ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2008/02(02)-0139-27
DOI:10.3778/j.issn.1673-9418.2008.02.004

E-mail:fcst@public2.bta.net.cn
http://www.ceaj.org
Tel:+86-10-51616056

A Comprehensive Performance Evaluation of Buffer Replacement Strategies in RDBMS*

LUAN Hua[1,2+], DU Xiaoyong[1,2], FENG Yu[3], WANG Shan[1,2]

1. Key Laboratory of Data Engineering and Knowledge Engineering, Ministry of Education, Renmin University of China, Beijing 100872, China
2. College of Information, Renmin University of China, Beijing 100872, China
3. Industrial Research Center of Database and Business Intelligence, Ministry of Education, Beijing 100085, China

+ Corresponding author: E-mail: luanhua@ruc.edu.cn

关系数据库缓冲区置换策略的性能评测

栾　华[1,2+],杜小勇[1,2],冯　玉[3],王　珊[1,2]

1. 中国人民大学 教育部数据工程与知识工程重点实验室,北京 100872
2. 中国人民大学 信息学院,北京 100872
3. 数据库与商务智能教育部工程研究中心,北京 100085

摘　要:缓冲区是计算机领域一个非常重要的研究主题。现在存在很多缓冲区置换策略,广泛地用于操作系统、文件系统、数据库以及存储系统中。提出了一种性能评测方法来评测关系数据库的缓冲区置换策略。这种方法采用直接测试方式,使用五个性能指标,包括新提出的指标以及由于模拟方法的局限性在以前的研究中很少使用的性能指标,同时利用显著性测试并提出一种联合使用多个显著性测试的方法进行置换策略之间的比较,设计和使用多种工作负载在单用户和多用户情况下有效地评测各种策略。使用这种评测方法,在一个真实的关系数据库环境中,对十三种典型的缓冲区置换策略的性能进行了详细、全面的实验研究,从中可以获得一些有用的结论。

关键词:缓冲区置换策略;性能评测方法;显著性测试;访问模式;TPC-H 和 TPC-C 基准

文献标识码:A　　**中图分类号:**TP302

* the National Natural Science Foundation of China under Grant No.60496325,60573092 (国家自然科学基金); the Project 985 of Ministry of Education of China (985 工程项目).

Received 2007-12, Accepted 2008-03.

LUAN Hua, DU Xiaoyong, FENG Yu, et al. A comprehensive performance evaluation of buffer replacement strategies in RDBMS. Journal of Frontiers of Computer Science and Technology, 2008,2(2): 139-165.

Abstract: Caching is one of the most fundamental topics in modern computing. There exist many buffer replacement strategies, which are widely used in operating systems, file systems, databases, storage systems, etc. A performance evaluation methodology to evaluate buffer replacement strategies for RDBMS is proposed. This methodology adopts direct measurement way and employs five performance metrics including the ones newly proposed by this paper and those seldom considered in other publications due to their simulation method. Significance tests and a joint way to use these tests for cross-strategy comparison are presented and various workloads are designed and used in order to effectively evaluate buffer replacement strategies in both single-user and multiuser environments. Using this methodology, it performs a detailed and comprehensive experimental study about the performance of thirteen representative buffer replacement strategies in a real RDBMS environment. From the experimental results, some useful conclusions are made.

Key words: buffer replacement strategy; performance evaluation methodology; significance test; access pattern; TPC-H and TPC-C benchmark

1 Introduction

In order to increase system performance, cache is widely used in storage systems, databases, operating systems, file systems, etc. One of the most important design decisions is the replacement strategy of pages in the buffer cache. Many buffer replacement strategies have been proposed, such as LRU[1], CLOCK[2], GCLOCK[3], FBR[4], LRU-K[5], 2Q-Simple[6], 2Q-Full[6], LRFU[7], ARC[8], LIRS[9], CAR[10], CART[10] and CLOCK-Pro[11]. However, how to choose or design an appropriate buffer replacement strategy for a data intensive system is an important and difficult problem. When researchers proposed their buffer replacement strategies, they did experimental studies, but the evaluation method they used has some limitations and only a partial comparison of the strategies mentioned above has been made. So it is meaningful to use a better evaluation methodology to perform a thorough experimental study on the existing buffer replacement strategies, which will provide valuable insight into the effects of various strategies on the overall system performance. To the best of our knowledge, there has been no research to put these strategies together and conduct a comprehensive and detailed performance evaluation.

Simulation evaluation method was used in all the previous work[4-11], except that the authors of [11] implemented CLOCK-Pro in Linux 2.4.21, and utilized hit ratio as the main performance metric in comparing different algorithms. However, a high hit ratio is not necessarily correlated to high performance in real systems[12]. So besides hit ratio, some other performance metrics, such as throughput and execution time, should be included in an evaluation methodology.

Many researchers usually concluded from their evaluation results that Strategy A “is close to” or “performs better than” B. But it is not clear what has been claimed precisely. Does it mean statistically significant? A better evaluation methodology should be able to explain the results more exactly.

In this paper, we propose a performance evaluation methodology and conduct a thorough experimental study to evaluate various buffer replacement strategies. [12] provided a simulation evaluation method to compare a few replacement algorithms. Our work is different in many aspects. Firstly, we implement thirteen well-known or newly-published buffer replacement algorithms, which are more recent than those of [12], in a database product-KingbaseES[13], and adopt a direct measurement method, instead of a simulation way. Secondly, besides metrics used in [12], we introduce several new performance metrics. Thirdly, we introduce significance tests in statistics to compare various buffer replacement strategies and propose a joint way to use them. Fourthly, unlike [12], which considers only multiuser tests, we consider the performance of various algorithms in both single-user and multiuser environments by designing and using workloads of sequential references, correlated references, looping references, temporally-clustered references, probabilistic references, TPC-H[14] and TPC-C Benchmark[15].

The rest of this paper is organized as follows. In Section 2 we simply survey the related work. Our performance evaluation methodology is provided in Section 3. In Section 4 we briefly introduce various buffer replacement strategies. In Section 5 we report and analyze the experimental results. Conclusions appear in Section 6.

2 Related Work

Usually, the algorithm designers conduct performance comparisons between their own algorithms and some previous algorithms in their studies[4-11]. For example, the authors of CLOCK-Pro compared the performance difference of CLOCK, CAR, LIRS and CLOCK-Pro strategies. Generally, these experiments were done by using simple simulation technique-algorithm simulators and trace-driven workloads, and most of these studies only focused on the hit ratio metric of buffer replacement strategies in a single-user environment. [12] presented an evaluation method to compare its algorithm with other five ones. The authors employed a hybrid simulation model including a database system simulator, and mainly considered the throughput of algorithms in a multiuser environment.

Although the above literature can provide valuable information in some aspects, there are also some limitations. First, a simulation evaluation method is used in their experiments and performance metrics are incomplete. Second, there are no clear conclusions about cross-strategy comparison because all of the algorithms are not compared in the same testbed. Our work can deal with these problems well.

3 Performance Evaluation Methodology

In this section, we describe our performance evaluation methodology focusing on four aspects: direct measurement way, workload design, five performance metrics and significance tests.

3.1 Direct Measurement Way

As pointed out in [12], there are three ways to evaluate different buffer replacement strategies: direct measurement, analytical modeling, and simulation. Direct measurement is a precise way but it may be expensive in implementation. Analytic modeling, while quite cost-effective, can not model the different algorithms in sufficient detail. Simulation, especially trace-driven simulation, is often used in many publications such as [5,6,8,9]. It does have several advantages including credibility and fine workload characterization. However, there also exist some limitations such as its difficulty to characterize the interference and correlation between concurrent activities in a multiuser environment. Most importantly, this method is usually combined with simple replacement algorithm simulators, so that except hit ratio many metrics can not be acquired exactly especially for multiuser workloads. Thus, hit ratio of algorithms almost becomes the unique consideration and researchers can not know the practical performance advantages of the algorithms. The ultimate goal of a buffer replacement algorithm is to maximize throughput and reduce execution time in a real system. Hit ratio is only one of the factors that affect system performance in practice. For example, an algorithm has a high hit ratio but its lock contention is serious, and much time is wasted in waiting for lock release. Therefore, though its hit ratio is better than those of other algorithms, it does not outperform others in overall system performance. [12] made some process in this aspect by implementing a database system simulator, simulating three hardware components in its model and using throughput as its performance metric. But in essence, it still adopted simulation measurement method which could not fully and exactly reflect the actual system behavior. For example, the execution time due to concurrency control on buffer pages is not included in the total execution time of [12], so its throughput is not precise.

To study the performance merits of various buffer replacement strategies in a real environment, we implemented thirteen buffer replacement strategies in a database product-KingbaseES[13] and ran the KingbaseES database under various workloads to test each algorithm. The KingbaseES can collect the statistical data about physical reads, logical reads and so on. These data are used to calculate our performance metrics.

3.2 Workload Design

Various workloads are designed to evaluate the buffer replacement strategies in both single-user and multiuser environments. For single-user tests, five workloads aiming at five access patterns are developed. For multiuser tests we use the TPC-H and TPC-C Benchmark.

3.2.1 Single-User Test

Empirical studies of database reference behavior classify the cache access patterns into several types[12,16,17]. In this paper, our single-user workloads contain the following five access patterns.

3.2.1.1 Sequential References

Sequential references refer to a sequence of one-time-only use requests where all blocks are ac cessed one after another, and never re-accessed. This access pattern may flush out possibly important pages in buffer cache and decrease system performance. For example, the operation to a database is a SELECT count(*) against a large table A. Then data blocks of table A will take up many pages in cache. If all these pages are of no use in the future, this will consequently cause extra physical reads. If one replacement strategy can deal with this problem well, it is considered as scan-resistant.

Workload-the SQL operation SELECT count(*) FROM A is performed, where the size of A is a little smaller than the buffer size.

3.2.1.2 Correlated References

Correlated references mean that repeated re-references to the same pages take place in a short span of time, but these pages are relatively infrequently referenced overall. The repeated re-references are called correlated references which can not be regarded as hints that these pages will be referenced frequently. In RDBMS, this is likely to happen under update or delete operations, first reading a row in a page and later updating or deleting a value in the row. The page is continuously referenced but this does not demonstrate that the page is a hot page. We will examine whether buffer replacement strategies can filter out correlated references.

Workload-two similar workloads are generated. The first workload consists of a series of SELECT operations. The second workload is constructed by replacing some SELECT operations in the first workload with DELETE operations. Once the pages are accessed by the SELECT and DELETE operations, they will not be referenced again. The second workload has some correlated references but the first doesn't.

3.2.1.3 Looping References

Looping references mean that all blocks are accessed repeatedly in a regular interval. In a nested loop join, if there is not any index on the inner table, a full-table scan should be repeated many times. This behavior is similar to looping references. For looping references, the more pages that have been visited are in cache, the better the performance is.

