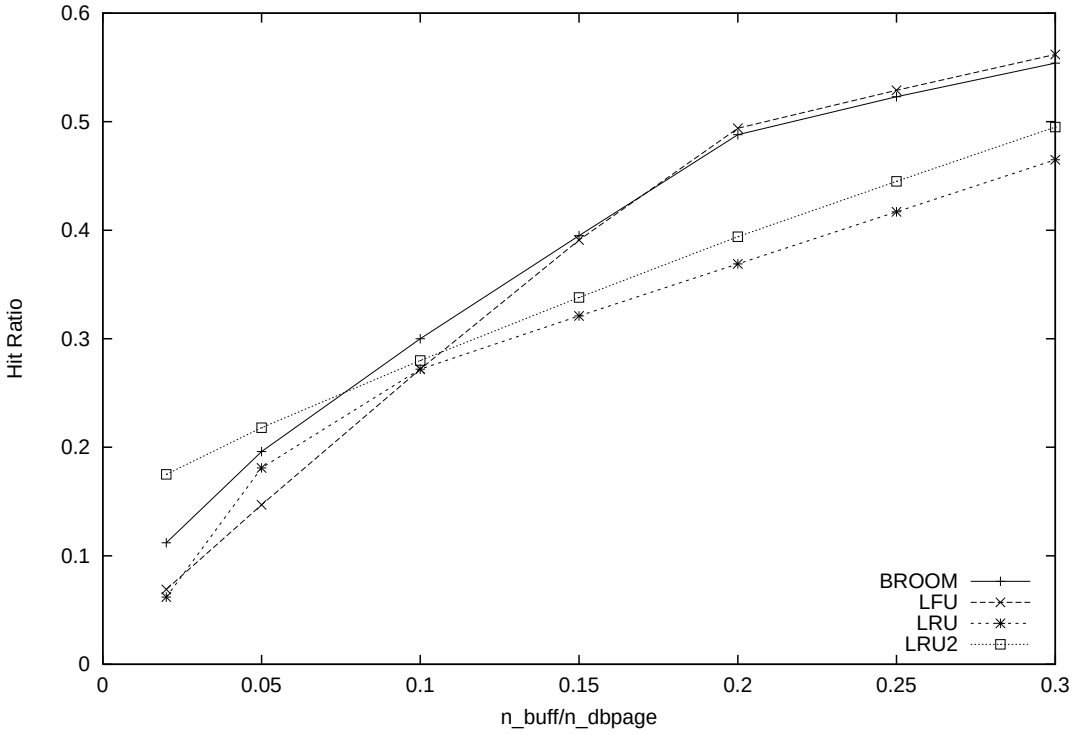

Figure 5-4: Hit rates for varying buffer sizes

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

Figure 5-4 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 "posi-

tion" 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. In such a case, LFU will treats P_i and P_j equally, unliked LRU2 which can see that it is more desirable to keep P_j than to keep P_i in the buffer.

In our simulation, the degree of hotness of the 600 hot pages varies over time. Pages which are present in a active locality pattern will be repetitively accessed several times making them much hotter than those which are not present in one. Thus at any one time, there is a set of hot pages which are the "hottest among the hot" and the composition of such a set varies as locality patterns become active or inactive. Given a small buffer, LRU2 is able to identify this set of pages and makes full uses of the buffer. LFU on the other hand is unable to identify this dynamically changing set of hottest pages (this is mentioned in [26]). As such, LRU2's hit rate is better than LFU's hit rate when the buffer is small. When the buffer is large enough to keep all hot pages, the need to identify the set of hottest pages is removed. As such, LFU is able to outperform LRU2, because it avoids replacing hot pages.

As for BROOM, when the buffer is small, BROOM uses the distances in its rules to keep track of position in the online stream, and thus (like LRU2) outperforms LFU. However, BROOM is unable to identify the set of hottest pages as fast as LRU2 resulting in a lower hit rate than LRU2. For large buffers, BROOM's basic rules contain similar knowledge as frequencies and thus BROOM performs close to LFU. Both BROOM and LFU are able to outperform LRU2 because LRU2 is confused by the random page requests send out by the terminals. The 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. Thus BROOM 's hit rate is the highest for medium buffer.

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

ii) The effect of locality: Changing $n_{local}/n_{pattern}$

The mixture of sequential and locality patterns plays an important part in determining the hit rate of a policy, and this is illustrated in Figures 5-5 to Figures 5-7.

For a small buffer (Figure 5-5), if most of the patterns are sequential patterns($n_{\text{local}}/n_{\text{pattern}}$is small), then LRU and LRU2 have little advantage over BROOM and LFU. As mention earlier, LRU2 outperformed both BROOM and LFU for small buffer because it could identify the set of hottest pages. When there is no locality pattern, this set of hottest pages does not exist and LRU2 loses its advantage over BROOM and LFU. By looking at Figure 5-5, we can see that this advantage returns as the number of locality patterns increases and LRU2 eventually becomes the best policy. BROOM, which could only detect some of the hottest pages, is the second best policy as the number of locality patterns increases. The performance of LRU remains fairly constant although locality increases, this is due to the fact that the average length of the loops in the locality patterns are larger than the size of the buffer.

For a medium buffer (Figure 5-6), the same trend occurs. BROOM performs better than LFU but loses out to both LRU and LRU2 as the number of locality patterns increases. However, there is a difference in that LRU eventually becomes the best policy instead of LRU2. This is caused by two factors. Firstly, the size of the buffer is now larger than the size of the loops in the locality patterns. Secondly, with a high number of locality patterns, LRU's replacement decisions tend to be more accurate because once a page is accessed, it

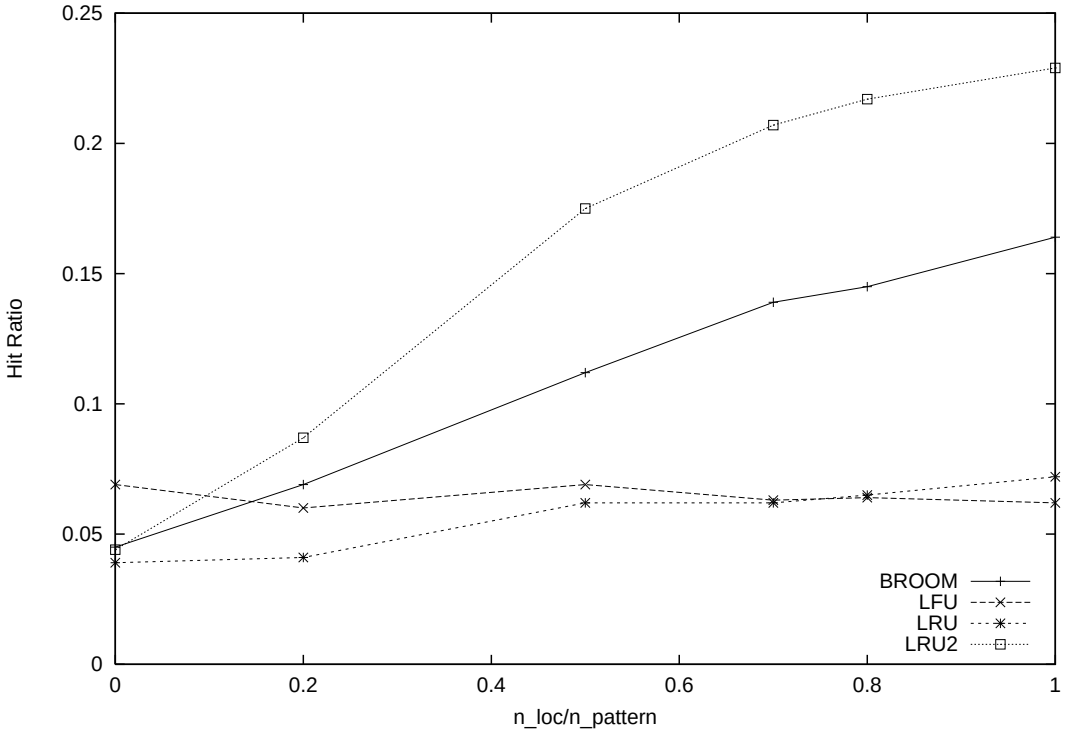

Figure 5-5: Varying n_{local} with n_{buff}/n_{dbpage} =0.02

is quite confirmed that it will be accessed in the near future.

