A Database Disk Buffer Management Algorithm based on Prefetching

H. Seok Jeon* Sam H. Noh

Department of Computer Engineering
Hong-Ik University
Mapo-Gu Sangsoo Dong 72-1 Seoul, Korea 121-791
tel) +82-2-320-1470 fax) +82-2-320-1105
email: {hsjeon, noh}@cs.hongik.ac.kr
(* Also, with Thinkware Systems Corp.
Poi-Dong 196-2, Kangnam-Gu, Seoul, Korea 135-260
tel) +82-2-571-9160 fax) +82-2-571-0515)

Abstract

This paper proposes a prefetch-based disk buffer management algorithm, which we call W^2R (Weighing/Waiting Room). Instead of using elaborate prefetching schemes to decide which block to prefetch and when, we simply follow the LRU-OBL (One Block Lookahead) approach and prefetch the logical next block along with the block that is being referenced. The basic difference is that the W^2R algorithm logically partitions the buffer into two rooms, namely the Weighing Room and the Waiting Room. The referenced, hence fetched block is placed in the Weighing Room, while the prefetched logical next block is placed in the Waiting Room. By so doing, we alleviate some inherent deficiencies of blindly prefetching the logical next block of a referenced block. Specifically, a prefetched block that is never used may replace a possibly valuable block and a prefetched block, though referenced in the future, may replace a block that is used earlier than itself. Using the DB2 and OLTP traces, we show through trace driven simulation that for the workloads and the environments considered the W^2R algorithm improves the hit rate by a maximum of 23.19 percentage points compared to the 2Q algorithm and a maximum of 9.27 percentage points compared to the LRU-OBL algorithm.

1 Introduction

Many algorithms for improving the performance of database disk buffer management have been proposed. A large group of work can be categorized into those that tune the buffer management algorithm based on information from the query optimizer regarding the plan of the query. Examples of these are the Hot Set model [SS86] and the DBMIN algorithm [CD85]. Many extensions and variants of these algorithms have also been reported [ABGM90, COL92, FNS91, FNS95, JCL90, YC91].

Another category of works rely on the judicious replacement of buffers as it has been pointed out that for multitasking environments, information from query optimizer may not be inappropriate for performance enhancement. [OOW93]. Algorithms such as the LRU, the LRU-K [OOW93], the 2Q [JS94], and the FBR [RD90] are examples. Specifically, while the LRU algorithm bases its replacement decision on a block's most recent reference, the LRU-K algorithm considers multiple past references, specifically K past references, in making its decision. This allows the algorithm to observe the importance of the block over a longer time span, instead of deciding the block's importance over a single reference. The 2Q algorithm considers a block "worthwhile to retain" in the cache only after it is referenced more than once. Upon the first reference, the block is put in a "probationary" part of the cache referred to as the A1 queue. Only after a second reference, is the block inserted into the "main" part of the cache referred to as the Am queue, hence the name 2Q. Basically, a block is upgraded to the main queue only after its "hotness" is proven.

While the LRU-K and 2Q algorithms base their replacement decision basically on the recency of block references, the FBR algorithm makes its decision on the frequency of block references. In so doing, Robinson and Devarakonda observe that the filtering of correlated references is an important factor in the efficiency of the algorithm.

The LRU-K, 2Q, and the FBR algorithms do not use any form of prefetching. Recent developments in buffer management replacement algorithms has lead to the investigation of incorporating prefetching to the replacement algorithms. It has been shown that for some workloads incorporating prefetching can result in considerable improvement in the performance of buffer management [Smi78, Smi85].

Research in incorporating prefetching have mainly been conducted in the systems arena, and again be categorized into three groups depending on when and which block to prefetch. The first group of algorithms maintain a history of past behavior of the applications [LD97, CKV93, KE93, PZ91]. This speculative approach, however, may result in performance degradation due to inaccurate prefetching and history maintenance overhead.

The second category of algorithms obtain hints from applications themselves [CFKL95, CF96, PGG$^+$95, TPG97] prior to execution. This approach, though promising, has not yet proven itself in general workload environments. Though it has been shown that hints may be obtainable for specific applications within a specific environment [MDK96], how this will be applicable to buffer management in general is still an open question.