Workload-a SELECT operation on a table is repeated many times.

3.2.1.4 Temporally-clustered References

Temporally-clustered references mean that blocks accessed more recently are the ones more likely to be accessed in the near future.

Workload-a table is scanned along one direction, then rescanned in the reverse direction and the two scan operations are performed once again.

3.2.1.5 Probabilistic References

For probabilistic references, each block has a stationary reference probability, and all blocks are accessed independently with the associated probabilities. The common example is to scan a table using a B-tree index.

Workload-the workload of probabilistic references is generated through index scans.

3.2.2 Multiuser Test

OLAP and OLTP are two typical workloads for DBMS. Two kinds of benchmarks are developed to simulate OLAP and OLTP applications. One is TPC-H Benchmark for OLAP applications and the other is TPC-C Benchmark for OLTP. They are both multiuser tests.

3.2.2.1 TPC-H Benchmark

The TPC Benchmark H (TPC-H)[14] is a performance benchmark established by the Transaction Processing Council (TPC) to demonstrate Data Warehousing/Decision Support Systems (DSS). It consists of a set of business oriented ad-hoc queries and concurrent data modifications. These queries and updates are executed against a standard database under controlled conditions.

We study the different behavior of buffer replacement strategies on OLAP workload using the TPC-H Benchmark.

3.2.2.2 TPC-C Benchmark

The TPC Benchmark C(TPC-C)[15] is comprised of a set of basic operations designed to exercise system functionalities in a manner representative of complex OLTP application environments. The Company portrayed by the benchmark is a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. As the Company's business expands, new warehouses and associated sales districts are created. The components of the TPC-C database are defined to consist of nine separate and individual tables and there are five transaction types in TPC-C with select, update, insert, delete and join operations.

TPC-C is a multi-user workload containing multiple access patterns. We compare various buffer replacement strategies on TPC-C Benchmark using several performance metrics.

3.3 Performance Metrics

In this section, we explain five performance metrics: pollution ratio, difference of physical reads, logical reads, hit ratio and execution time. Hit ratio is widely used in comparisons of various strategies and almost becomes the unique way in simulation evaluation method. In this work, direct measurement method is used, so execution time can be exactly obtained and it is included in our metric set. Physical reads and logical reads seldom appear as performance metrics in publications. In this paper, they are used in novel manners. In addition, a new metric pollution ratio is proposed for sequential references to capture the abilities of various algorithms to deal with sequential references, which is defined as follows.

3.3.1 Pollution Ratio

$$pollution\ ratio=\frac{N_{sr}}{number\ of\ pages\ in\ the\ cache}$$

where N_{sr} is the number of pages loaded and not evicted after sequential accesses.

The higher the pollution ratio is, the more seriously the buffer cache is "polluted", which demonstrates the buffer replacement strategy can't deal with sequential references well.

3.3.2 Difference of Physical Reads

Physical reads refer to the number of data blocks read from disk. We use the difference of two physical reads of two similar workloads (the details about the two similar workloads are depicted in Section 3.2.1) as the quantitative metric of the buffer replacement strategies on the problem of correlated references. If one buffer replacement strategy can filter out correlated references very well, the difference of the two physical reads should be zero.

3.3.3 Logical Reads

Logical reads refer to the number of read requests. If the page is not in the buffer cache, a physical read is then performed to read the page from disk into the buffer cache. If the page is in the cache, no physical read is generated. At the same time interval, the larger the number of logical reads, the higher the system throughput. We regard logical reads in the same span of time as the overall performance metric in our TPC-C test. In TPC-H test, because the running time of different algorithms is not the same, we use throughput as the overall system performance metric.

3.3.4 Hit Ratio

Hit ratio is defined as the ratio of the difference of logical reads and physical reads to logical reads.

$$hit\ ratio=\frac{logical\ reads-physical\ reads}{logical\ reads}$$

3.3.5 Execution Time

Execution time is the average CPU time spent by a logical read. It refers to the average CPU execution time interval from the moment when the read request reaches the buffer cache to the moment when database engine gets the page. Both the concurrency control mechanism on buffer pages and the overhead of algorithms have some impacts on this metric.

We examine the execution time of various replacement strategies by running TPC-H and TPC-C Benchmark, and show how it affects the overall system performance.

3.4 Significance Tests

In previous studies about various buffer replacement algorithms, such conclusions could usually be drawn: Algorithm A "significantly outperforms" B, or these algorithms are close. But these claims were not specified precisely-different standards were used by researchers to come to the same conclusions and no statistical significance analysis was conducted to verify the results. In this paper, in order to effectively and precisely demonstrate and explain the evaluation results, we introduce two statistical significance tests into comparisons of buffer replacement strategies, and propose a novel method to jointly use these two significance tests.

3.4.1 Sign Test

Sign test is used to compare two buffer replacement strategies based on the paired performance metrics such as hit ratio or logical reads. The notation is as follows.

(1)N is the number of different cache sizes.

(2)x_i is the performance metric of buffer replacement strategy X at the ith buffer cache, $i=1,2,3,\cdots,N$.

(3)y_i is the performance metric of buffer replacement strategy Y at the ith buffer cache, $i=1,2,3,\cdots,N$.

(4)n is the number of times that x_i and y_i differ.

(5)k is the number of times that x_i is larger than y_i.

The null hypothesis H_0 is $k=0.5n$, or k has a binomial distribution of $Bin(n,p)$ where $p=0.5$. The alternate hypothesis H_a is $k>0.5n$ ($k<0.5n$), which means that buffer replacement strategy X is better (worse) than buffer replacement strategy Y. The P-value indicates the probability of the observed evidence against the null hypothesis. The smaller the P-value, the more contradictory is the evidence to H_0. The P-value (one-sided) can be calculated using the following equations.

$$P=\begin{cases}\sum\limits_{i=k}^{n}\binom{n}{i}*0.5^{n} & \text{if } k\geqslant 0.5n\\ \sum\limits_{i=0}^{k}\binom{n}{i}*0.5^{n} & \text{if } k\leqslant 0.5n\end{cases}$$

3.4.2 T-test

T-test also uses the paired performance metrics to compare two buffer replacement strategies. Besides the same notation used in sign test, we add the following items.

(1)$d_i=x_i-y_i$ is the difference of x_i from y_i.

(2)d is the average of the d_i values for $i=1,2,\cdots,n$.

The null hypothesis H_0 is $d=0$. The alternative hypothesis H_a is $d>0$ ($d<0$). The P-value can be computed by using the t-distribution with a $n-1$ degree of freedom.

3.4.3 Joint Method

Sign test only considers the sign of two values, and ignores the absolute difference. T-test focuses on the absolute difference, but is sensitive to outliers. Therefore, we propose a method to jointly use sign test and t-test.

For each pair of buffer replacement strategies, we can get two test results, one for sign test and one for t-test. Thus, for the thirteen algorithms we study in this paper, we can get two 13*13 tables, which are filled in with "=", ">", "<", ">>" and "<<". These signs show the gap of two buffer replacement strategies on some performance metric. If the P-value of buffer replacement strategy X and Y is greater than 0.05, it is thought that X and Y perform equally well and the corresponding cell is filled in with a "=". If the P-value is equal to or less than 0.01, we use ">>" ("<<") to show that the performance gap of two algorithms is very large. When the P-value is between 0.01 and 0.05, we think buffer replacement strategy X is better (worse) than Y to some extent and use ">" ("<") to denote it.

After getting the two tables for the two significance tests, we compute the score of every buffer replacement strategy in each test. We denote the weights of "<<", "<", "=", ">", ">>" by $-\beta$, $-\alpha$, γ, α, β respectively. For each buffer replacement strategy, we calculate the score of the strategy by adding up the weights of the signs in the corresponding row. The final score of a buffer replacement strategy is the average of the two scores that are calculated from sign test and t-test.

In our test, we set α, β and γ to 1,2 and 0 respectively. The choice of the values represents the relative importance of "<<" and "<". If the values are changed, the ordering result may be somewhat different.

4 Buffer Replacement Strategies

In this section, we briefly introduce thirteen buffer replacement strategies we evaluated: LRU[1], CLOCK[2], GCLOCK[3], FBR[4], LRU-K[5] 2Q-Simple[6], 2Q-Full[6], LRFU[7], ARC[8], LIRS[9], CAR[10], CART[10] and CLOCK-Pro[11]. These strategies nearly cover all of the existing strategies that belong to the same category-trace and utilize history information to replace pages. Besides these algorithms, there also exist other strategies such as SEQ[18], EELRU[19], DEAR[20], etc. In this paper, we do not consider these algorithms because they belong to another type that detects and adapts to access regularities[9,21].

LRU is a classical buffer replacement policy, which was developed originally for patterns of use in instruction logic[22,23]. Because pure LRU has an unaffordable cost in some areas such as CPU cache, CLOCK and GCLOCK appear to simulate the LRU replacement strategy. LRU can't do well in some applications because it only utilizes limited access information (recency of references). So researchers propose new replacement algorithms, such as LRU-K, FBR, LRFU and LIRS to collect and use "deeper" history information. In order to mitigate logarithmic implementation complexity of LRU-K, 2Q-Simple and 2Q-Full are proposed which reduce the complexity to constant per requ est. All these policies require user-defined parameters except LRU and CLOCK. In order to remove this limitation, a self-tuning algorithm-ARC is proposed. But this strategy also has some disadvantages, such as the cache hit serialization problem. CAR and CART policies are proposed to improve on ARC. CLOCK algorithm is still plagued by disadvantages of LRU, such as the disregard of "frequency". Inspired by LIRS, CLOCK-Pro is proposed to improve on CLOCK.

4.1 LRU

LRU(Least Recently Used)[1] is one of the simplest and most popular algorithms. When the buffer cache is full and room is needed for a new page, this strategy selects the page with the largest recency for replacement. LRU is easy to implement with constant time and space overhead. This algorithm only considers the recency of pages, and does not capture the "frequency". For a looping access pattern that is larger than the cache size, LRU always replaces the blocks that will be used soon, leading to reference misses.

4.2 CLOCK

CLOCK[2] is used to simulate the LRU replacement algorithm. It regards the whole cache as a circular queue, using a clock pointer pointing to the victim page that may be replaced. When a cache hit occurs, no page movement is needed, so CLOCK can lighten lock contention and reduce algorithm complexity. It is often used in environments where there are significant complexity constraints.

4.3 GCLOCK

GCLOCK (Generalized CLOCK)[3] is a variation of the basic CLOCK algorithm. In the CLOCK algorithm the referen ce bit of each page is set to one on a buffer page hit, while in GCLOCK algorithm the reference counter associated with each page can be incremented to a page-related weight. The weight value represents a tradeoff between the accuracy and the speed of the clock-sweep algorithm. A large value makes GCLOCK approximate the LRU strategy. But it will result in too many times of clock sweeps to find a page to replace. So, in our implementation, we don't want it to be very large and set it to 5.