In the case of a large buffer, we find that observation (B3) still holds. However, the performances of the four policies converge as the number of locality patterns increases.

iii) 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 (Figure 5-8), LRU2 begins to break away from the other policies as n_{appear}exceeds 4. As we have mentioned earlier, LRU2 is able to identified the set of hottest page. When n_{appear}increases, this set of pages becomes hotter and thus the hit rate of LRU2 increases. With the knowledge mined, BROOM is able to stay close to LRU2 as n_{appear}increases, unlike LRU and LFU. We can also see that LRU 's hit rate increase with n_{appear}. This is due to the fact that the average length of the loops in the locality

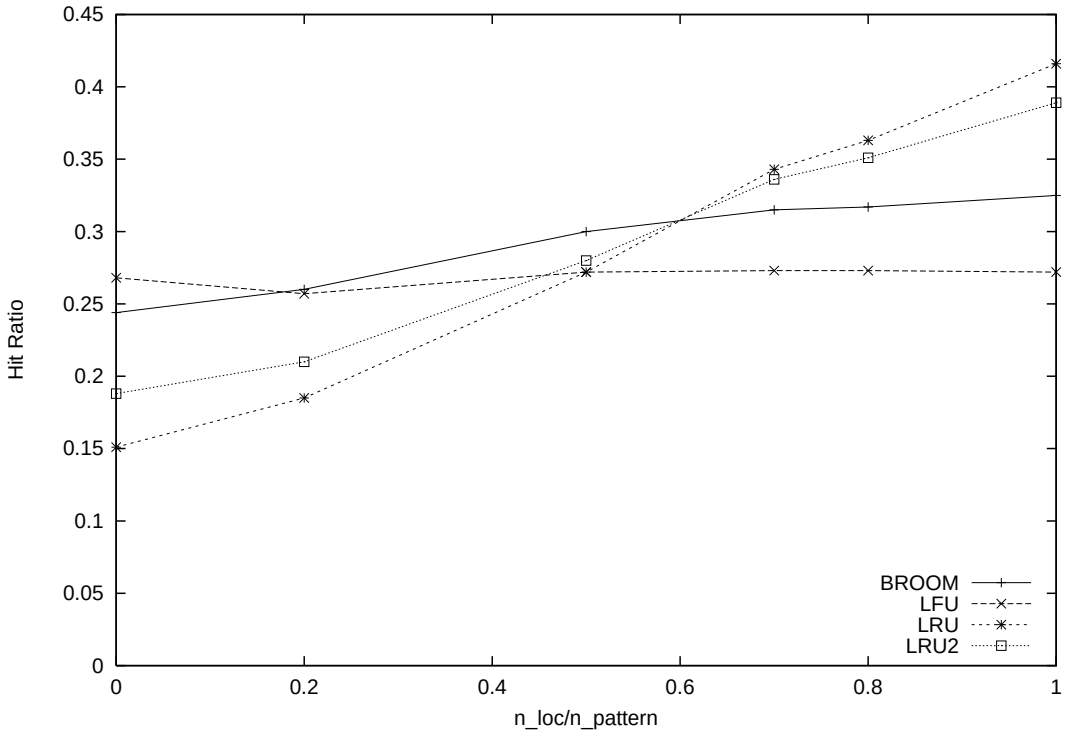

Figure 5-6: Varying n_{local} with n_{buff}/n_{dbpage}=0.100

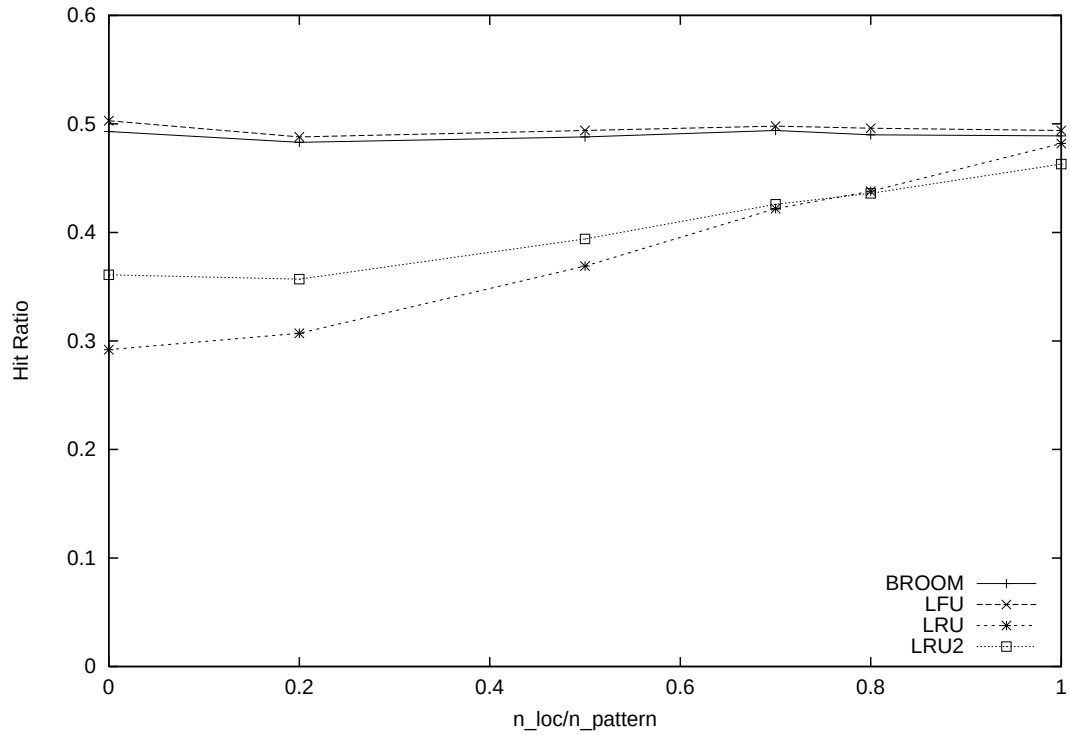

Figure 5-7: Varying n_{local} with n_{buff}/n_{dbpage}=0.200

patterns decreases as n_{appear}increases.

For medium buffers (Figure 5-9), the buffer size is large enough to hold pages from both active sequential and active locality patterns. BROOM is able to distinguish between these two types of pages and will replace a page from an active sequential pattern during a page miss. The increase in n_{appear}in this case will provide the same increase in hit rate for both BROOM and LRU2 because both BROOM and LRU2 will keep the set of hottest pages from active locality pattern in the buffer. BROOM however has another added advantage: it is able to select the correct page to be replaced from among those pages which are present in active sequential patterns. As such its performance is better than the rest of the policies with a medium buffer.

For large buffers (Figure 5-10), n_{appear}has negligible effect as almost all the hot pages are kept in the buffer.