The final approach does not require any information neither from the application nor from observations of past behavior. The LRU-OBL (One Block Lookahead) algorithm [Smi78, Smi85] (and its variant of prefetching multiple blocks at once) is the only algorithm known to date using this approach. This algorithm simply prefetches the logical next block of the currently referenced block if it is not resident in cache. It has been shown that through this simple algorithm, improvements of up to 80% in the hit rate was possible for some workloads [Smi85]. Another practical benefit of this algorithm is that it is simple to implement, and prefetching does not require any additional overhead from the user nor the system.

The algorithm that we propose in this paper, referred to as the W^2R (Weighing/Waiting Room) algorithm, extends the LRU-OBL algorithm and borrows from the 2Q algorithm. It prefetches blocks exactly like the LRU-OBL algorithm requiring no overhead in determining which block to prefetch and when. However, the management of these blocks is different. The W^2R algorithm partitions the buffer into two rooms where one is used to manage referenced blocks and the other used to manage prefetched blocks. We show that by such partitioning some deficiencies inherent to the LRU-OBL algorithm are alleviated resulting in enhanced performance. Through trace driven simulation using the DB2 and OLTP database traces, we show that the W^2R algorithm improves the hit rate of the buffer by a maximum of 23.19 percentage points compared to the 2Q algorithm and 9.27 percentage points compared to the LRU-OBL algorithm for the environments that we considered.

Another advantage of the W^2R algorithm compared to the LRU-OBL algorithm is its modularity. Whereas the LRU-OBL algorithm inherently intertwines the replacement and prefetching concepts into one algorithm, the structure of the proposed algorithm clearly divides these two concepts making it simple to incorporate new buffer management replacement algorithms, possibly improving its performance even more.

The rest of the paper is organized as follows. Section 2 discusses in more detail the motivation behind the W^2R algorithm and the algorithm itself. In Section 3, the experiments and the results of these experiments are discussed. The modularity of the W^2R algorithm is discussed in Section 4. Finally, Section 5 concludes with a summary and directions for further research.

2 The W^2R (Weighing/Waiting Room) Algorithm

Recall the actions of the LRU-OBL algorithm. Management of the buffer itself is done using the LRU algorithm while in prefetching blocks a simple approach is used. That is, it prefetches the logical next block of the currently referenced block. This basically requires no overhead to determine the block to prefetch and when to prefetch.

However, because it does a simple-minded prefetch, the prefetched block may actually do harm instead of good to the performance of buffer management. There are two sources from which harm may be induced. First, consider a block in the disk that is referenced for the first time and brought into the buffer replacing the LRU block. This block is placed at the most recently used (MRU) position. More significantly, the logical next block, if not currently in the buffer, is also brought into the buffer replacing the current LRU block in the buffer. Not only that, this prefetched block is also placed at the MRU position. The problem with this action is that the prefetched block may never be referenced in the future, and if this is the case, the prefetched block has simply removed a potentially valuable block and taken its place, resulting in wasted buffer space.

The numbers in Table 1 support this conjecture. Taking the OLTP and DB2 traces (which we describe in more detail later) we counted the number of blocks prefetched using the LRU-OBL algorithm for various buffer sizes. These numbers are shown in the second row. The third row shows the percentage of the prefetchec blocks that are never referenced. Note that the percentage of blocks prefetched but never referenced are roughly 46% to over 66% of the prefetched blocks.

The second source of possible harm due to blindly prefetching the logical next block is that even if the prefetched block is referenced at some future time, by prefetching the block right along with the referenced block, it may be replacing a block too early. This may result in replacing a block that is used earlier than the prefetched block. This is a typical situation of prefetching at the wrong time [CFKL95].

The W^2R algorithm tackles these sources of harm by partitioning the buffer into two rooms, that is, the Weighing Room and the Waiting Room. To describe the W^2R algorithm, and also the management of the buffer in general, we use the weight analogy. In general, the block to be replaced by the incoming block is the block that is considered to be the least likely to be re-referenced. This likelihood can be represented as a weight. Each block is given a weight, and the heavier block is considered more likely to be re-referenced. Then, in general, the lightest block is replaced by the incoming block as it is considered to be the least likely to be referenced again.