4.4 FBR

FBR(Frequency-based Replacement)[4] policy maintains a LRU list, and divides the list into three sections: new, middle and old. Each page has a reference counter. If a cache hit happens, the page is moved to the MRU position of the new section; moreover, if the page was in the middle or the old sections, its reference counter is incremented. But if it was in the new section, the reference counter is not changed. Thus, FBR can resolve the correlated reference problem by factoring out locality. On a cache miss, the page with the smallest reference count in the old section is replaced. The advantage of FBR is that it combines recency and frequency. A limitation of this strategy is that it has to periodically decrement the reference counters to prevent cache pollution due to the stale pages with high reference count. The algorithm also has several tunable parameters, such as the sizes of three sections, and we set these parameters as in the original paper.

4.5 LRU-K

LRU-K[5] takes into account the history knowledge of the last $K(K\geqslant 2)$ references in deciding the victim block for eviction. The authors thought of LRU-2 as a better algorithm than LRU-K where $K>2$, so we implemented LRU-2 in the experimental study. LRU-2 remembers the last two times when one page was requested, and replaces the page with the least recent penultimate reference. A limitation of the LRU-2 algorithm is that it needs logarithmic implementation complexity. It also has two tunable parameters: correlated reference period and retained information period. We use a heap to store the pages within the correlated reference period and a list to record the history information. The sizes of the heap and the list are the same as the parameter values of 2Q, because 2Q is an approximation of LRU-2.

4.6 2Q

2Q(Two Queue)[6] was proposed to reduce the logarithmic complexity of LRU-2. It has two types of algorithms: 2Q-Simple and 2Q-Full. Like LRU-2, they use the recency of several past references to decide the page to be evicted. In 2Q, a simple LRU list is used instead of the heap structure in LRU-2. 2Q-Full algorithm maintains a history page information list, while 2Q-Simple does not. We set their parameters as the authors suggested: *Kin* is 25% of the cache size and *Kout* is 50% of the cache size. Therefore, according to these values, the size of the heap in LRU-2 is set to 25% of the cache size and the history list to 50% of the cache size.

4.7 LRFU

LRFU(Least Recently/Frequently Used)[7] algorithm also combines recency and frequency of references. Each page is associated with a CRF value, and each reference to the page contributes to the value, which is used to choose the page for eviction. LRFU has an important parameter λ, which decides whether LRFU collapses to LRU algorithm or LFU algorithm. Therefore, the complexity of LRFU fluctuates between constant and logarithmic in cache size per request. According to the experimental results the authors provided, for most of various cache sizes, the performance of LRFU becomes stable when λ is varied from 0.001 to 0.01. So, in order to acquire stable performance results we set λ to 0.005 except specified otherwise. Besides λ, LRFU has another parameter c, which represents the correlated period. Similar to those of 2Q and LRU-2, we set c to 25% of the cache size.

4.8 LIRS

LIRS(Low Inter-reference Recency Set)[9] algorithm regards IRR(Inter-Reference Recency) as the recorded history information of each page, where IRR of a page refers to the number of other blocks accessed between two consecutive references to the page. LIRS maintains a variable-size LRU stack to keep the pages that have been seen recently. The referenced blocks are divided into two sets: High Inter-reference Recency(HIR) block set and Low Inter-reference Recency(LIR) block set. The minor part of the cache with the size of L_{hirs} is used to store blocks from HIR block set. The parameter L_{hirs} can crucially affect the performance of LIRS. We set it to 1% as in the original paper and the size of LRU stack to 1.5 times as large as the cache size. In the expected case LIRS has a constant complexity, but it requires “stack pruning” operation that in the worst case may touch many pages in the cache which increases the implementation complexity. CLOCK-Pro[11] algorithm can be regarded as a variant of LIRS, which combines CLOCK algorithm and LIRS algorithm.

4.9 ARC

ARC(Adaptive Replacement Cache)[8] is a self-tuning policy that combines recency and frequency and requires no user-specified parameters. The whole cache is divided into two queues, and each queue is managed by LRU policy. The two queues separate the pages that have been only visited once recently from those which have been seen at least twice recently. The size of every queue changes according to the evolution of the workloads. ARC also maintains two history information lists, and a hit to a block in the history lists causes the block to move to the second queue. ARC has a low overhead complexity similar to LRU. CAR[10] is a variant of ARC, which uses CLOCK to substitute the LRU queue in ARC. So, it has the advantages of both ARC and CLOCK algorithms. It removes the cache hit serialization problem of ARC: it only needs to set the bit of the hit page and does not need to lock the whole queue to move the page. CART[10] strategy is similar to CAR and employs a much stricter criterion to distinguish pages with short-term utility from those with long-term utility. Therefore, it can deal with correlated references better.

5 Experimental Evaluation

For single-user tests, our experiments were performed on a PC with a 2.0 GHz CPU, 1 GB memory and one IDE disk. SMP machines were used in multiuser tests. The OS buffer cache is disabled in all the tests. Using the new evaluation methodology, we measure the performance of thirteen buffer replacement algorithms when dealing with five access patterns and running TPC-H and TPC-C Benchmark in KingbaseES system. In the following sections, we will show and analyze the experimental results.

5.1 Sequential References

In this experiment, the buffer cache of KingbaseES contains 12 000 pages, and each page holds 8 KB. There is a large table A in our database, which includes 11 113 data blocks. We performed a SELECT count(*) operation against table A after the buffer cache had been used by other workload for a period of time. Figure 1 shows the pollution ratio of various replacement algorithms.

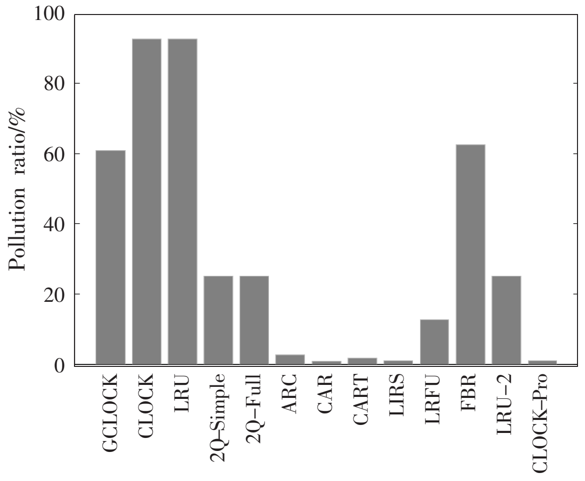


Fig.1 Sequential references

图 1 顺序访问

The pollution ratios of GCLOCK, CLOCK, LRU and FBR are higher than those of 2Q-Simple, 2Q-Full, LRFU and LRU-2 algorithms, and the values of 2Q-Simple, 2Q-Full, LRFU and LRU-2 algorithms are higher than those of ARC, CAR, CART, LIRS and CLOCK-Pro. The maximal pollution ratio in the figure is 92.6%, while the minimum is less than 1%.

If all the data blocks of table A are in the buffer cache, the pollution ratio is 92.6%(11 113/12 000). Among all the thirteen algorithms the pollution ratios of CLOCK and LRU reach the maximum, which demonstrates the buffer cache is "polluted" most seriously when using CLOCK and LRU strategies. The two algorithms are not scan-resistant. The GCLOCK replacement strategy considers reference counts when it decides the page to evict, and every page from table A is only referenced once, so some pages of table A are replaced, thus the pollution ratio of GCLOCK is lower than those of CLOCK and LRU. How many pages of table A will be replaced is decided by the status of the buffer cache, and this status is decided by the previous workload.

The pollution ratios of 2Q-Simple, 2Q-Full and LRU-2 algorithms are all 25%. If we change their parameters, the pollution ratios change too, and the result value is always equal to the parameter. So the abilities of these algorithms to deal with sequential references depend on their parameter settings. The same is true of LIRS, LRFU and FBR algorithms. The pollution ratio of LIRS is equal to its parameter L_{hirs} and the pollution ratio of LRFU changes with the parameter λ. The parameters of the ARC, CAR, CART and CLOCK-Pro strategies vary with the workloads. We compare their pollution ratios with the values of their parameters before executing sequential references and find that the pollution ratios are decided by the parameters. So the workload characteristics before sequential references occur decide the robustness of these buffer replacement strategies.

To sum up, we can draw the following conclusions:

(1)LRU and CLOCK algorithms are not scan-resistant.

(2)The robustness of 2Q-Simple, 2Q-Full, LIRS, LRFU, FBR and LRU-2 strategies on sequential references is decided by their parameters.

(3)The abilities of GCLOCK, ARC, CAR, CART and CLOCK-Pro to deal with this problem are decided by the characteristics of the previous workload.

5.2 Correlated References

We constructed two workloads according to the method described in Section 3.2.1.

If one buffer replacement strategy can filter out correlated references, the number of physical reads on the second workload should be identical to that on the first workload. Figure 2 shows that GCLOCK, CLOCK, LRU, 2Q-Full, LIRS, LRFU, FBR, LRU-2 and CLOCK-Pro algorithms can deal with correlated references well. The authors of 2Q-Full, LRFU,

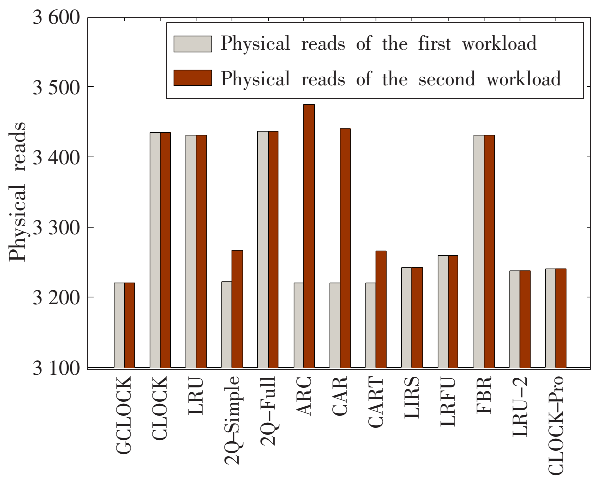


Fig.2 Correlated references

图 2 相关访问

FBR and LRU-2 considered this pro-blem when they designed the algorithms. For example, the FBR strategy uses a new section to filter out correlated references, and LRFU revises CRF values of the pages. Though other algorithms were not designed for this purpose, they can also solve this problem well. For example, the LRU replacement strategy only uses the recency information while ignoring reference count. In CLOCK-Pro, cold pages can be turned into hot pages only after several references.

2Q-Simple, ARC, CAR and CART can't eliminate the effect of DELETE operations. 254 and 219 extra physical reads occur in ARC and CAR algorithms respectively. The numbers of extra physical reads (=45) in 2Q-Simple and CART are less than those of ARC and CAR. One common feature of 2Q-Simple, ARC and CAR is that, if a page is requested once, it will be regarded as a hot page. So they perform poorly on this kind of workload. CART improves CAR and ARC on this problem, but there still exist some extra physical reads.