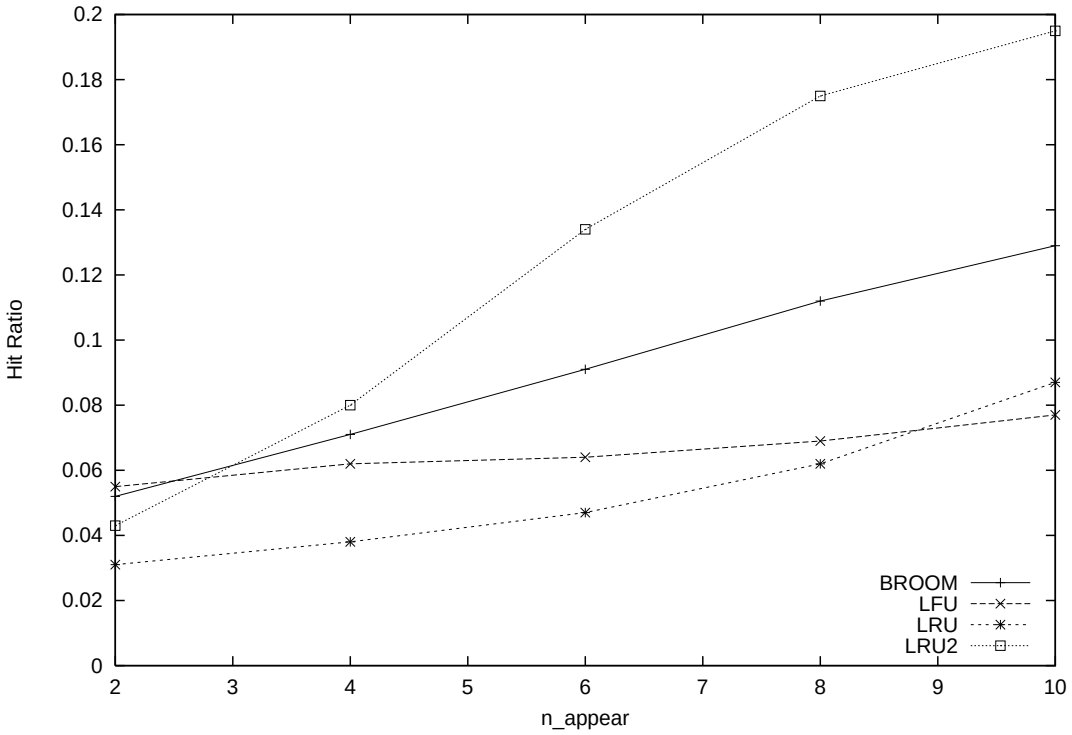

Figure 5-8: Varying n_{appear} with n_{buff}/n_{dbpage} =0.02

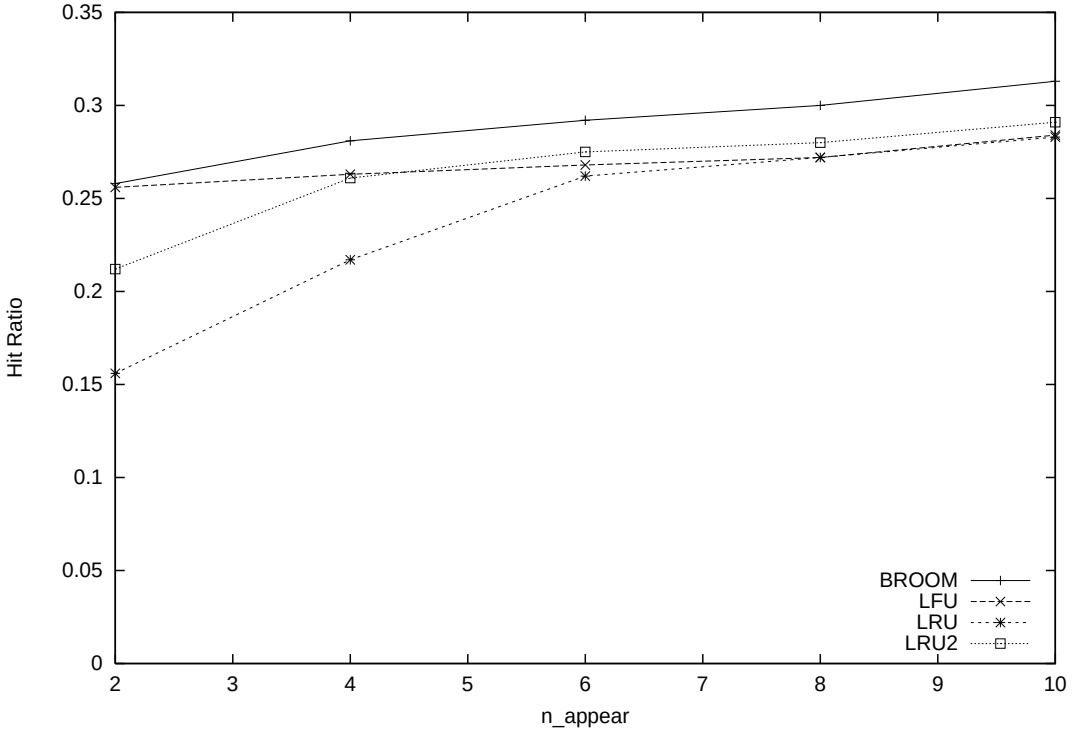

Figure 5-9: Varying n_{appear} with n_{buff}/n_{dbpage}=0.100

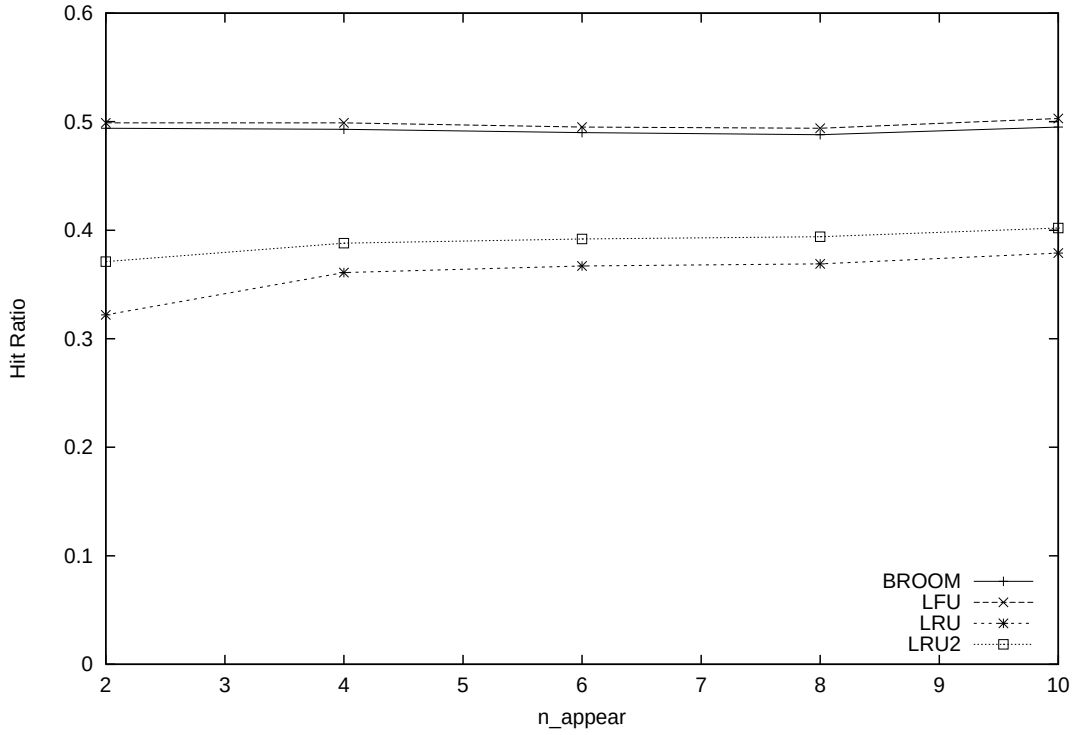

Figure 5-10: Varying n_{appear} with n_{buff}/n_{dbpage}=0.200

iv) The effect of random references: Changing r_{random}

As mentioned in Chapter 5.1, the reference stream is interspersed with random references, and we model this by a probability r_{random}, which specifies the fraction of references that are random By observing Figures 5-11 to 5-13, we can see that 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.