Table 1: The number of prefetched blocks using the LRU-OBL algorithm and the percentage of those that are never referenced for the DB2 and OLTP traces.

Buffer Size	500	1000	2000	3000
Number of Blocks Prefetched	233,859	200,484	172,773	156,418
Blocks Prefetched but Never Referenced (%)	62.13%	56.86%	51.15%	46.81%

(a) DB2 trace

Buffer Size	500	1000	2000	3000
Number of Blocks Prefetched	748,896	688,834	610,019	560,201
Blocks Prefetched but Never Referenced (%)	66.26%	66.04%	63.94%	61.80%

(b) OLTP trace

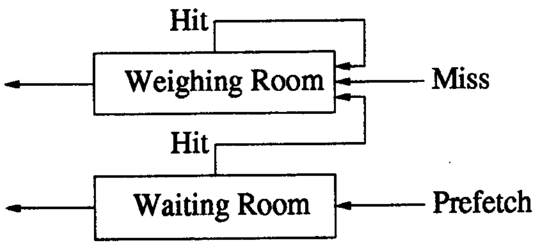


Figure 1: Structure of the W^2R algorithm.

The actual weight of each block in the buffer is determined by the algorithm that is being used. For example, the LRU algorithm assumes that the MRU block is the most likely to be re-referenced, hence is considered the heaviest. Conversely, the LRU block is considered the lightest. In its implementation, the weight of each block would be represented by the position in the list of blocks.

The Weighing Room, in the W^2R algorithm, is where the weights of the blocks are contested, and rank is formed among the blocks. Only blocks that have been referenced have weights associated with them. In buffer management algorithms such as the LRU or 2Q, the whole buffer is simply the Weighing Room as only blocks that have been referenced are brought into the buffer.

The Waiting Room is where the blocks remain and wait until they obtain permission to be weighed with the other blocks. This permission is obtained when and only when the block has actually been referenced. Until this time, the prefetched blocks are weightless. They obtain weight only when referenced.

Figure 1 shows the logical structure of the W^2R algorithm.

By partitioning the buffer into a Weighing Room and a Waiting Room, the first source of harm mentioned above is alleviated because blocks that are not referenced after being prefetched are not promoted to the Weighing Room. Hence, a prefetched block that is never referenced cannot replace a block that has some weight (that is, in the Weighing Room). That is to say, the wrong block may be prefetched into the buffer, but will not replace a block that has proven its worth by having been referenced.

The second source of harm is completely resolved as the prefetched block enters the Weighing Room only after it is referenced. Hence, the prefetched block does not replace a block that is referenced prior to the prefetched block. The block being promoted to the Weighing Room is promoted right when it is needed, and never before.

Once a block is promoted to the Weighing Room, how much weight it will have is a matter of the algorithm used in the Weighing Room, that is, what kind of scale is used to weigh the blocks. The algorithm for the W^2R algorithm with the LRU algorithm used as the scale in the Weighing Room is shown Figure 2. Upon a request for block i, it is checked if block i is in either the Weighing or Waiting Rooms. If block i is not in the buffer, it is fetched from the disk and moved directly to the Weighing Room (line 21 of Algorithm W^2R). If it is, it is moved to the head of the Weighing Room, that is, it becomes the heaviest block in the room (lines 5 and 14 of Algorithm W^2R). Note that the actions of lines 5, 14, and 21 (marked by '*') are specific to the LRU algorithm being used in the Weighing Room.

As for the logical next block $i+1$, action is taken only when it is not found in either of the two rooms. When this happens, the block is prefetched from disk and put in the Waiting Room. The blocks in the Waiting Room are organized in a FIFO manner. When a block in this queue is accessed, it is moved to the Weighing Room. Otherwise, it is simply pushed off the queue as new blocks come in.