5.3 Looping References

For looping references, two experiments are designed. In the first experiment, the cache size is smaller than the number of the referenced pages in one iteration, and in the second experiment, the cache size is slightly larger than the number of the referenced pages.

5.3.1 Smaller Cache Size

We set the cache size at 11 000 pages, and performed the same SELECT operation ten times against the table A, which has 11 113 pages (slightly greater than the cache size). Before the SELECT operations, the cache was clean.

Figure 3 shows the hit ratio of each iteration. Except 2Q-Full, LIRS, LRFU, LRU-2 and CLOCK-Pro, the hit ratios of all the algorithms are all 0%. Although for each iteration, the same pages are referenced, these algorithms do not learn the characteristics of the access pattern after seeing the pages repeatedly. They always replace the pages which will be accessed again. This leads to the complete cache misses of the next iteration, and all the pages need to be read from disk again.

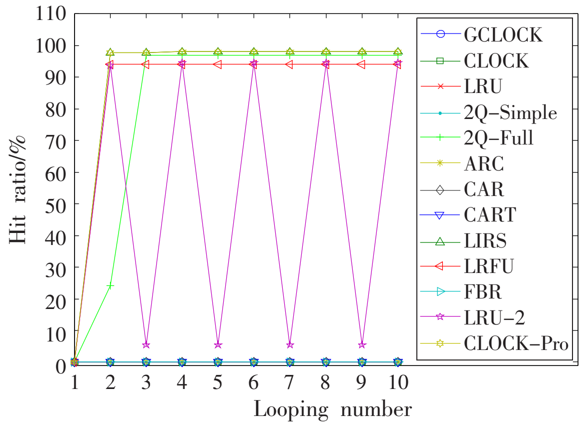


Fig.3　Looping references

(the buffer cache has not been used by other workloads)

图 3　循环访问(缓冲区没有被其他工作负载使用)

Although the hit ratios of 2Q-Full, LIRS, LRFU, LRU-2 and CLOCK-Pro are greater than 0%, they do not act identically. Firstly, their maximal hit ratios are different. The maximums of LIRS and CLOCK-Pro are larger than those of other algorithms. CLOCK-Pro is inspired by LIRS, so there are some similarities in their behavior. 2Q-Full has slightly higher hit ratios than LRFU and LRU-2. Secondly, the speeds that they reach their peak values differ from each other. LRFU and LRU-2 reach their peak values during the second iteration, and the other three algorithms stabilize after the fourth iteration. Finally, the stabilization states are different. The hit ratios of LRU-2 fluctuate between its peak value and valley value, and it is interesting that the sum of peak value and valley value is 100% exactly. Other algorithms stabilize after they reach the peak values.

If the buffer cache was not clean before running our workload, the hit ratios of all the algorithms were 0%. So the algorithms can not adapt well to this pattern after other workloads have occurred.

5.3.2 Larger Cache Size

In this experiment, the cache size is greater than the number of pages accessed by each SELECT operation against table B which has 1 701 pages. Before the experiment, the buffer cache had been used by other workloads. Since the cache size is greater than 1 701, the hit ratio after several iterations should be able to reach 100%. Figure 4 shows the result when the cache size is 2 000 (Throughout this paper, the cache size generally refers to the number of the pages the cache contains).

GCLOCK, CLOCK and LRU show good adaptability. Their hit ratios reach 100% in the second iteration. ARC, CAR, CART and CLOCK-Pro are slightly worse, and the hit ratios also reach 100% after two iterations. LRU-2 does not adapt to this access pattern well. Though the hit ratio increases with the number of iterations, the values are much lower. The hit ratios of the other algorithms are all 0%.

We increased the cache size to 4 000, and tested 2Q-Simple, 2Q-Full, LIRS, LRFU, FBR and LRU-2 again. In Figure 5, we can see that, except 2Q-Simple, all the algorithms have hit ratios of 100% after several iterations. We continued the experiment with larger cache size. When the cache size is 6 000, the maximal hit ratio of 2Q-Simple is still 0%, and the hit ratio does not reach 100% until the cache size is close to 7 000. It is because, for this access pattern, 2Q-Simple uses only FIFO queue, i.e., there are only 25% pages used. When the cache size is 7 000, and 1 750(=7 000*25%) is greater than the number of pages of table B(=1 701), all the pages of table B can be held in the FIFO queue.

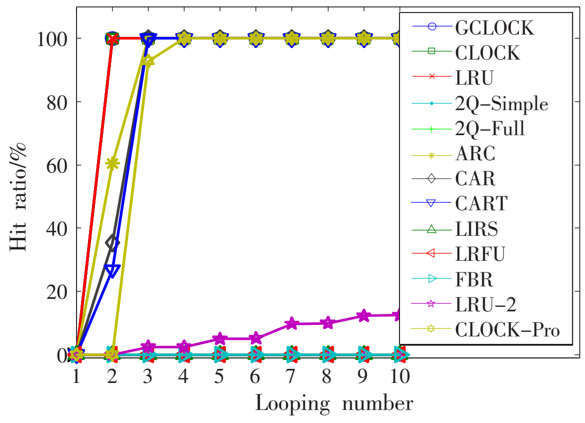


Fig.4 Looping references
(the buffer cache contains 2 000 pages)
图 4 循环访问(缓冲区有 2 000 个页面)

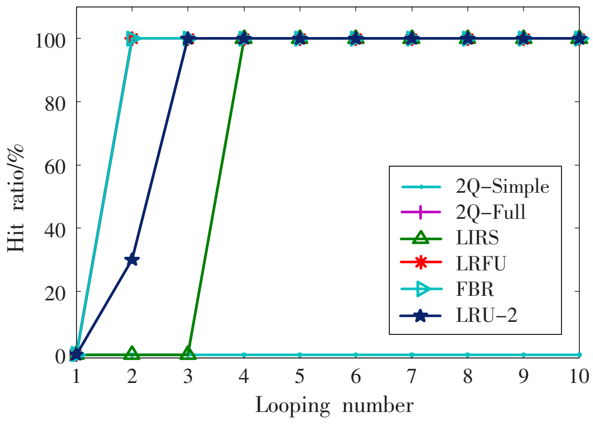


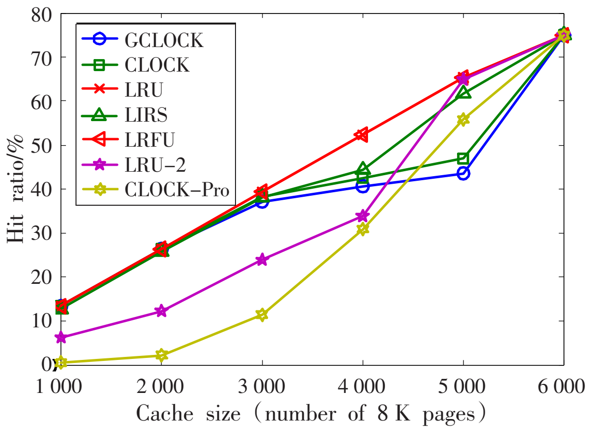
Fig.5 Looping references
(the buffer cache contains 4 000 pages)
图 5 循环访问(缓冲区有 4 000 个页面)

5.4　Temporally-Clustered References

In this workload, more than 5 000 pages are accessed in one scan. The hit ratios on this workload are shown in Figure 6. In order to show each line clearly, we put the thirteen curves in two figures. We display the 2-3 best algorithms and the 2-3 worst algorithms in the first figure, and the others in the second figure.

This experiment verifies the argument that LRU is the optimal policy under the access pattern of temporally-clustered references[23]. We can see that the hit ratio of LRU is proportional to the cache size. For LRFU, the parameter λ is set to 1.0. With this setting, the LRFU algorithm is similar to LRU, so its hit ratios are equal to those of LRU. When the cache size is small, the hit ratios of GCLOCK and CLOCK are very close to those of LRU. There is a large gap between the hit ratios of CLOCK-Pro, LRU-2 and those of LRU. The difference between the hit ratios of other algorithms and those of LRU is not so large. As we increase the cache size, the curves of GCLOCK, CLOCK, 2Q-Full, CAR and CART depart further and further from that of LRU. There also exists an obvious gap between the hit ratios of 2Q-Simple, LIRS and those of LRU. When the cache size is large, the curves of 2Q-Simple, ARC and FBR almost overlap with that of LRU. CLOCK-Pro, LRU-2 and LIRS also begin to approach LRU. However, the curves of GCLOCK, CLOCK, 2Q-Full, CAR and CART are still far away from that of LRU. When the cache size is 6 000, the hit ratios of all the algorithms converge to the same point.

Here, we investigate the recency of the replaced pages to help to explain the performance of the replacement strategies on temporally-clustered patterns. The most significant difference among the replacement strategies is the pages to be replaced. Thus, studying the characteristics of these pages can help us to understand the replacement strategies. The recency of a page is defined as the distance between the last access time and the current time. This time refers to logical time. Consider a reference sequence *abcdefg*, page *a*'s access time is 1, page *g*'s access time is 7, then the distance between *a* and *g* is 6. The characteristic of temporally-clustered patterns is that the page which was just accessed has higher possibility to be accessed



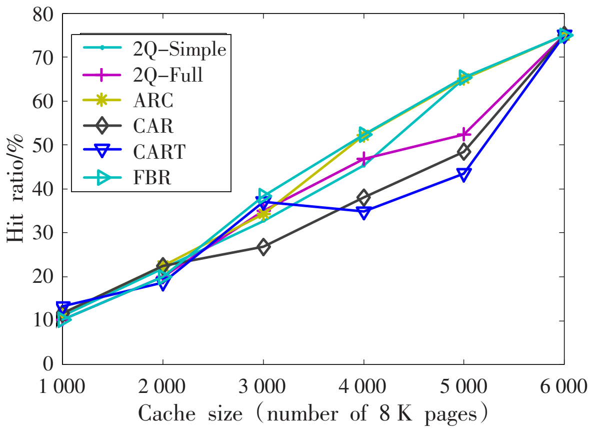


Fig.6　Temporally-clustered references

图 6　时间聚集访问

again, i.e., for this access pattern, a replacement strategy should keep the pages with smaller recency in the cache, and replace the pages with larger recency.

The recency distribution of LRU at different cache sizes has the same characteristic-the recency of every replaced page is equal to the cache size. For example, when the cache size is 5 000, the recency of all replaced pages is also 5 000. Under this workload, if the page with the largest recency is always evicted, the recency of each replaced page should be equal to the cache size.