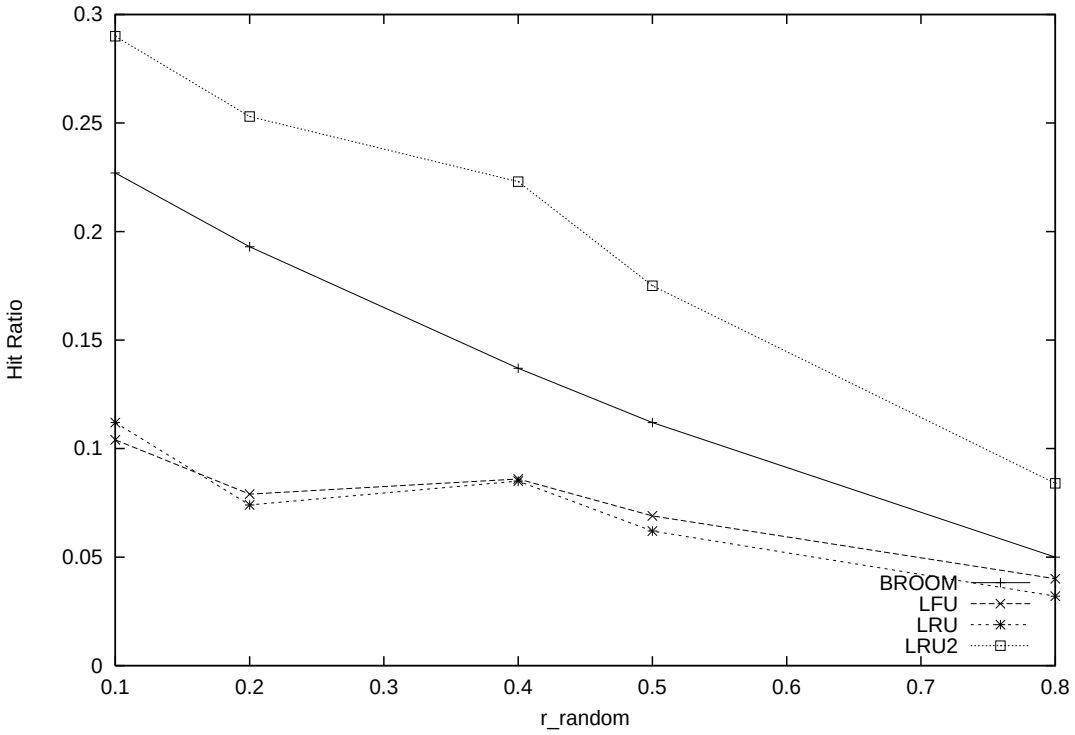

Figure 5-11: Varying r_{random} with n_{buff}/n_{dbpage} =0.02

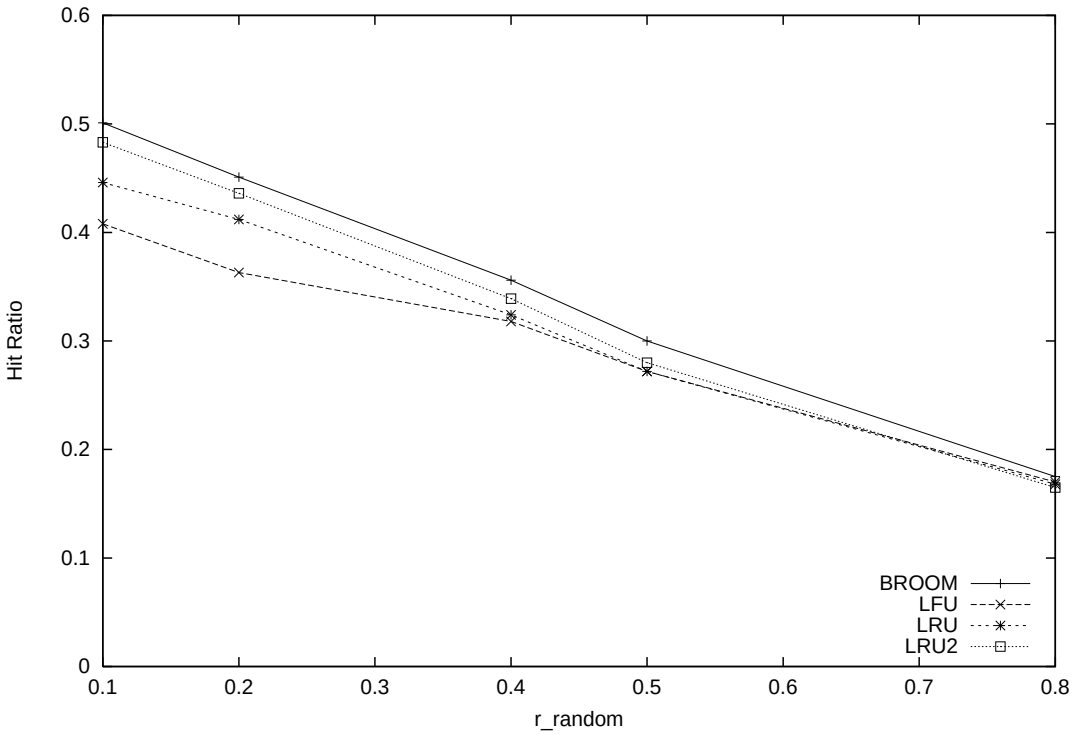

Figure 5-12: Varying r_{random} with n_{buff}/n_{dbpage}=0.100

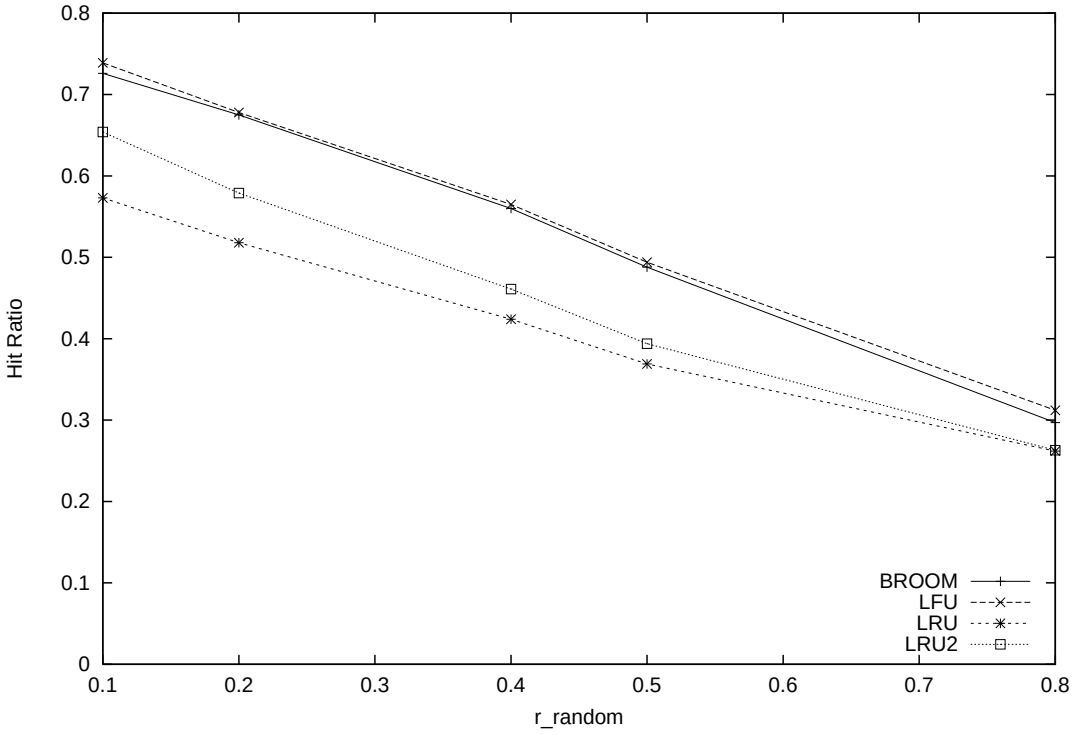

Figure 5-13: Varying r_{random} with n_{buff}/n_{dbpage}=0.200

v) The effect of number of patterns: Changing n_{pattern}

When the number of patterns (n_{pattern}) increases, it will be natural that 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.