A new parameter is introduced in the W^2R algorithm. Given a fixed buffer size, one now has to decide how to partition it into two rooms. By introducing the Waiting Room, we are in effect, reducing the size of the Weighing Room compared to conventional buffer management algorithms. A judicious selection of the room size is necessary for efficient management of the buffer. This matter is discussed in the next section along with the experiments.

```
Algorithm W²R :
input : requested block number i
output : requested block
if i is in Weighing Room then
*  put i at head of Weighing Room
/ * Weighing Room managed as an LRU queue * /
   if i+1 is not in either Room then
      prefetch i+1 from disk and
      put at head of Waiting Room
/ * Waiting Room is managed as a FIFO queue * /
   end if
else if i is in Waiting Room then
   remove i from Waiting Room
*  put i at head of Weighing Room
   if i+1 is not in either Room then
      prefetch i+1 from disk and
      put at head of Waiting Room
   end if
else
   fetch i from disk
*  put i at head of Weighing Room
   if i+1 is not in either Room then
      prefetch i+1 from disk and
      put at head of Waiting Room
   end if
end if
```

Figure 2: Algorithm for the W^2R algorithm using the LRU algorithm in the Weighing Room.

3 Experimental Results

In this section, we discuss the experimental evaluation of the W^2R algorithm. A description of the simulator and the traces that were used is given in the next subsection. In the subsequent subsection, we report and discuss the results of these experiments.

3.1 The Simulator and Traces

The simulator developed to evaluate the algorithm is programmed in C++. The basic component in this simulator is the buffer cache module which takes the traces as input. The buffer cache module checks if the block number is in the buffer. If it is a hit, appropriate action, which is dependent on the algorithm used, is taken. Otherwise, a block fetch request action to the disk is emulated. The block size, size of the buffer, and the algorithm used for managing the buffer are controllable parameters. Figure 3 shows the composition of the simulator.

The traces used to drive the simulator are the DB2 and OLTP traces [JS94]. These traces are those identical to the traces used in the papers by Johnson and Shasha [JS94] and by O'Neil and others [OOW93]. The DB2 trace is obtained from running a DB2 commercial application and contains 500,000 block requests to 75,514 distinct blocks. Obtained from the On-Line Transaction Processing System, the OLTP trace contains records

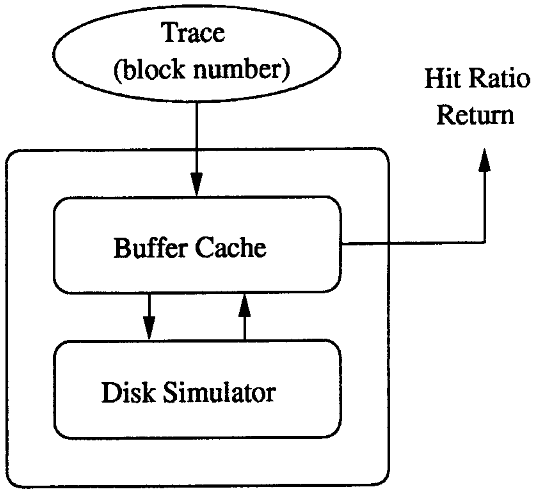


Figure 3: Components of the simulator.

of block requests to a CODASYL database for a window of one hour. It contains a total of 914,145 requests to 186,880 distinct blocks.

3.2 The Results

Table 2 shows the hit rates of the W^2R algorithm compared with the LRU, 2Q, and the LRU-OBL algorithms. For the W^2R algorithm, the best hit rate that was obtained in the experiments that we conducted are reported.

Notice that the hit rates of the 2Q algorithm obtained through our simulator are slightly worse than those reported by Johnson and Shasha [JS94]. In their paper, Johnson and Shasha point out that the size of the A1 queue could be critical on the performance of the algorithm. To minimize the effect of this queue, the A1 queue was modified to hold pointers to blocks instead of the blocks themselves. This allowed the queue to hold much more information regarding the access of the blocks. However, when the block pointed to in the A1 queue is moved to the Am queue, an extra disk access is inevitable, possible having other effects on the performance. To make a fair comparison among the algorithms, we implemented the 2Q algorithm with the A1 queue holding actual blocks instead of pointers to blocks. Hence, we see that the 2Q algorithm performs slightly worse than was originally reported.