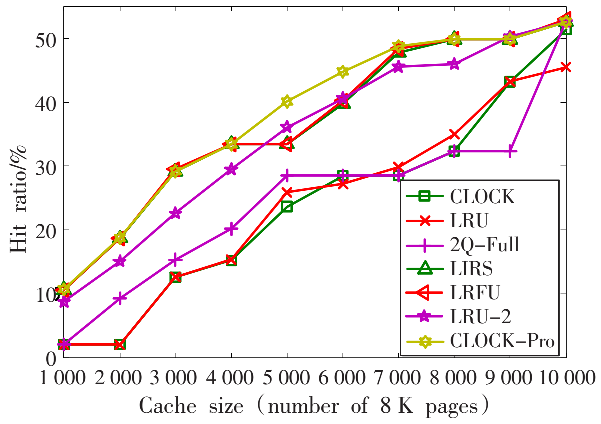
We find that the pages replaced by LRFU have the same characteristic as those of LRU, so LRFU has the same performance as LRU. When the cache sizes are 4 000 and 5 000, the recency of the pages replaced by FBR is identical to the cache size. Therefore, its hit ratios are equal to LRU's.

If the recency distribution of one algorithm is different from that of LRU, we need to analyze the result by combining recency with other characteristics of the replaced pages and the way the buffer algorithm works. When the cache size is 1 000, the hit ratios of GCLOCK and CLOCK are close to that of LRU. In the recency distribution of CLOCK, the recency of some pages is equal to the cache size, and the recency of other pages changes from 1 to 1 999. We analyze the reference counts of the replaced pages and find that the pages whose recency is not 1 000 are accessed twice. Under this workload, the pages accessed twice will be referenced after a long time, so it is reasonable to replace these pages.

When the cache sizes are 1 000 and 2 000, LIRS replaces the pages whose recency is 1% of the cache size, i.e., the value of L_{hirs}. It is because LIRS always replaces the pages in the LRU queue whose size is L_{hirs} of the cache size. So, in this case LIRS has relatively higher hit ratios. CLOCK-Pro does not perform well, although it is inspired by LIRS. Perhaps, this is due to their replacement policy-algorithms using "reuse distance" in replacement decision don't perform very well for temporally-clustered patterns.

5.5 Probabilistic References

The workload of probabilistic references can be generated by index scans. Figure 7 shows the hit



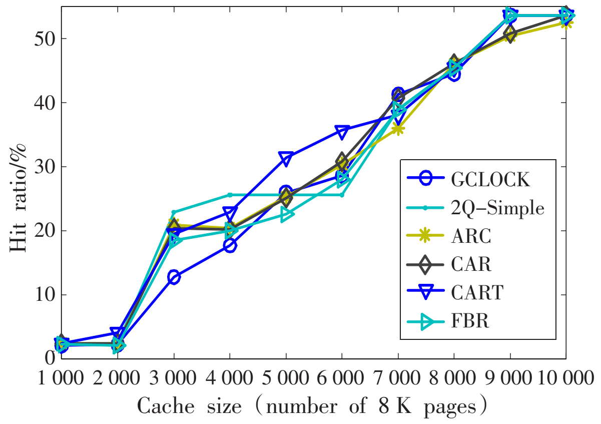


Fig.7 Probabilistic references

图 7 概率访问

ratios of the thirteen algorithms when the cache size is varied from 1 000 to 10 000. In this experiment, the λ parameter of LRFU is set to 0, which makes LRFU similar to LFU.

We can see that CLOCK-Pro, LRFU, LIRS and LRU-2 have relatively higher hit ratios. The hit ratios of CLOCK and LRU are much lower than those of other algorithms. When the cache size is less than 6 000, there is a gap between the hit ratios of CLOCK-Pro, LRFU, LIRS and LRU-2 and those of other algorithms. But when the cache size increases, the hit ratios of these algorithms begin to approach those of CLOCK-Pro except the 2Q-Full algorithm.

For probabilistic patterns, the buffer replacement strategy which always chooses the pages with the smallest reference counts to replace is the optimal[23,24]. Theoretically, CLOCK-Pro, LRFU, LIRS and LRU-2 replace the pages accessed only once with high priority, so they deliver high hit ratios. However, LRU and CLOCK do not consider the frequency information, so they do not perform well on this access pattern.

In order to explain the results of these algorithms, we trace the reference counts (frequency) of the replaced pages. We find that the reference counts of the pages replaced by CLOCK-Pro are mostly one. In Figure 8, we show the rate between the number of replaced pages whose reference counts are greater than one and the number of all replaced pages. When the cache size is 5 000, the rate is highest, and it is only about 4%.

LRFU, LRU-2 and LIRS have similar features as CLOCK-Pro. Because there are many pages whose reference counts are the same, different algorithms may replace different pages, which leads to different hit ratios.

Theoretically, FBR first replaces the pages whose frequency is low, but the algorithm decreases the reference count of every page periodically and there are other conditions that decide which page can be replaced. Thus, as shown in Figure 9, the reference counts of the replaced pages are often larger than one.

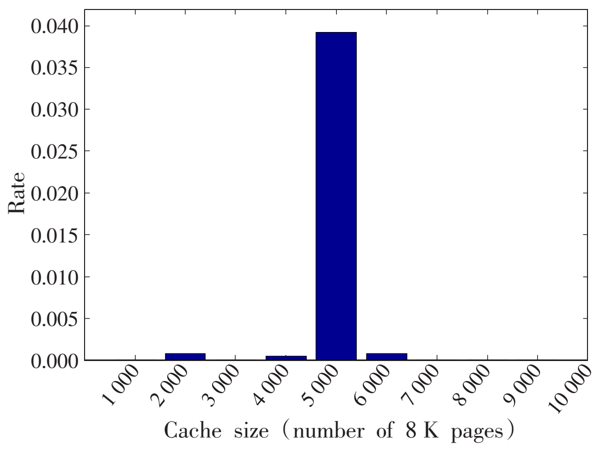


Fig.8 Frequency distribution of CLOCK-Pro

图 8 CLOCK-Pro 的频率分布

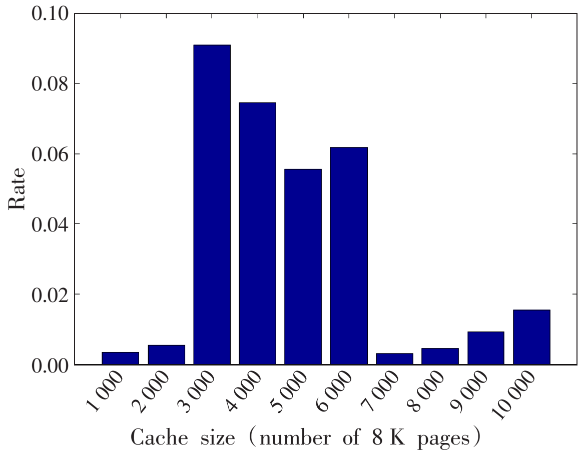


Fig.9 Frequency distribution of FBR

图 9 FBR 的频率分布

Figure 10 and Figure 11 show that LRU and CLOCK often replace the pages whose reference counts are larger than one. The two algorithms do not consider the hints of frequency on the future behavior, so their hit ratios are low on probabilistic patterns.

The performance of other algorithms can also be analyzed by comparing the frequency information of the replaced pages. Here, we do not discuss it any more due to space limitation.

5.6 TPC-H Benchmark

In this section, we show the experimental results of the buffer replacement algorithms on TPC-H workload. TPC-H operates on a data warehouse of 1 GB, the size of which can be increased with a Scaling Factor(SF). We set *SF* to 1, so two users are used. In the throughput test, two query streams and one update stream are executed simultaneously. The experiment was done on an SMP machine, with 4 3.6 GHz CPUs, 5 GB memory and 2 SCSI disks. The cache size is 230 000.

Figures 12, 13, 14 show the throughput, hit ratios and execution time of the replacement strategies, respectively. From these figures, the following