Using our default values, it can be calculated that each pattern is activated least than three times in a reference stream with 300000 page requests. This is hardly enough for pattern discovery, and we can see the deterioration of BROOM's hit rate for small and medium buffers in Figures 5-14 and 5-15.

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

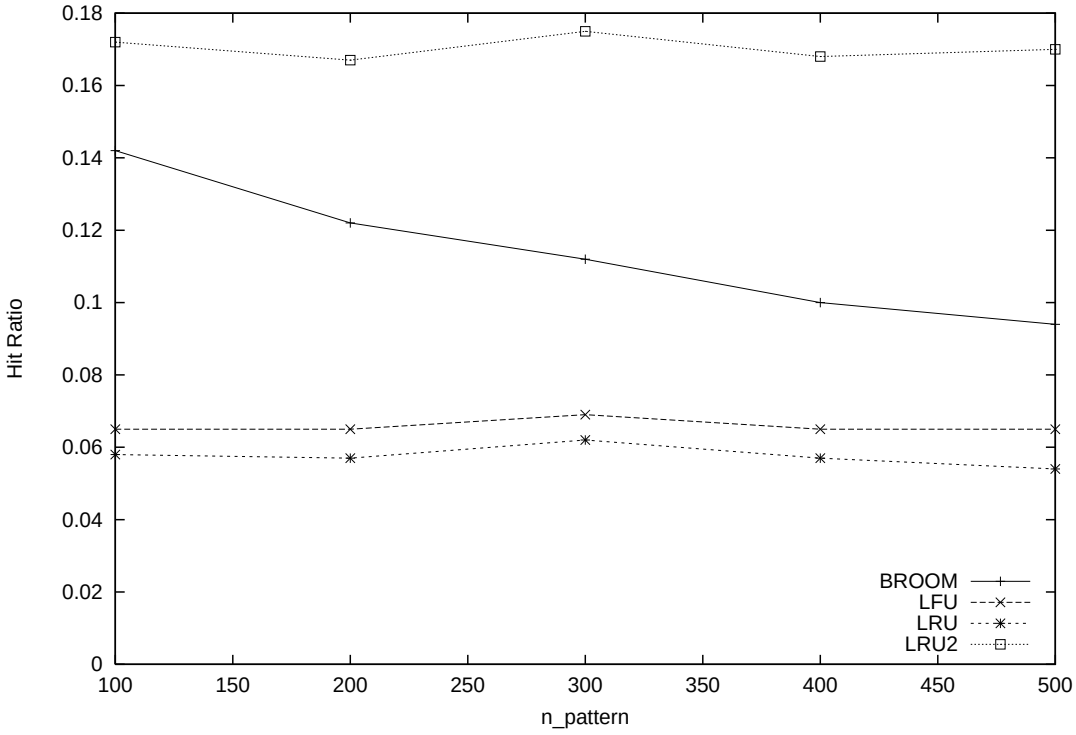

Figure 5-14: Varying $n_{pattern}$ with n_{buff}/n_{dbpage} =0.02

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 5-16), since it keeps all the hot pages in the buffer.

Due to the fact that 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.