To obtain the hit rates for the W^2R algorithm the experiments were conducted by varying the Waiting Room sizes. For the given buffer size, the Waiting Room size was set to a fixed size starting from 1 increasing in increments of 1. For each fixed size the hit rate was obtained through the simulator. The results show that the W^2R algorithm shows substantial hit rate improvements compared to the 2Q algorithm, the increase being at least 14 percentage points or more. Compared to the LRU-OBL algorithm, the W^2R algorithm still performs considerably better, with a minimum increase in the hit rate of

1.58 percentage points to a maximum of 4.43 percentage points.

Along with the best hit rates, Table 2 also shows the range of the size of the Waiting Room at which the best hit rate occurs. These are shown in parentheses on the W^2R hit rate row. Note that the best hit rate occurs when the Waiting Room size is relatively small ranging in the 20's to 30's for both traces, and that these value ranges overlap considerably for each of the traces.

The reason behind these results can be explained through Figures 4(a) and (b). These figures show the position at which a block in the Waiting Room is referenced and moved to the Weighing Room for each of the traces. That is, consider a block that first enters the Waiting Room. It is put at position 0 when it enters and as time progresses and new blocks come in, this block moves up in position until it is referenced and moved to the Weighing Room, or simply falls off the FIFO queue. The y-axis in these figures represent the position when either the move to the Weighing Room or the falling-off occurs for the blocks of each trace. The x-axis is simply the time sequence. These figures were obtained when using the W^2R algorithm for a total buffer size of 3000 with the Waiting Room size fixed to 2700.

In these figures we notice a thick dark shade along the 0 location throughout the time sequence. Closer observation shows that the position at which most of the blocks are referenced is below the 30th position for the DB2 trace and below the 50th position for the OLTP trace. The rest of the blocks are referenced at random positions without any regularity. This implies that a small Waiting Room of roughly 20 to 50 blocks in size, is enough to reap a majority of the benefit provided by the Waiting Room. Keeping a Waiting Room larger than this brings about only marginal benefits and can even result in poor performance as a result of reducing the Weighing Room size. For the traces and buffer sizes considered Waiting Room sizes starting from roughly 10 (that is, close to 1% of the total buffer size) to about 30% of the total buffer size resulted in better performance than the LRU-OBL algorithm.

As the reference characteristics of the DB2 and OLTP traces are typical of database workloads, Waiting Room sizes in the 20-50 range is conjectured to be sufficient to reap the benefits of the W^2R algorithm for efficient database buffer management.

4 Modularity of the W^2R Algorithm

The W^2R algorithm has another advantage in that it is modular. That is, any known buffer management replacement algorithm may be used for the Weighing Room. As stated previously, the hit rate of the Weighing Room is dependent on the workload and the algorithm used. Recent results have shown that the LRU algorithm may not be the best block replacement algorithm [RD90, OOW93, JS94]. As new algorithms that show better performance are developed and implemented, these algorithms may be incorporated into the W^2R algorithm.

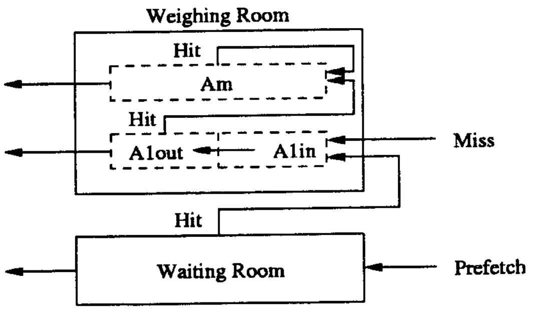


Figure 5: The W^2R algorithm using the 2Q algorithm for the Weighing Room.