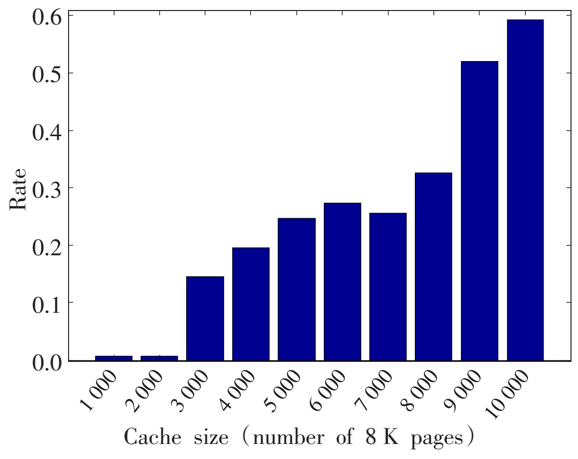


Fig.10 Frequency distribution of LRU

图 10 LRU 的频率分布

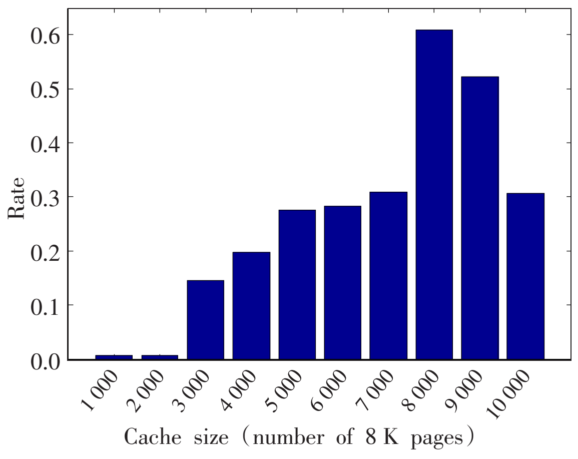


Fig.11 Frequency distribution of CLOCK

图 11 CLOCK 的频率分布

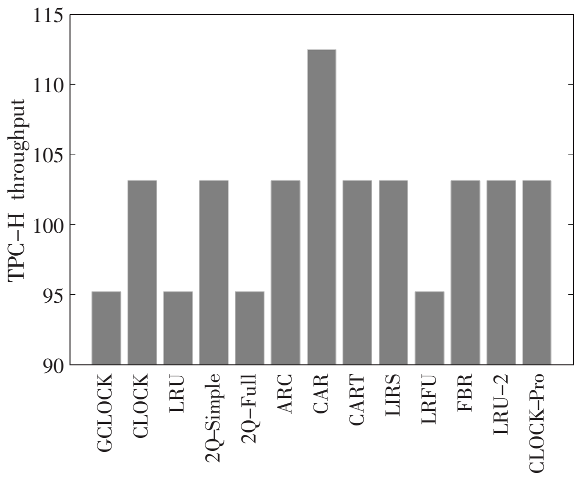


Fig.12 TPC-H throughput

图 12 TPC-H 吞吐量

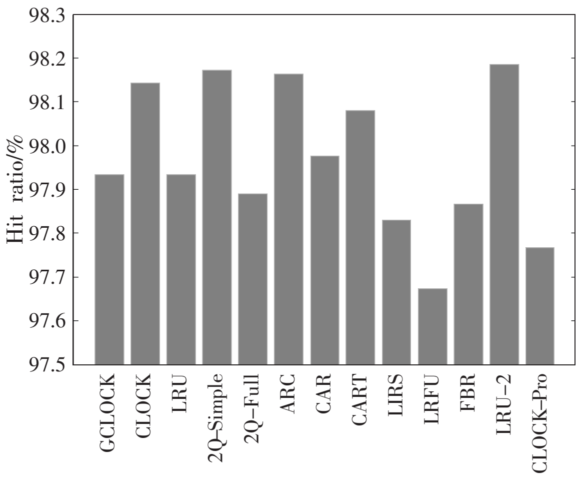


Fig.13 TPC-H hit ratio

图 13 TPC-H 命中率

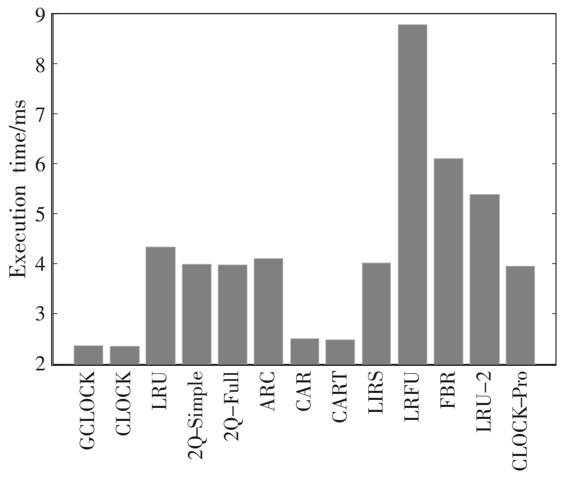


Fig.14　TPC-H execution time

图 14　TPC-H 执行时间

observations can be made. First, GCLOCK, LRU, 2Q-Full and LRFU have lower throughput than other algorithms, and the CAR strategy is the best. Many algorithms perform similarly. Second, the hit ratios of the algorithms show little difference. The highest hit ratio (LRU-2) is 98.19%, and LRFU has the lowest hit ratio which is 97.67%. Third, the relationship between throughput and hit ratio is complex. The low hit ratio can not always mean bad behavior in throughput. The CAR algorithm achieves the best throughput, while its hit ratio is not highest. Maybe this is related to the low execution time of CAR. Finally, as to execution time, these buffer replacement strategies perform quite differently due to the effect of implementation overhead and lock contention. In our test, LRFU uses the longest time to complete a logical read. GCLOCK, CLOCK, CAR and CART algorithms spend less time than others.

5.7　TPC-C Benchmark

In this section, we use three performance metrics to compare various buffer replacement strategies on TPC-C workload: logical reads, hit ratio and execution time. Our TPC-C database contains 10 warehouses (about 900 MB), and the running time of TPC-C is 1 200 seconds. The major part of the test was conducted on the PC used for single-user tests. In order to compare the algorithms in different environments, we also run the test in an SMP machine with 4 1.5 GHz CPUs, 2 GB memory and 3 SCSI disks (in Section 5.7.3).

5.7.1　Experimental Results

Figure 15 shows the hit ratios of the algorithms when the cache size is varied from 2 560 pages to 26 880 pages. For clarity, we list the hit ratios at ten different cache sizes in Table 1. Overall, the hit ratios of LIRS at different cache sizes are lower than the hit ratios of other algorithms. When the

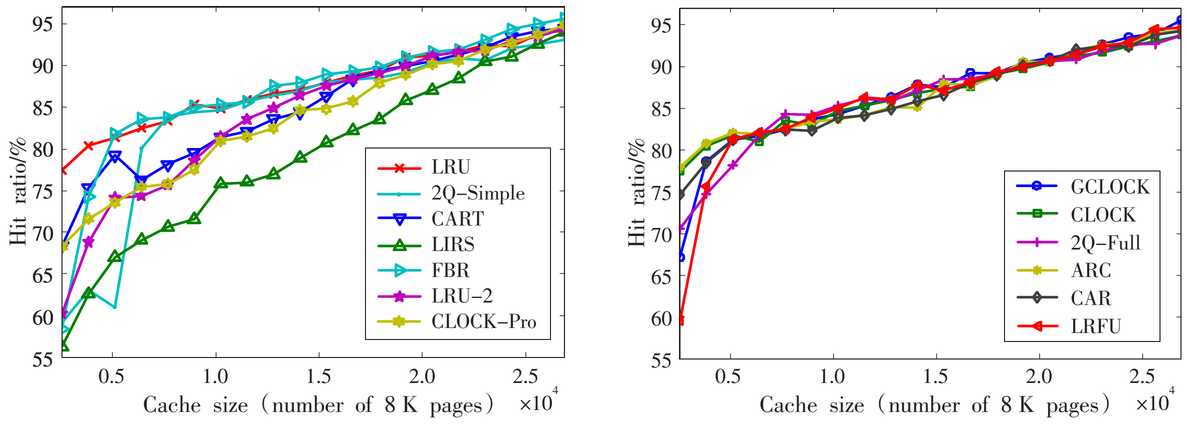


Fig.15　TPC-C hit ratio

图 15　TPC-C 命中率

cache size is smaller than 6 400 pages, the gap among the hit ratios of various algorithms is large. For example, the hit ratio of LRU algorithm is 80.36% when the buffer cache contains 3 840 pages, while the hit ratio of LRU-2 is only 68.79%. As the cache size increases, the hit ratios of the strategies except CART, CLOCK-Pro and LRU-2 are close. For large cache sizes, the hit ratios of all the strategies are similar to each other.

Figure 16 shows the logical reads of all the algorithms. Table 2 shows the logical reads at ten buffer cache sizes. The result indicates that, overall FBR performs best and LIRS performs worst. The curves of CART, CLOCK-Pro and LRU-2 algorithms are adjacent to each other, and those of other algorithms mix together.

We compare the results of Figure 15 and Figure 16 and find that the smaller the cache size,

Table 1 Hit ratio (the cache size is from 2 560 pages to 25 600 pages)

表 1 命中率(缓冲区大小为 2 560 到 25 600 个页面)

Algorithm	2 560	5 120	7 680	10 240	12 800	15 360	17 920	20 480	23 040	25 600
GCLOCK	67.17	81.21	82.42	84.26	86.31	87.54	89.22	91.02	92.64	93.96
CLOCK	77.42	81.81	83.54	84.67	85.97	87.37	89.10	90.51	91.66	93.07
LRU	77.46	81.32	83.36	84.83	86.59	87.89	89.41	91.19	92.48	93.76
2Q-Simple	59.07	60.97	83.83	84.66	86.39	87.97	88.52	90.27	90.63	92.52
2Q-Full	70.57	78.21	84.28	85.26	85.81	88.42	89.06	90.64	91.90	92.69
ARC	77.93	82.06	82.54	83.65	85.13	87.97	88.80	90.53	92.61	94.06
CAR	74.66	81.17	82.41	83.85	84.88	86.55	88.95	90.68	92.41	93.79
CART	68.30	79.19	78.12	81.38	83.55	86.30	89.34	90.54	92.13	94.14
LIRS	56.23	66.96	70.63	75.79	76.92	80.72	83.51	87.07	90.47	92.60
LRFU	59.58	81.32	82.54	84.94	86.00	87.11	89.30	90.67	92.44	94.42
FBR	58.38	81.78	83.74	85.27	87.53	88.95	89.76	91.62	93.06	94.97
LRU-2	60.22	74.15	78.68	81.45	84.90	87.58	89.05	91.16	91.78	93.48
CLOCK-Pro	68.26	73.56	75.81	80.92	82.46	84.81	87.90	90.15	91.89	93.70

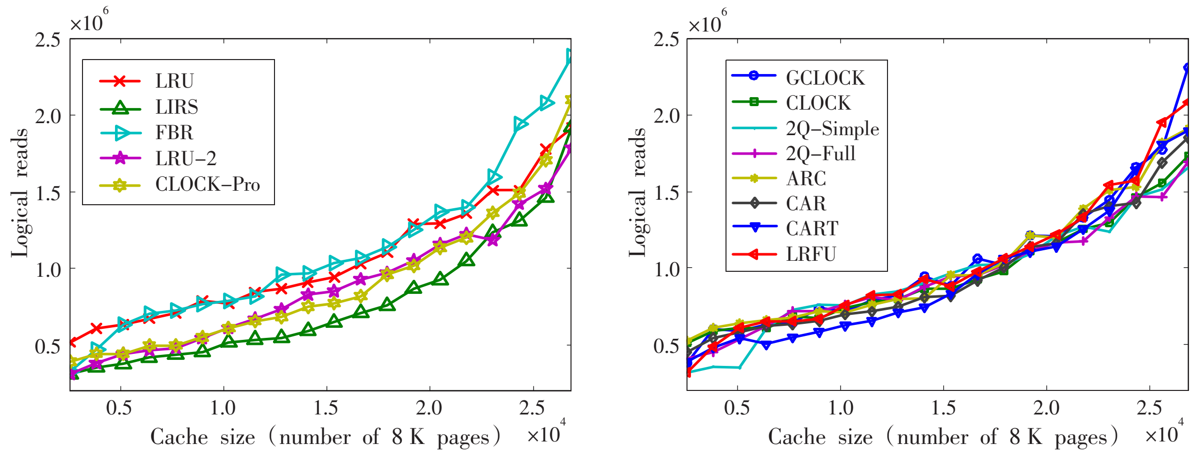


Fig.16 TPC-C logical reads

图 16 TPC-C 逻辑读

the larger the gap among the hit ratios of different buffer replacement strategies. But the curves of the corresponding logical reads are close to each other. When the cache size is large, though the hit ratios are similar, the difference of logical reads is large. This illustrates that the increase of hit ratio when hit ratio is high can affect system throughput more greatly than that when hit ratio is low.

The execution time is shown in Figure 17. The execution time of FBR algorithm is longest, because this strategy need decrease the reference count of each page periodically and its implementation is also complex. The execution time of CLOCK and GCLOCK algorithms is smaller than that of others because these two algorithms can decrease the number of lock operations and need not move pages.

