A Buffer Cache Management for Locality with Prefetching

S.Subha
SITE, VIT, Vellore
TamilNadu, India

ABSTRACT
This paper proposes buffer cache architecture. The proposed model consists of three units – main buffer cache unit, pre-fetch unit and LRU Evict Unit. An algorithm is proposed to retain the evicted entry from main buffer cache unit in the LRU Evict Unit thereby giving it a second chance of access. On subsequent access in the LRU Evict Unit, the entry is promoted to the pre-fetch unit to accommodate more entries in the main buffer cache unit. On access in the pre-fetch unit, an entry is fetched into the main buffer cache. The LRU replacement policy is used in all the units. The proposed model is compared with buffer cache architecture with LRU replacement algorithm. The performance is comparable for sequential input where as 3% improvement in performance is seen for random input.

General Terms
Database buffer cache algorithm

Keywords
Buffer cache architecture, Least Recently used, Performance of database management system

1. INTRODUCTION
Buffer caches play an important role in database management systems. The size of the buffer cache plays a significant role in its performance. Pre-fetching plays a significant role in buffer cache performance. Various algorithms for buffer cache have been proposed. Some of them are LFU, LFU-k, 2Q, FBR, LRFU. The author in [12] proposes a buffer cache algorithm with pre-fetching that performs better than the waiting room - weighing room algorithm proposed in [3]. Pre-fetching has also been proven to be effective as discussed in [3, 4, 7 and 8]. File system speed has impact on buffer cache management [1]. The Least Recently Used replacement algorithm (LRU replacement) is often used in buffer cache. The major performance issue is that the LRU replacement entry is accessed after being evicted from the buffer cache. Pre-fetching helps in performance to a certain extent if the access pattern is sequential. This paper proposes an architecture which saves the LRU evicted entry inside the main memory in a separate unit called LRU Evict Unit. This reduces the number of misses based on the access pattern. The concept of access distance is used. This concept refers to the number of distinct entries that are accessed between consecutive accesses to a record. For example if the access pattern were 100, 200, 300, 100, 200, the access distance for 100 is 2 at time = 4 units as 200, 300 are accessed before it is accessed again.

An algorithm to place/replace entries in the proposed buffer cache model is developed. It is simulated for sequential and random input. The performance for sequential input is comparable with LRU replacement algorithm. A performance improvement of 3% is seen for random input.

The rest of the paper is organized as follows. Section 2 gives the Motivation, section 3 proposed model, section 4 Simulations, section 5 Conclusion and section 6 lists the references.

2. MOTIVATION
Consider a buffer cache of size four entries. Let the pre-fetch unit for this cache consist of three entries. Consider the following record access. Let the record id be 100, 200, 300, 400, 500, 600, 100. In the LRU replacement algorithm misses for 100, 200, 300, 400 happens with total number of misses = 4. The record id 500 replaces record id 100 by LRU replacement algorithm with number of misses = 5. Misses occur for 600 and 100 with total misses = 7. Consider the following architecture with associated algorithm; the system has a LRU Evict Unit in addition to the main buffer cache unit and Pre-fetch unit. The size of this unit is two entries.

1. Check if the address is in Main cache unit. If so, increment the number of hits and stop.
2. Check if the record is found in LRU evict. If found, the record is fetched for processing and placed in the Pre-fetch Unit. The LRU replacement policy is adapted in the Pre-fetch unit. Increments hit count and stop.
3. Check if the record is found in Pre-fetch Unit. If found, promote it to Main cache unit, increment hit count and stop.
4. Place the record in Main Cache unit. The policy for placement is as follows. If there is a vacant slot, place the record. If all slots are full, place the record in LRU slot. The LRU record is placed in the LRU Evict unit. The policy for placement in LRU Evict is as follows. If there is vacant slot, it is placed in it. Else, the LRU of LRU Evict is replaced. Pre-fetch the next block based on OBL into the pre-fetch unit. Increment the number of misses and stop.

According to the above algorithm, misses occur for 100, 200, 300, 400 with total number of misses = 4. For 500, the record with record id 100 is replaced. The record with record id 100 is placed in the LRU Evict unit. Record with id 600 replaces record with id 200 which is placed in the LRU Evict unit. The total number of misses is = 6. For record with record id 100, a hit is observed in LRU Evict Unit. It is transferred into the Pre-fetch Unit. There is an increase in number of hits in this algorithm. This is the motivation behind this paper.

3. PROPOSED ARCHITECTURE

The proposed system is shown in Figure 1. The system consists of three parts.