To incorporate a different algorithm in the Weighing Room, the only modifications that need to be made to Algorithm W^2R of Figure 2 are to lines 5, 14, and 21 (as marked by '*'). Line 5 needs to be replaced with the actions taken when a block within the buffer is referenced. For instance, in the 2Q algorithm, when a block is hit within the buffer the block is checked to see if it is in the Am queue or the A1 queue, and thereafter, appropriate action is taken. Hence, to use the 2Q algorithm in the Weighing Room, line 5 would be replaced with the following lines of code.

```
if i is in Am queue then
    put i at head of Am queue
else if i is in A1out queue then
    remove i from A1out queue
    put i at head of Am queue
else
    /* Do Nothing! */
```

Lines 14 and 21, on the other hand, need only be replaced by the actions taken when a block is fetched from disk and moved to the buffer. For the 2Q algorithm, it is simply

put i *at head of A1in queue.*

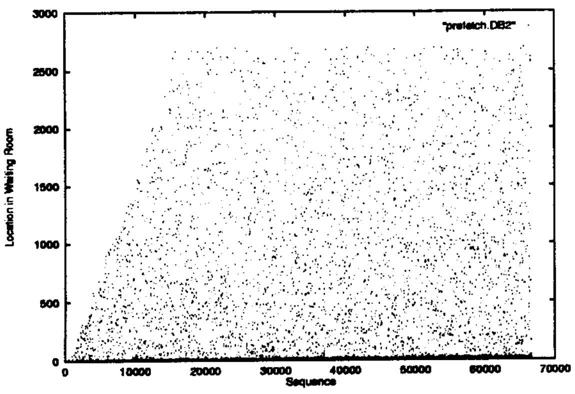
Note that for the LRU algorithm, the action taken when a block is hit within the buffer and when a block is fetched from disk and moved to the buffer is the same. Hence, lines 5, 14, and 21 are identical in Algorithm W^2R.

As the 2Q algorithm, to date, is the algorithm reporting the best hit rate we chose to incorporate this algorithm into the W^2R algorithm. Figure 5 shows the structure of the W^2R algorithm that incorporates the 2Q algorithm in the Weighing Room, which we denote as W^2R-2Q. The results of the experiments using the W^2R-2Q algorithm, compared to those reported in the previous section are shown in Figure 6. The W^2R algorithm with the LRU algorithm used in the Weighing Room is now denoted as W^2R-LRU.

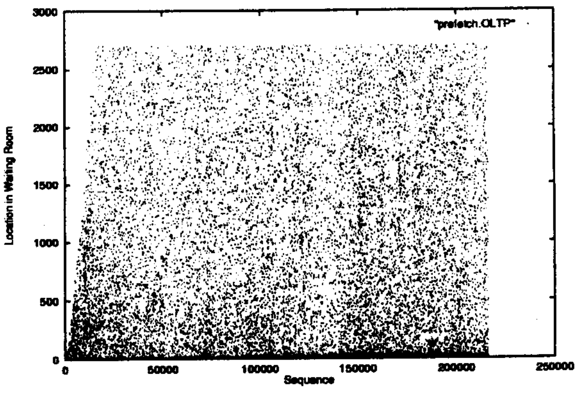
For the LRU-OBL and the W^2R-LRU algorithms, the points in the graph are those listed in Table 2. For the W^2R-2Q algorithm, we used approximately the

Table 2: The hit rates of various algorithms for the traces used. The last two rows show the difference in hit rate between the W^2R algorithm and the 2Q and LRU-OBL algorithms.

	DB2			OLTP		
Buffer Size	1000	2000	3000	1000	2000	3000
LRU	65.44	70.38	72.95	32.83	42.47	47.10
2Q	66.92	71.93	74.29	37.70	43.95	47.81
LRU-OBL	79.68	84.33	87.29	51.62	60.43	65.48
W^2R (Waiting Room Size)	81.87 (23-25)	86.40 (20-28)	88.88 (22-28)	56.05 (35-36)	64.64 (32-35)	68.88 (32-45)
Difference: W^2R– 2Q	14.95	14.47	14.59	18.35	20.69	21.07
Difference: W^2R– LRU-OBL	2.19	2.07	1.59	4.43	4.21	3.4



(a) DB2 trace



(b) OLTP trace

Figure 4: Position of the blocks in the Waiting Room when a reference to the block occurs.

same Waiting Room size as those of the W^2R-LRU algorithm, as listed in Table 2.