The average time to acquire lock per logical read is shown in Figure 18. The curves are obviously clustered into two groups. GCLOCK, CLOCK,

Table 2　Logical reads (the cache size is from 2 560 pages to 25 600 pages)

表 2　逻辑读(缓冲区大小为 2 560 到 25 600 个页面)

Algorithm	2 560	5 120	7 680	10 240	12 800	15 360	17 920	20 480	23 040	25 600
GCLOCK	377 830	586 461	641 767	719 409	824 564	881 548	1 019 856	1 208 932	1 446 202	1 774 263
CLOCK	508 173	607 855	683 901	741 956	797 363	863 670	978 496	1 166 080	1 296 313	1 556 137
LRU	518 259	633 685	707 341	769 231	866 400	941 715	1 102 697	1 291 457	1 509 265	1 779 566
2Q-Simple	317 136	346 806	724 793	757 731	849 470	963 790	1 024 976	1 208 545	1 236 519	1 514 217
2Q-Full	408 569	534 200	718 816	760 831	790 757	946 086	1 017 744	1 167 491	1 332 018	1 465 666
ARC	523 517	639 303	674 283	711 590	796 854	954 574	1 032 512	1 198 269	1 508 357	1 822 291
CAR	449 301	580 449	632 466	696 482	746 309	815 799	1 003 981	1 161 477	1 402 106	1 687 767
CART	388 770	541 795	544 821	624 817	710 404	831 335	1 064 170	1 140 737	1 375 107	1 809 134
LIRS	309 928	379 227	437 166	518 187	546 639	648 710	756 657	924 039	1 234 359	1 461 276
LRFU	316 355	609 537	651 771	755 407	826 723	879 477	1 063 263	1 219 610	1 541 238	1 953 809
FBR	327 367	631 329	725 691	786 273	956 916	1 036 974	1 141 081	1 369 463	1 596 861	2 081 435
LRU-2	311 990	439 310	477 384	610 962	731 637	851 333	969 076	1 157 456	1 084 604	1 520 375
CLOCK-Pro	389 768	442 921	495 520	609 566	681 750	771 379	958 723	1 135 600	1 361 961	1 707 505

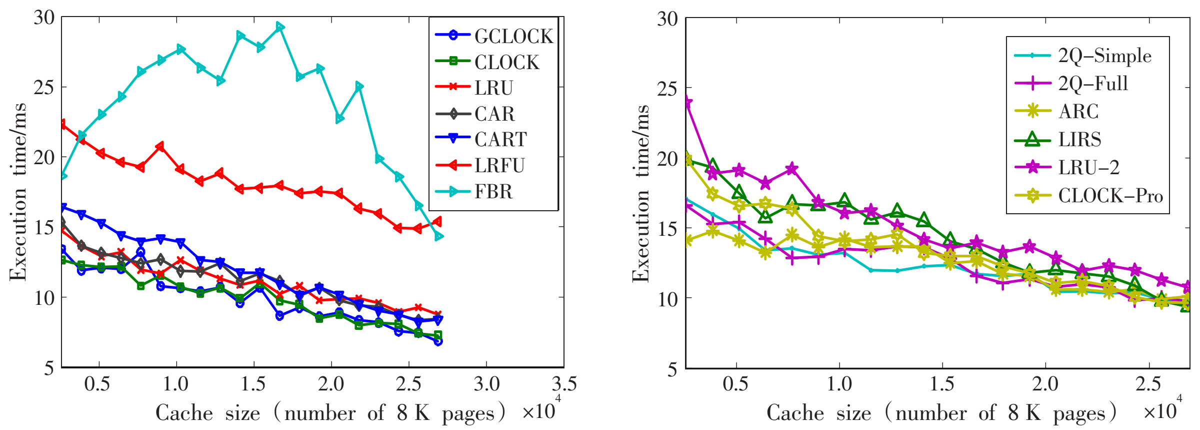


Fig.17　TPC-C execution time

图 17　TPC-C **执行时间**

CAR and CART algorit hms can lighten lock contention and spend less time in acquiring locks than other strategies. The lock cost of LRU is very high. Among the other algorithms, LIRS and CLOCK-Pro strategies spend more time in lock operations. In CLOCK-Pro algorithm, a global variable needs to be adjusted dynamically, and the corresponding lock is required, so CLOCK-Pro also has a high lock cost.

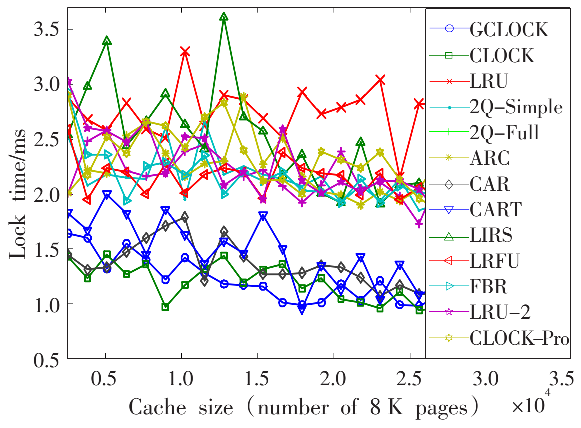


Fig.18 TPC-C lock time

图 18 TPC-C **封锁时间**

5.7.2 Significance Tests

In order to easily distinguish the behavior of all the buffer replacement strategies and precisely compare the performance, we used statistical significance tests for various performance metrics. Table 3 and Table 4 are the results of t-test and sign test on hit ratio respectively. In Table 3, the score of GCLOCK is 9, because there are four “>>”, one “>” and eight “=”. In Table 4, its score is also 9. So the final score of GCLOCK is 9.

We show the final score of every buffer replacement strategy in Table 5. This table shows the overall performance of all the algorithms on hit ratio. We can group the algorithms according to the final scores. For example, FBR and LRU strategies belong to the same group, and CAR, CLOCK and ARC stay together. It also shows that the FBR algorithm is the best on hit ratio and GCLOCK ranks third.

Table 3 T-test

表 3 T-test

T-test	1	2	3	4	5	6	7	8	9	10	11	12	13
GCLOCK(1)	=	=	=	>	=	=	=	>>	>>	=	=	>>	>>
CLOCK(2)	=	=	<<	>	=	=	=	>>	>>	=	=	>>	>>
LRU(3)	=	>>	=	>	>	>	>>	>>	>>	=	=	>>	>>
2Q-Simple(4)	<	<	<	=	<	<	<	=	>>	<	<	=	=
2Q-Full(5)	=	=	<	>	=	=	=	>>	>>	=	=	>>	>>
ARC(6)	=	=	<	>	=	=	>	>>	>>	=	=	>>	>>
CAR(7)	=	=	<<	>	=	<	=	>>	>>	=	=	>>	>>
CART(8)	<<	<<	<<	=	<<	<<	<<	=	>>	=	<	=	>>
LIRS(9)	<<	<<	<<	<<	<<	<<	<<	<<	=	<<	<<	<<	<<
LRFU(10)	=	=	=	>	=	=	=	=	>>	=	<<	>>	>>
FBR(11)	=	=	=	>	=	=	=	>	>>	>>	=	>>	>>
LRU-2(12)	<<	<<	<<	=	<<	<<	<<	=	>>	<<	<<	=	=
CLOCK-Pro(13)	<<	<<	<<	=	<<	<<	<<	<<	<<	<<	<<	=	=

In the same way we can get the results about logical reads and execution time, as shown in Table 6 and Table 7 respectively.

5.7.3 Relationship

Logical reads, hit ratio and execution time are not independent. For each cache size, we sort the hit ratios and the logical reads of the thirteen algorithms in increasing order, and get two sorted lists. At a certain cache size, we can get a pair $<a,b>$ for each algorithm, where a and b ($a,b \in \{1,2,\cdots,13\}$) are the positions of the hit ratio and logical reads of the algorithm in the sorted lists, respec-

Table 4　Sign test

表 4　Sign test

T-test	1	2	3	4	5	6	7	8	9	10	11	12	13
GCLOCK(1)	=	>	=	=	=	=	>>	>>	>>	=	<<	>>	>>
CLOCK(2)	<	=	<<	=	=	=	=	=	>>	<	<<	=	>>
LRU(3)	=	>>	=	>>	>	=	>>	>>	>>	=	<	>>	>>
2Q-Simple(4)	=	=	<<	=	<<	=	=	=	>>	<	<<	=	=
2Q-Full(5)	=	=	<	>>	=	=	=	=	>>	=	<<	>	>>
ARC(6)	=	=	=	=	=	=	=	=	>>	=	<<	=	>>
CAR(7)	<<	=	<<	=	=	=	=	>	>>	<	<<	=	>>
CART(8)	<<	=	<<	=	=	=	<	=	>>	<<	<<	=	>>
LIRS(9)	<<	<<	<<	<<	<<	<<	<<	<<	=	<<	<<	<<	<<
LRFU(10)	=	>	=	>	=	=	>	>>	>>	=	<<	=	>>
FBR(11)	>>	>>	>	>>	>>	>>	>>	>>	>>	>>	=	>>	>>
LRU-2(12)	<<	=	<<	=	<	=	=	=	>>	=	<<	=	=
CLOCK-Pro(13)	<<	<<	<<	=	<<	<<	<<	<<	>>	<<	<<	=	=

Table 5　The final scores about hit ratio

表 5　命中率的最终得分

Hit ratio	LIRS	CLOCK-Pro	LRU-2	CART	2Q-Simple	CAR	CLOCK	ARC	2Q-Full	LRFU	GCLOCK	LRU	FBR
Final score	-24	-16	-9.5	-7	-5.5	2	2.5	5.5	6	6	9	14.5	16.5

Table 6　The final scores about logical reads

表 6　逻辑读的最终得分

Logical reads	LIRS	LRU-2	CLOCK-Pro	CART	CAR	CLOCK	2Q-Full	2Q-Simple	GCLOCK	ARC	LRFU	LRU	FBR
Final score	-24	-17.5	-13.5	-5	-4	-3	-2.5	-1.5	10	10	10.5	18	22.5

Table 7　The final scores about execution time

表 7　执行时间的最终得分

Execution time	GCLOCK	CLOCK	CAR	LRU	CART	2Q-Simple	2Q-Full	ARC	CLOCK-Pro	LIRS	LRU-2	LRFU	FBR
Final score	-22	-22	-14	-13	-5.5	-2	-0.5	-0.5	7.5	12.5	15.5	20	24

tively. Figure 19 plots the result pairs when the cache size is varied from 2 560 to 15 360 pages.

We can draw the following conclusion from the trend of the curves: overall, the number of logical reads increases with the increase of hit ratio. So the buffer replacement strategy with high hit ratio can be chosen to improve the system throughput. However, the number of logical reads is not in strict proportion to hit ratio. When two hit ratios are close to each other, it does not assure that the buffer replacement strategy with higher hit ratio will show higher performance.

We also conducted the experiment on a SMP machine, and increased the buffer cache from 89 600 pages to 102 400 pages. In this environment, it is CPU bound, not I/O bound as in the above experiment. Figure 20 shows the result. The curves fluctuate more greatly than the curves in Figure 19, which illustrates, because of the impact of execution time, it is more difficult to say that the number of logical reads is large if the hit ratio is high. So execution time is also an important factor that influences the system throughput, especially when CPU is used heavily.