1. Main cache unit
2. LRU Evict
3. Pre-fetch Unit

The record is searched for in the Main Cache Unit depicted by (1). If not found, it is searched in Pre-fetch Unit and fetched to Main Cache Unit depicted by (2). The victim of replacement is placed in LRU Evict Cache depicted by (3). If not found in Pre-fetch Unit, the record is searched in the LRU Evict Unit and placed in Pre-fetch Unit depicted by (4). The Pre-fetch Unit is assumed to have size to place n number of records n>1. The size of the LRU and Pre-fetch unit is assumed to be less than the Main cache unit.

The algorithm for the proposed architecture is given next.

Algorithm LRU_Retain_Buffer Cache Algorithm: Given an address a this algorithm returns TRUE if it is found in the cache unit else returns FALSE

```
1. Check if the address is in Main cache unit. If so, write TRUE and stop.
2. Check if the record is found in LRU evict. If found, the record is fetched for processing and placed in the Pre-fetch Unit. Write TRUE and stop.
3. Check if the record is found in Pre-fetch Unit. If found, promote it to Main cache unit, write TRUE and stop.
4. Place the record in Main Cache unit. The policy for placement is as follows. If there is a vacant slot, place the record. If all slots are full, place the record in LRU slot. The LRU record is placed in the LRU Evict unit. The policy for placement in LRU Evict is as follows. If there is vacant slot, it is placed in it. Else, the LRU of LRU Evict is replaced. Pre-fetch the next block based on OBL into the Pre-fetch unit. Write FALSE and stop.
```

The time complexity of the algorithm is O (n) for n records. The algorithm gives a second chance for access to the LRU block that is evicted from the Main Cache unit by placing it in LRU Evict unit. A record which is accessed at time x, is placed in LRU Evict on becoming LRU candidate and stays in the LRU Evict Unit till it is accessed next or becomes LRU record in it. The size of the LRU Evict Unit and Pre-fetch Unit play an important role in the performance of the algorithm. For the best case, if the size of the LRU Evict is equal to the total number of distinct addresses in the application, only the first access to the record will incur a miss.

4. SIMULATIONS

The proposed algorithm is simulated for sequential and random input. C routines were written to generate the record ids for sequential and random access. The performance of the proposed model and the LRU replacement algorithm are comparable for sequential input with 99% hits. The performance of the algorithm is shown in Table 1 and Table 2 for random input of size 100000 entries. For a given buffer cache size, the sizes of the LRU Evict Unit and Pre-fetch Unit are varied. It is observed that for a given buffer cache size, the misses are minimum for pre-fetch unit of 64 entries for LRU Evict Unit sizes of 64, 32, 16, 8, 4, 2. The pre-fetch unit size was varied from 64 to 2 in powers of two for this study. In these tables c stands for the LRU Evict Unit size and p stands for Pre-fetch Unit size in number of entries. If the buffer cache size is greater than the number of unique entries in the run, the number of misses is equal to the number of unique entries. This was verified for various sizes of the LRU Evict Unit and Pre-fetch Unit. Table 3 gives this data for random input. The algorithm is compared with traditional LRU replacement algorithm. A performance improvement of 3% was seen. This is shown in Table 4.

Table 1 Buffer Cache Size = 512 entries. random input size=100000 entries. The rows are c and columns p

c/p	64	32	16	8	4	2
64	42615	44575	45323	45725	45861	46001
32	43145	44582	45349	45750	45907	46024
16	44483	46039	46800	47206	47380	47460
8	45202	46781	47558	47978	48146	48215
4	45666	47251	47955	48359	48513	48623
2	45861	47379	48121	48530	48730	48818
0	46019	47583	48328	48719	48915	48998