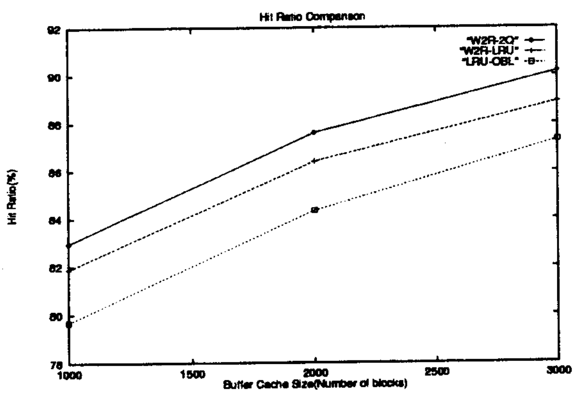
The increase in performance when using the W^2R-2Q algorithm is mainly due to the improvement in the hit rate at the Weighing Room. Because the blocks residing in the Weighing Room change when the algorithm is changed, the blocks in the Waiting Room will also be slightly affected. Overall though, we see that using an improved algorithm for the Weighing Room results in an improvement in the total hit rate. Specifically, the difference in the total hit rate between the W^2R-2Q algorithm and the LRU-OBL algorithm, as listed in Table 3, show that a maximum of 23.19 percentage point and a maximum of 9.27 percentage point improvements are made compared to the 2Q and the LRU-OBL algorithms, respectively.

5 Conclusion

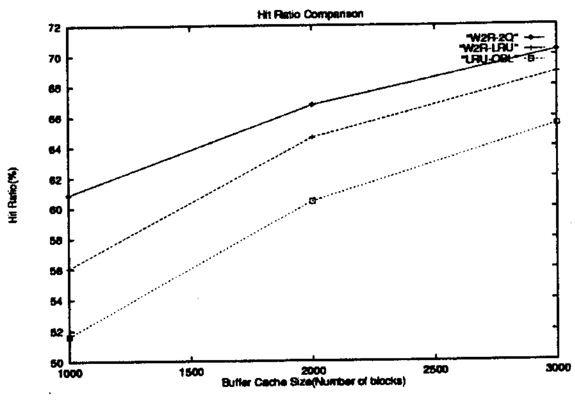
In this paper, we proposed a prefetch-based database disk buffer management algorithm, which we call the W^2R algorithm. Instead of using elaborate prefetching schemes to decide which block to prefetch and when, we simply follow the LRU-OBL approach and prefetch the logical next block along with the block that is being referenced. The difference being that the referenced block is placed in the Weighing Room, which holds referenced blocks only, while the prefetched logical next block is placed in the Waiting Room, which holds blocks prefetched but never referenced. By so doing, we alleviate some inherent deficiencies of blindly prefetching the logical next block of a referenced block. Specifically, a prefetched block that is never used may replace a possibly valuable block and a prefetched block, though referenced in the future, may replace a block that is used earlier than itself. Using the DB2 and OLTP traces, we show through trace driven simulation that for the workloads and the environments considered the W^2R algorithm improves the hit rate by a maximum of 23.19 percentage points compared to the 2Q algorithm and a maximum of 9.27 percentage points compared to the

Table 3: Difference in the hit rate between the W^2R-2Q and the 2Q and LRU-OBL algorithms.

	2Q			LRU-OBL		
Buffer Size	1000	2000	3000	1000	2000	3000
DB2 trace	16.03	15.67	15.85	3.37	3.27	2.85
OLTP trace	23.19	22.83	22.51	9.27	6.35	4.84



(a) DB2 trace



(b) OLTP trace

Figure 6: Hit rate using the 2Q algorithm in the Weighing Room for the W^2R algorithm.

LRU-OBL algorithm

To make this algorithm a resilient one, however, there still needs to be a simple standard guideline on how to partition the buffer into the Weighing Room and the Waiting Room. So far, the results presented here imply that for database workloads the Waiting Room should be made small, somewhere in the 30 to 50 block range. However, as only two traces were used further studies are still needed before a general conclusion can be drawn.

Also, we are currently conducting research in ways to make the algorithm adapt itself such that the performance is maximized. Even for the two traces, the Waiting Room size showing the maximum performance was slightly different. To resolve this problem a dynamic version of the W^2R algorithm that finds the best Waiting Room size on-line is being pursued.