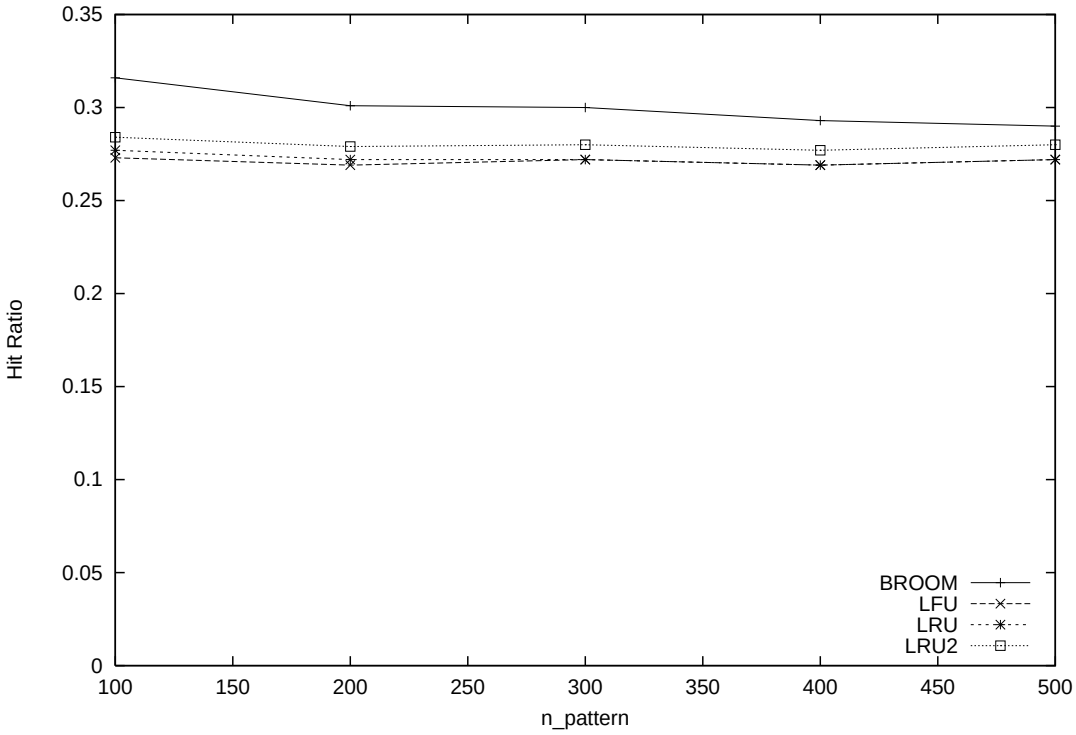

Figure 5-15: Varying $n_{pattern}$ with n_{buff}/n_{dbpage}=0.100

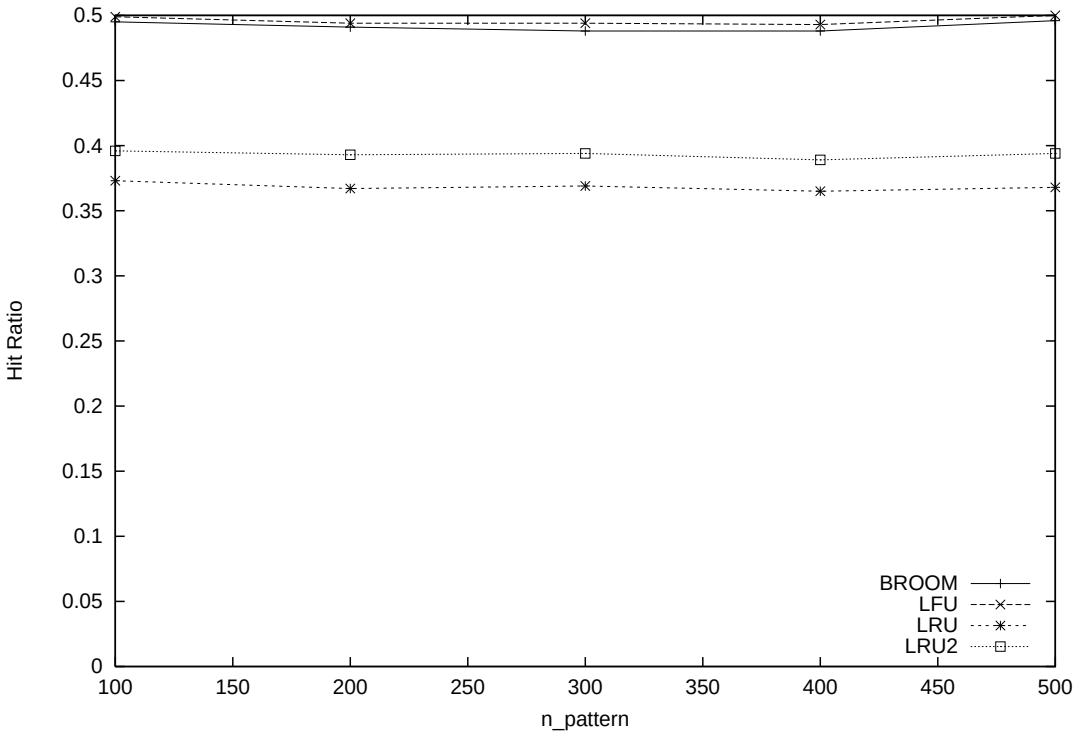

Figure 5-16: Varying $n_{pattern}$ with n_{buff}/n_{dbpage}=0.200

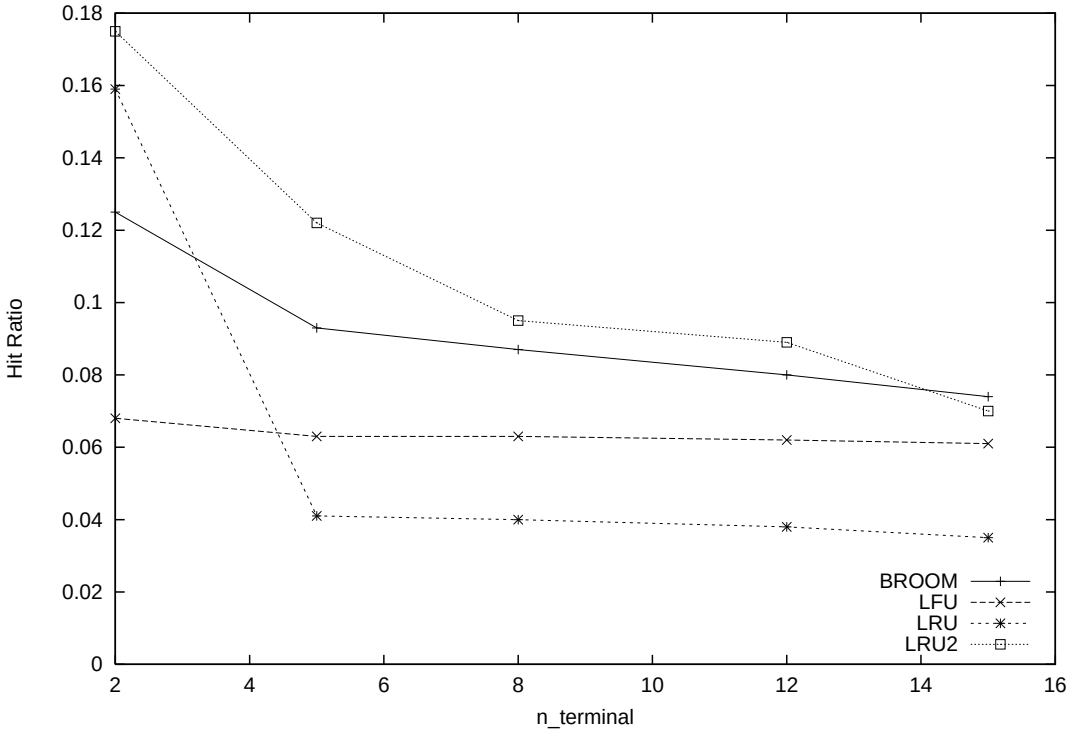

Figure 5-17: Varying $n_{terminals}$ with n_{buff}/n_{dbpage} =0.02

vi) The effect of number of terminals: Changing n_{terminal}

As each terminal in our system model is running one transaction, more patterns will be simultaneously active, when n_{terminal}increases. This may confuse BROOM's online decisions. We verify this in Figures 5-17 to 5-20, where we experimented with $n_{\text{terminal}}$from 2 to 15. The experiments confirm our intuition.

For small buffers, BROOM's hit rate decreases when there are more terminals. However, for medium and large buffers, BROOM's rules can keep in memory all the pages hinted by all the patterns that are possibly active (judging from the current references), so BROOM's hit rate becomes insensitive to n_{terminal}.

Even with a small number of patterns, the knowledge that BROOM (or any other policy) discovers would be useless if $n_{\text{terminal}}$is close to n_{buff}. This is there is hardly enough buffer frames for each active transaction.