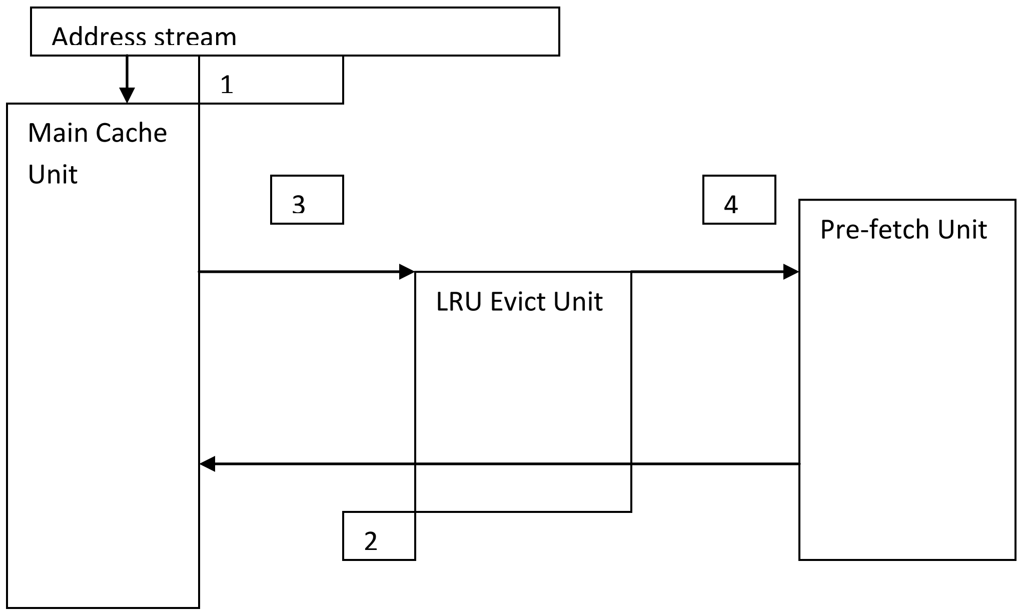

Figure 1 Proposed Buffer Cache Model

Table 2 Buffer Cache Size = 256 entries random input size = 100000 entries. The rows are c and columns p.

c/p	64	32	16	8	4	2
64	66436	68980	70178	70669	71005	71092
32	66803	69060	70132	70717	70973	71117
16	68261	70528	71619	72215	72513	72646
8	69043	71275	72350	72895	73222	73358
4	69370	71622	72763	73340	73632	73796
2	69513	71834	72991	73574	73857	74005
0	69727	72087	73182	73751	74034	74199

Table 3 Buffer Cache Size = 1024 entries random input size = 100000 entries. The rows are c and columns p.

c/p	64	32	16	8	4	2
64	1000	1000	1000	1000	1000	1000
32	1000	1000	1000	1000	1000	1000
16	1000	1000	1000	1000	1000	1000
8	1000	1000	1000	1000	1000	1000
4	1000	1000	1000	1000	1000	1000
2	1000	1000	1000	1000	1000	1000
0	1000	1000	1000	1000	1000	1000

Table 4 Performance Comparison for random input of 100000 entries

size	lru	proposed	%improve
512	46028	42615	3.413
256	69752	66436	3.316

The simulated results are compared with buffer cache proposed in [10] of same size. The buffer cache algorithm proposed in [6] performs same as the proposed algorithm in [11] for sequential input of size 500001 entries and random input of size 10001 entries. Hence, the comparisons were confined to the algorithm in [11] to save computational time.

5. CONCLUSION

This paper proposes an algorithm for buffer cache. The proposed algorithm assumes that the system has three units Main cache unit, LRU Evict Unit and Pre-fetch Unit. The algorithm is based on the concept of providing second chance to the LRU accessed record in the buffer cache. The algorithm is simulated and performance improvement of 3% is seen over LRU replacement algorithm for random input while the performance is comparable with LRU replacement algorithm for sequential input.

6. REFERENCES

[1] Ali R. Butt, Chris Gniady, and Y.Charlie Hu, The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms, ACM SIGMETRICS, '05.

[2] Elizabeth J. O'Neil and Patrick E. O'Neil, UMass/Boston, The LRU-K Page Replacement Algorithm for Database disk Buffering, SIGMOD,1993

[3] H.Seok Jeon, Sam H.Noh, A Database Disk Buffer Management Algorithm based on Prefetching, Proceedings of the seventh international conference on Information and Knowledge Management 1998, pp-167-174

[4] Hui Lei, Dan Ducha mp, An Analytical Approach to File Prefetching, Proceedings of the USENIX 1997 Annual Technical Conference, pp-275-288, 1997.

[5] Jong Min Kim_ Jongmoo Choi_ Jesung Kim_Sam H. Noh, Sang Lyul Min_ Yookun Cho, Chong Sang Kim, A Low-Overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References, OSDI, 2000

[6] Mukesh Kumar Chaudhary, Manoj Kumar, Mayank Rai, A Modified Algorithm for Buffer Cache Management, IJCA, No. 12, Article 8

[7]Mark Palmer, Stanley B. Zdonik, FIDO, A cache that learns to Fetch, Proceedings of 17th International Conference on Very Large Databases, pp255-264, 1991.

[8] Pei Cao, Edward W. Felten, Anna R. Karlin, Kai Li, A Study of Integrated Prefetching and Caching Strategies, Measurement and Modeling of Computer Systems, 1995

[9] Pei Cao, Edward W. Felten, Anna R. Karlin, Kai Li , Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching and Disk Scheduling, ACM Transactions on Computer Systems, 1996

[10] Song Jiang and Xiaodong Zhang, LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance, Proc. of SIGMETRICS 2002

[11] S Subha, An Algorithm for Buffer Cache Management, ITNG, 2009.

[12] Theodore Johnson, Dennis Shasha, 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, Proceedings of the Twentieth International Conference on Very Large Databases, 1994