5.8 Space Overhead

In this section, we compare the space overhead of different algorithms. In our implementation, there are mainly two types of space occupied by replacement strategies. The first is the hash table through which the buffer pages can be accessed quickly. The hash table exists for all of the algorithms, but the number of entries is different. Therefore the occupied space is not the same. In addition, most of the algorithms need to keep control information for each buffer page that exists in the cache and for some pages that have ever been in the cache. Table 8 shows the space overhead of various buffer replacement algorithms.

In Table 8, m is the size of the entry in hash table, which is 32 Byte in our implementation, and c is the size of control block of buffer page which varies from zero Byte to 60 Byte for different replacement algorithms. n is the buffer cache size. The data in the brackets show the space occupied by the algorithms when n is 70 000. Except GCLOCK and CLOCK, all the other strategies need to store the control blocks. These strategies are divided into three types according to the number of control

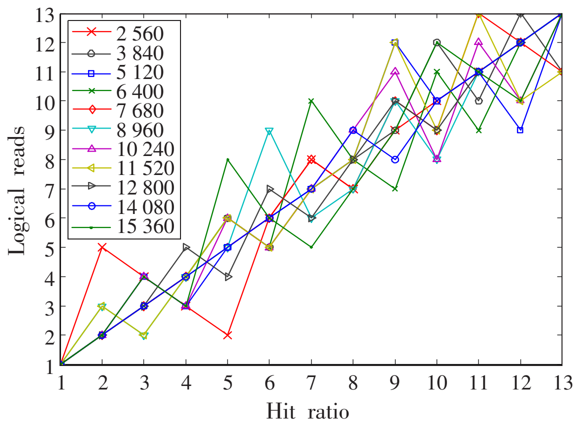


Fig.19 Relationship (I/O bound)

图 19 I/O 局限时的关系

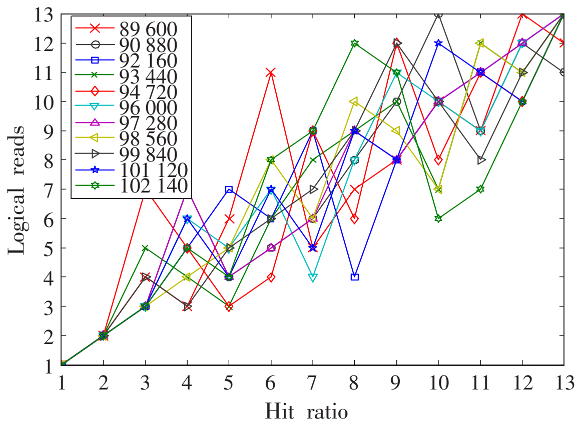


Fig.20 Relationship (CPU bound)

图 20 CPU 局限时的关系

Table 8 Space overhead

表 8 空间使用

	Space/Byte		Space/Byte
GCLOCK/CLOCK	$m*n$ (2 766 904+12)	CART	$(m+c)*2*n$ (5 532 728+5 040 072)
LRU	$m*n+c*n$ (2 766 904+560 008)	LIRS	$(m+c)*1.5*n$ (3 887 160+5 040 052)
2Q-Simple	$m*n+c*n$ (2 766 904+2 520 036)	LRFU	$m*n+c*n$ (2 766 904+3 920 040)
2Q-Full	$(m+c)*1.5*n$ (3 887 160+3 780 048)	FBR	$m*n+c*n$ (2 766 904+3 360 080)
ARC	$(m+c)*2*n$ (5 532 728+5 040 060)	LRU-2	$(m+c)*1.5*n$ (3 887 160+6 300 052)
CAR	$(m+c)*2*n$ (5 532 728+5 040 060)	CLOCK-Pro	$(m+c)*2*n$ (5 532 728+5 600 040)

blocks and entries in hash table. For LRU, 2Q-Simple, LRFU and FBR algorithms, the number of control blocks is equal to the cache size, because they need no extra blocks to keep history information of the pages that have been replaced. 2Q-Full, LIRS and LRU-2 use half of the cache to remember recently evicted buffer pages, while the others utilize more space to maintain history information so that the number of control blocks is two times larger than the cache size.

6 Conclusion

In this paper, we proposed a performance evaluation methodology and did a thorough experimental study of thirteen buffer replacement algorithms. In this methodology, we employed five performance metrics and two significance tests. Five workloads of various access patterns were developed and TPC-H and TPC-C benchmarks were used to compare different algorithms in a real database environment. We hope our work will help researchers and developers of database systems to design, choose and evaluate their buffer replacement strategies. In addition, the following conclusions can be drawn from our experimental study:

(1)Significance analysis can be applied to the evaluation of buffer replacement strategies, and jointly used for cross-strategy comparison.

(2)The performance of the buffer replacement strategies depends heavily on the workload characteristics. For example:

①LRU and CLOCK are not scan-resistant. The other algorithms can do better than LRU and CLOCK for sequential references.

②Except 2Q-Simple, ARC, CAR and CART, the other strategies can deal with the correlated reference pattern well.

③For temporally-clustered references, LRU and LRFU algorithms show better performance than the other algorithms.

④LRU-2, CLOCK-Pro, LRFU and LIRS perform well on probabilistic pattern workloads.

(3)Hit ratio is a critical performance metric of buffer replacement strategies, but it alone can not determine the overall system performance. Execution time of replacement algorithms also affects the system throughput, especially when the system is CPU bound.

(4)The same increase in hit ratio when hit ratio is high will result in a larger increase in system throughput than that when hit ratio is low. So even hit ratio is already high, it is still helpful to improve it.

References:

[1] Mattson R L, Gecsei J, Slutz D R, et al. Evaluation techniques for storage hierarchies[J]. IBM Systems Journal, 1970,9(2):78-117.

[2] Corbato F J. A paging experiment with the multics system, MIT Project MAC Report MAC-M-384[R]. May 1968.

[3] Nicola V F, Dan A, Dias D M. Analysis of the generalized clock buffer replacement scheme for database transaction processing[C]//Proceedings of 1992 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, 1992:35-46.

[4] Robinson J T, Devarakonda M V. Data cache management using frequency-based replacement[C]//Proceedings of 1990 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, 1990:134-142.

[5] O'Neil E J, O'Neil P E, Weikum G. The LRU-K page replacement algorithm for database disk buffering[C]//Proceedings of ACM SIGMOD Conference, 1993:297-306.

[6] Johnson T, Shasha D. 2Q: a low overhead high performance buffer management replacement algorithm[C]//Proceedings of VLDB Conference, 1994:297-306.

[7] Lee D, Choi J, Kim J H, et al. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies[J]. IEEE Transactions on Computers, 2001,50(12):1352-1361.

[8] Megiddo N, Modha D S. ARC: a self-tuning, low overhead replacement cache[C]//Proceedings of the 2nd USENIX Conference on File and Storage Technologies, 2003: 115-130.

[9] Jiang S, Zhang X. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance[C]//Proceedings of 2002 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, 2002:31-42.

[10] Bansal S, Modha D. CAR: clock with adaptive replacement[C]//Proceedings of the 3rd USENIX Symposium on File and Storage Technologies, 2004:187-200.

[11] Jiang S, Chen F, Zhang X. CLOCK-Pro: an effective improvement of the CLOCK replacement[C]//Proceedings of 2005 USENIX Annual Technical Conference, 2005: 323-336.

[12] Chou H T, Dewitt D J. An evaluation of buffer management strategies for relational database systems[C]//Proceedings of VLDB Conference, 1985:174-188.

[13] KingbaseES. http://www.kingbase.com.cn.

[14] Transaction Processing Performance Council. TPC Benchmark H (Decision Support) Standard Specification, Revision 2.0.0[S].

[15] Transaction Processing Performance Council. TPC Benchmark C Standard Specification, Revision 1.0[S]. 1992.

[16] Choi J, Noh S H, Min S L, et al. Towards application/file-level characterization of block references: a case for fine-grained buffer management[C]//Proceedings of 2000 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, 2000:286-295.

[17] Effelsberg W, Loomis M E S. Logical, internal, and physical reference behavior in CODASYL database systems[J]. IEEE Transactions on Computers, 1995,44(4): 546-560.

[18] Glass G, Cao P. Adaptive page replacement based on memory reference behavior[C]//Proceedings of 1997 ACM

SIGMETRICS Conference on Measuring and Modeling of Computer Systems，1997:115-126.

[19] Smaragdakis Y，Kaplan S，Wilson P. EELRU：simple and effective adaptive page replacement[C]//Proceedings of 1999 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems，1999:122-133.

[20] Choi J，Noh S H，Min S L，et al. An implementation study of a detection-based adaptive block replacement scheme[C]//Proceedings of 1999 Annual USENIX Technical Conference，1999:239-252.

[21] Butt A R，Gniady C，Hu Y C. The performance impact of kernel prefetching on buffer cache replacement algorithms[C]//Proceedings of 2005 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems，2005:157-168.

[22] Denning P J. The working set model for program behavior[J]. Communications of the ACM，1968,11(5):323-333.

[23] Coffman E G. Operating systems theory[M]. [S.l.]：Prentice-Hall，1973.

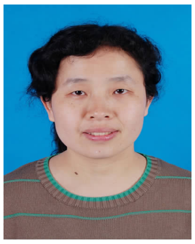
[24] Aho A V，Denning P J，Ullman J D. Principles of optimal page replacement[J]. Journal of the ACM，1971,18(1):80-93.

LUAN Hua was born in 1980. She is a Ph.D. candidate at Renmin University of China. She is a member of CCF. Her research interests include databases and data warehouses，etc.

栾华(1980-),女,山东龙口人,中国人民大学博士生,CCF 会员,主要研究领域为数据库、数据仓库等。

DU Xiaoyong was born in 1963. He received the Ph.D. degree from Nagoya Institute of Technology，Japan in 1997. He is a professor and doctoral supervisor at Renmin University of China. He is a member of CCF. His research interests include high performance databases，intelligent information re trieval and semantic web，etc.

杜小勇(1963-),男,浙江开化人,1997 年于日本名古屋工业大学获得博士学位,目前是中国人民大学教授、博士生导师,CCF 高级会员,主要研究领域为高性能数据库、智能信息检索以及语义网等。



FENG Yu was born in 1973. She received the Ph.D. degree in computer science from Institute of Computing Technology，Chinese Academy of Sciences in 2002. Her research interests include high performance databases，data warehouses and data mining，etc.

冯玉(1973-),女,河南正阳人,2002 年于中国科学院计算技术研究所获得博士学位,主要研究领域为高性能数据库、数据仓库以及数据挖掘等。

WANG Shan was born in 1944. She received the M.S. degree in computer science from Renmin University of China in 1981. She is a professor and doctoral supervisor at Renmin University of China. Her research interests include high performance databases，knowledge systems，data warehouses and grid data management，etc.

王珊(1944-),女,江苏无锡人,1981 年于中国人民大学获得硕士学位,中国人民大学教授、博士生导师,主要研究领域为高性能数据库、知识工程、数据仓库以及网格数据管理等。