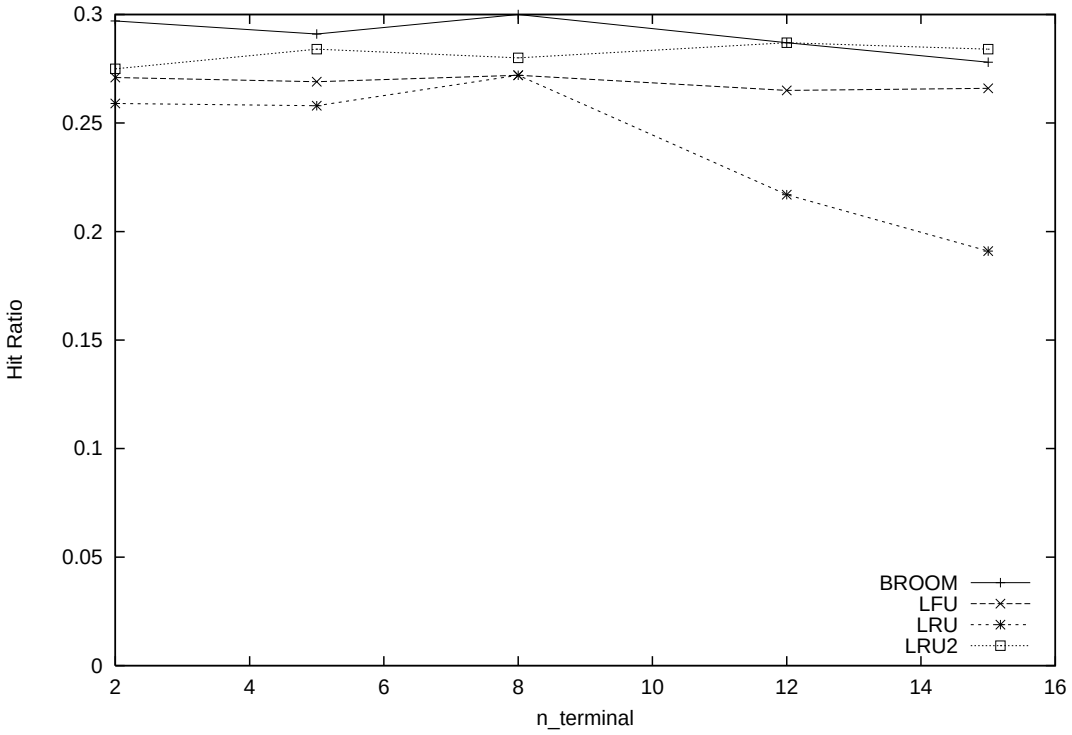

Figure 5-18: Varying $n_{terminals}$ with n_{buff}/n_{dbpage}=0.100

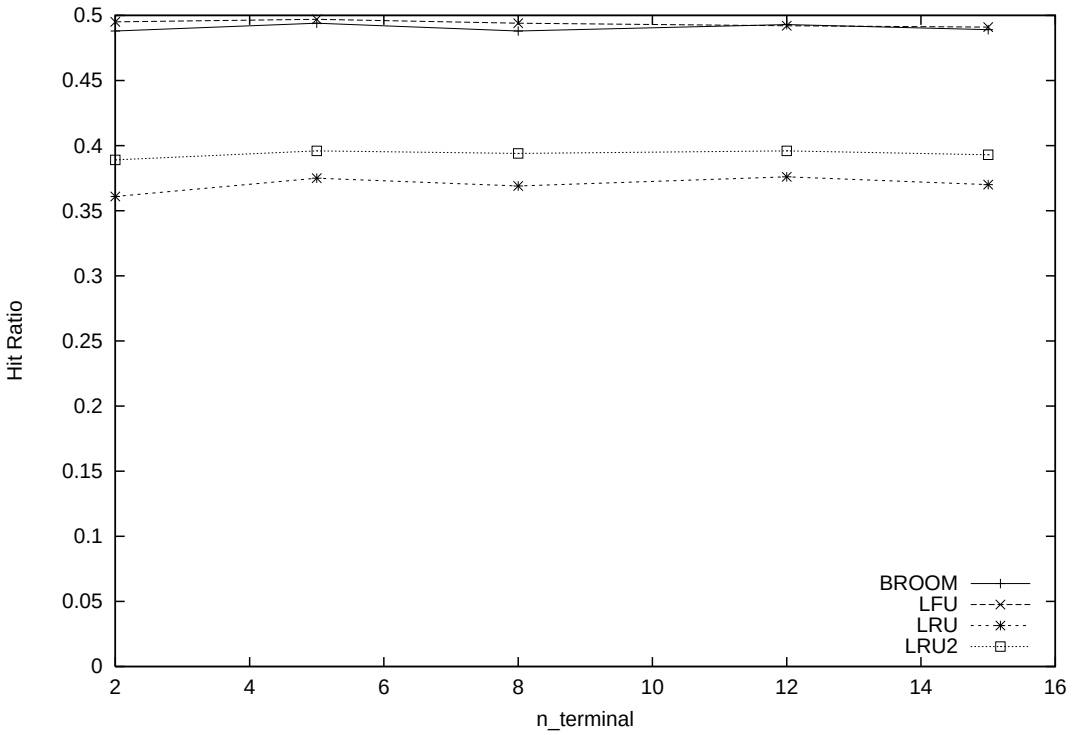

Figure 5-19: Varying $n_{terminals}$ with n_{buff}/n_{dbpage}=0.200

Chapter 6

Conclusion

In this chapter, we will first discussed some issues of concern in the use of BROOM. We will then describe the conditions suitable for using BROOM and conclude the thesis by suggesting some directions for future work.

6.1 Issues of concern

6.1.1 BROOM's Overheads in Space and Time

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 Chapter 5, mining the training stream took less than 5 minutes on a SunSparc Ultra-1 processor. To avoid repeatedly scanning through the training stream, we can make use of the data structure that is used for storing advanced rules (shown in Figure 4-2) during mining. After using the Apriori algorithm to extract the frequent sets, the data structure is first built for these sets. A pass is then made through the training stream while monitoring what item sets are present in the window. Having detected an item set in the window, the distance of all high frequency pages from the item set is then updated. As frequent sets contain mostly high frequency pages, the time complexity for filling in this data structure is $O(n_{\text{freq}} \times \ell_{\text{train}})$, 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 Chapter 5) 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. We would however like to point that we are storing patterns only for pages which will help to increase the page hit. These pages can be identified using the "Five Minutes Rule" which state that only pages which will be accessed again in 5 minutes are worth keeping in the buffer [16]. Furthermore, 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 [17], 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 [17], our calculations show that this increase in processing time is still worth the increase in hit rate, especially where observation (B2) is concerned.

6.1.2 Frequency of Mining

Another pertinent issue is how often mining should be done. Naturally, this depends on the validity period for the extracted rules.

A DBMS administrator can use her knowledge of the applications to decide when the rules may be outdated. For applications that change slowly (like data warehousing), the administrator may decide to perform mining every week. On the other hand, if updates are frequent, mining may be required every few hours which is still feasible, since BROOM may take only a few minutes to mine the rules (see above).

When deciding on the validity period of extracted rule, special care must however be taken to ensure that *local changes* are not confused with *global changes*. To illustrate, consider a user of a decision support system who issues fifty different queries within two hours and repeat them after the two hour period. If a person observes him for one hour and compare the observation to the next hour, he will see local changes in the queries and will conclude that the period of validity is less than one hour. On the other hand, if the observation period is extended to four hours or more, he will see that there are in fact no global changes.

Beside relying on the knowledge of DBMS administrator, one could also simply monitor the hit rate, and trigger a new round of mining when the hit rate drop below a certain threshold. Here, extra care must be taken in selecting a good threshold value. A threshold value that is too low will result in unnecessary mining while too high a threshold value will result in changes being undetected.

6.2 When to use BROOM?

The following conditions are suitable for using BROOM:

Existence of a small set of stable reference patterns. This is the most important condition since BROOM is based on this assumption. Clearly, for a pattern to be considered stable, it must remain valid throughout the training stream and for any online stream that is a target for applying the mined rules.

The number of such stable patterns is also important. BROOM works best if a small number of patterns are active most of the time. This will mean that the overheads for using BROOM will be low and the effectiveness of the rules extracted will be high. We believe such cases exist in real applications. For example, Gray *et al* observed instances where 80% of disk accesses go to 3% of the physical pages [16]. This 3% strongly indicates the existence of patterns and locality and the 80% indicates that these patterns are active most of the time.

Medium buffer size. As indicated by observation (B2), BROOM is most useful for medium-sized buffers. Changes in the number of hot pages, budget constraint, etc. may mean that the buffer is not large enough to contain all the hot pages. On the other hand, it may not make sense to operate a system with a buffer that is smaller than 10% of the hot pages. The buffer size between these two extremes is just the range suitable for BROOM.

Large page size. As mentioned above, a large page size will allow more rules to be stored in one page. This thus reduces the number of buffer frames that need to be reserved for BROOM's data structures.

6.3 Future Work

This thesis presents a new replacement policy BROOM, and demonstrates that it can perform better than other replacement policies under certain conditions. We have some suggestions which might improve BROOM further and which we think are worth looking into.

- Firstly, as the overhead in time use for detecting activated rules and updating of predicted arrival time may affect the response time to a page request, it will make sense to run procedures **Update** and **BROOM** as two concurrent processes which share a common memory. **BROOM** will be given a higher priority and **Update** will only run when there are no page requests coming in or when the disk is busy fetching pages into the buffer. In such a model, activated rules will not affect predicted arrival time immediately. The delay in update may not have too much effect on the performance because pages "hinted" by an activated rule may not be in the buffer. Furthermore, we are only interested in the furthest page in the buffer. As long as the delay does not cause the predicted furthest page to change, the performance will be the same.