References

[ABGM90] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems*, 15(3):359–384, 1990.

[CD85] H. T. Chou and D. DeWitt. An evaluation of buffer management strategies for relational database systems. In *Proceedings of the 11th ACM SIGMOD Conference*, pages 127–141, 1985.

[CF96] Pei Cao and Edward W. Felton. Implementation and Performance of Integrated Appplication-Controlled File Caching, Prefetching, and Disk Scheduling. *ACM Transactions on Computer Systems*, 14(4):311–343, November 1996.

[CFKL95] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. A study of integrated prefetching and caching strategies. In *Proceedings of the Joint International Conference on Measurement & Modeling of Computer Systems (SIGMETRICS '95 and Performance '95)*, pages 188–197, 1995.

[CKV93] K. Curewitz, P. Krishnan, and J.S. Vitter. Practical Prefetching via Data Compression. In *Proceedings of the 1993 ACM Conference on Management of Data (SIGMOD)*, pages 257–266, May 1993.

[COL92] C. Y. Chan, B. C. Ooi, and H. Lu. Extensible buffer management of indexes. In *Proceedings of the 18th VLDB Conference*, pages 444–454, 1992.

[FNS91] Christos Faloutsos, Raymond Ng, and Timos Sellis. Predictive load control for flexible buffer allocation. In *Proceedings of the 17th VLDB Conference*, pages 265–274, 1991.

[FNS95] Christos Faloutsos, Raymond Ng, and Timos Sellis. Flexible and Adaptable Buffer Management Techniques for Database Management Systems. *IEEE Transactions on Computers*, 44(4):546–560, 1995.

[JCL90] R. Juahari, M. Carey, and M. Linvy. Priority hints: An algorithm for priority-based buffer management. In *Proceedings of the 16th VLDB Conference*, 1990.

[JS94] Theodore Johnson and Dennis Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of the 20th VLDB Conference*, pages 439–450, 1994.

[KE93] David Kotz and Carla Schlatter Ellis. Practical Prefetching Techniques for Multiprocessor File Systems. *Journal of Distributed and Parallel Databases*, 1(1):33–51, January 1993.

[LD97] Hui Lei and Dan Duchamp. An Analytical Approach to File Prefetching. In *Proceedings of the USENIX 1997 Annual Technical Conference*, pages 275–288, January 1997.

[MDK96] Todd C. Mowry, Angela K. Demke, and Orran Krieger. Automatic Compiler-Inserted I/O Prefetching for Out-of-Core Applications. In *USENIX 2nd Symposium on Operating Systems Design and Implementation*, October 1996.

[OOW93] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 297–306, May 1993.

[PGG+95] R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. Informed Prefetching and Caching. In *Proceedings of the 15th Symposium on Operating System Principles*, pages 79–95, December 1995.

[PZ91] Mark Palmer and Stanley B. Zdonik. FIDO: A cache that learns to fetch. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 255–264, September 1991.

[RD90] J. T. Robinson and N. V. Devarakonda. Data Cache Management Using Frequency-Based Replacement. In *Proceedings of the 1990 ACM SIGMETRICS Conference*, pages 134–142, 1990.

[Smi78] Alan Jay Smith. Sequential program prefetching in memory heirarchies. *IEEE Computer*, 3(3):7–21, December 1978.

[Smi85] Alan Jay Smith. Disk cache-miss ratio analysis and design considerations. *ACM Transactions on Computer Systems*, 3(3):161–203, August 1985.

[SS86] G. M. Sacco and M. Schkolnick. Buffer management in relational database systems. *ACM Transactions on Database Systems*, 11(4):473–498, 1986.

[TPG97] Andrew Tomkins, R. Hugo Patterson, and Garth A. Gibson. Informed Multi-Process Prefetching and Caching. In *Proceedings of the International Conference on Measurement & Modeling of Computer Systems (SIGMETRICS '97)*, pages 100–114, June 1997.

[YC91] P. S. Yu and D. W. Cornell. Optimal buffer allocation in a multi-query environment. In *Proceedings of the 7th International Conference on Data Engineering*, pages 622–631, 1991.