- Secondly, to reduce the amount of memory required to store advanced rules, the distance between the antecedent and consequent can be represented with less accuracy. For example, if the distance between the antecedent and consequent has a range between 1 to 1024, we can cluster this range into 32 values, with each value representing 32 page accesses. As such, a distance of 250 will then be represented by value of $\lceil 250/32 \rceil$ or 8. Although some accuracy is lost when we convert the value back to actual distance (we will get 256 for the example), this error could be offset because we

are using a range of values ($Early1$ and $Late1$, $Ealy2$ and $Late2$) for the predicted arrival.

- Thirdly, to factor out locality, an additional FIFO queue can be used to store pages that are previously not in the buffer. Pages in the queue will not be eligible for replacement. When a page fault occurs, a page in the actual buffer will be evicted using the BROOM replacement policy and the page at the head of the queue will be moved into the actual buffer. The requested page will then be put at the end of the queue. In the case of a page hit, the requested page will be moved to the end of the queue. If the requested page is in the actual buffer, the page at the beginning of the queue will then be moved into the actual buffer. We believe that factoring out locality will bring the performance of BROOM closer to LRU2 for small buffer because LRU2 mainly relies on locality patterns to achieve its high hit rate.

While BROOM 's performance is not always the best among the replacement polices, it always avoids the worst. This is due to the mined knowledge that is used by BROOM when performing buffer replacement. This "adaptive" behavior demonstrates how data mining technology can be used to build an adaptive DBMS. We hope this thesis will encourage research in this direction.

The development of BROOM is actually part of our efforts to apply data mining techniques to general online optimization algorithms. Based on our experience with BROOM, we propose a general framework for applying data mining to online optimization, as shown in Figure 6-1.

This framework is an integration of two research areas: data mining and traditional algorithm design. We hope that more research will be done to apply this framework to

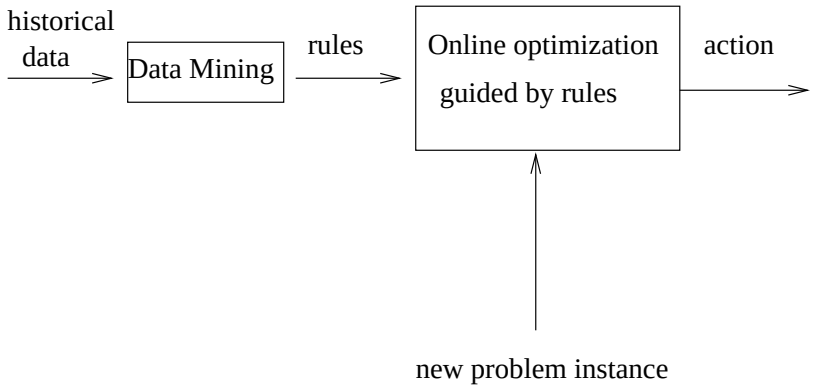

Figure 6-1: General Framework of BROOM

other online optimization problems.

Bibliography

[1] R. Agrawal and Ramakrishnan Srikant. *Fast algorithms for mining association rules.* in Proc. of the 20th VLDB Conference, Santiago, Chile, 1994.

[2] R. Agrawal and Ramakrishnan Srikant.*Fast algorithms for mining association rules in large databases.* Research Report RJ 9839, IBM Almaden Research Centre, San Jose, Calfornia, Jun. 1994.

[3] R. Agrawal, T. Imielinski and A. Swami. *Mining association rules between sets of items in large databases.* In Proc. of the ACM SIGMOD Conference on Management of Data, Washington D.C. May 1993.

[4] Ö. Babaoğlu and D. Ferrai. *Two-level replacement decisions in paging stores.* IEEE Transactions on Computers, 32(12):1151-1159, Dec. 1993.

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

[6] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. *Competitive paging with locality of reference.* In Proc. 23rd Annual ACM Symposium on Theory of Computing, pages 249-259, 1991.

[7] M.J Carey, R.Jauhari, and M. Livny. *Priority in DBMS resourse scheduling.* In Proc. of the 15th Conference on Very Large Data Bases, pages 397-410, Amsterdam, Aug. 1989.

[8] C.Y. Chan, B.C. Ooi and H. Lu. *On coupling multiple systems with a global buffer.* IEEE Transactions on Knowledge and Data Engineering, 8(2):339-344, Apr. 1996.

[9] A. Dan and D. Towsley. *An approximate analysis of the LRU and FIFO buffer replacement scheme.* In Proc. of the 1990 ACM SIGMOD International Conference on Management of Data pages 143-152, Atlantic City, NJ, May 1990.

[10] W. Effelsberg and T. Haerder. *Principle of database buffer management.* ACM Transaction on Database Systems, 9(4):560-595, Dec. 1994.

[11] U. Fayyad, G. Piatetsky-Shaipro, P. Smyth and R. Uthurusamy, Eds. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, Menio Park, CA, 1996.

[12] 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, pages 246-260, Valencia, Spain, March 1998.

[13] A. Fiat and A. R. Karlin. *Randomized and multipointer paging with locality of reference.* In Proc. of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing, pages 626-634, Las Vegas, Nevada, 29 May-1 June 1995.

[14] A. Fiat and M. Mendel. *Truly online paging with locality of reference (extended abstract).* In 38th Annual Symposium on Foundations of Computer Science, pages 326-335, Miami Beach, Florida, 20-22 October 1997. IEEE.

[15] P. A. Franaszek and T. J. Wagner. *Some distribution-free aspects of paging performance.* Journal of the ACM, 21:31-49, 1974.

[16] 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.* Proceedings of SIGMOD 87, June 1987, pp. 395-398.

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

[18] S. Irani, A. R. Karlin and S. Phillips. Strongly competitive algorithms for paging with locality of reference. In Proc. of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, pages 228-236, 1992.

[19] R. Jauhari, M.J. Carey, and M. Livny. *Priority-hints: an algorithm for priority-based buffer management.* In Proc. of the 16th Conference on Very Large Data Bases, pages 708-721, Brisbane, Australia, Aug. 1994.

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

[21] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. *Competitive snoopy caching.*. Algorithmica, 3(1):70-119, 1988.

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

[23] D.C. Montgometry, G.C. Runger. *Applied Statistics and Probability for Engineers* John Wiley & Sons Inc.

[24] V.F. Nicola, A. Dan and D.M. Dias. *Analysis of the generalised clock buffer replacement scheme for database transaction processing.* ACM SIGMETRICS and PERFORMANCE 20(1): 35-46, June 1992.

[25] E.J. O'Neil, P.E. O'Neil and G. Weikum. *An optimaliy proof of the lru-k page replacement algorithm.* Technical Report, UMass/Boston and University of Saarland, 1996.

[26] 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 International Conference on Management of Data, pages 297-306, Washington D.C. Usa, Aug. 1993.

[27] J.T. Robinson and M.V. Devarakonda. *Data cache management using frequency-based replacement.* In Proc. of the 1990 ACM SIGMOD International Conference on management of data, pages 134-142, Brisbane, August, Aug. 1990.

[28] D. D. Sleator and R. E. Tarjan. *Amortized efficiency of list update and paging rules.* Communications of the ACM, 28: 202-208, Feb. 1985.

[29] J. R. Spirn. *Program Behavior: Models and Measurements.* Elsevier Computer Science Library. Elsevier, Amsterdam, 1977.

[30] G. Weikum, C. Hasse, A. Mönkeberg, and P. Zabback. *The comfort automatic tuning project.* Information Systems, 19(5):381-432, May 1994.