Recency and Prior Probability (RPP) based Page Replacement Policy to Cope with Weak Locality Workloads having Probabilistic Pattern

Bhatta Jagdish
Central Department of Computer Science and IT
Tribhuvan University, Kathmandu, Nepal

Saud Arjun Singh
Central Department of Computer Science and IT
Tribhuvan University, Kathmandu, Nepal

ABSTRACT
Development of efficient block replacement policy is the topic of much research in operating system as well as database management systems. Among variety of page replacement algorithms Least Recently Used (LRU) algorithm is simple, flexible and has low overhead. LRU replaces page that is not accessed for longest time. But LRU makes bold assumption on recency factor only, which made LRU misbehave with weak locality workloads. This paper proposes a new Recency and Prior Probability (RPP) based page replacement policy for block replacement. The RPP combines recency and prior probability associated with the pages while selecting the victim frame from memory. If a page, to be used, has probability n times higher than another page, the page will get log(n) more chances to stay in the memory than lower probability pages. Hence with RPP, it is possible to prevent pages having higher probability of use from being replaced

General Terms: Page Replacement, Memory Management, Operating System

Keywords: LRU, LRFU, RPP, Weak locality, Probabilistic Pattern

1. INTRODUCTION

The program or data page that is not currently in main memory needs to get fetched into memory for which some other page should be removed from memory to allocate the space for incoming page, because generally memory is fully allocated to increase degree of multiprogramming. The process of choosing a page frame to replace, when a page fault occurs, is called page replacement and, the page frame chosen for the replacement is called victim frame.

Development of efficient block replacement policy is the topic of much research in operating system [1, 2] and database management systems [3,4,5]. A good block replacement policy should fulfill two criterions. First, it should be able to distinguish between hot and cold blocks. Along with, it needs to identify the blocks that are getting hot and blocks that are getting colder. Second, the policy should be efficient to implement both in terms of space and time. It is not better if the policy needs to remember large past history. Generally, time needed to execute the policy should be possibly O(1) or O(logn) in the worst case.

Among variety of page replacement algorithms, Least Recently Used (LRU) algorithm is simple, flexible and has low overhead. LRU replaces page that is not accessed for longest time. LRU adapts faster during change in working set with workloads having good locality of reference. But LRU makes bold assumption on recency factor only, which made LRU misbehave with weak locality workloads. Recency factor is the virtual time difference between the current time and last time when the oldest block is accessed.

LRU does not perform well with weak locality of reference. The access patterns of weak locality workloads can be categorized into three different groups [6];

- Sequential accesses over a large number of pages, such as "sequential scans" through a large file, may cause replacement of commonly referenced pages.
- Accesses inside loops with working set size slightly larger than the available memory, may replace pages that would be reused soon.
- LRU cannot distinguish pages with different access frequency or cannot efficiently manage an irregularly accessed page.

As a matter of fact, if the "frequency", of each page reference is taken into consideration, it will perform better in the case where workload has weak locality. Having analyzed the advantages and disadvantages of LRU and LFU, another page replacement algorithm LRFU (Least Recently frequently used) was proposed by combining them through weighting "page recency" and "frequency" factors [7]. Other studies has been performed by combining recency and frequency factor of pages [3,4,6].

Until up to now, no block replacement policy has made the use of prior probability associated with pages to select the page to be evicted. There are many areas where prior probability of some pages is known to be higher than other pages. For example, every time when a process takes a turn it definitely uses the page that contains page table but other pages associated with the process may or may not be used in the turn. With this fact, it can be said that probability of using page table pages is higher than probability of using non-page table pages. Making this bottom-line, this paper proposes a new block replacement policy that combines recency and prior probability associated with the pages while selecting the victim frame from memory.

2. RELATED WORK

As recent past is a good indicator of the near future, LRU considers that a page that is just now used will probably be used again very soon, and a page that has not been used for a long time, will probably remain unused. Here, recency is evaluated by maintaining LRU stack sorted on the basis of virtual time, which is the only factor for replacement. When page fault occurs, the page that has been unused for the longest time is evicted. Thus LRU is simple and easy to implement. It can adapt faster according as program behavior.

LRU like algorithm doesn't suffer from Belady's Anomaly as FIFO [8].

LRU shows more page faults in case of weak locality workloads. This miss behavior of LRU can be reduced by taking user level hints, utilizing and tracing history information, and detecting and adaptation of access regularity. By taking user-level hints, applications are hinted during caching and pre-fetching which rely on users understanding of data access patterns. Hence such work is only suitable for working manually, which eradicates burden of programmer. Detection and adaptation of access regularities is performed case by case in different algorithms like SEQ, EELRU, DEAR, AFC, UBM etc. Tracing and utilizing deeper history information is performed in different algorithms like LRFU, LRU-K, 2Q, ARC etc. including LIRS. For such deeper history information high implementation cost, and runtime overhead is required [6].

Most Recently Used (MRU) algorithm also works on the basis of recency factor as in LRU. It violates LRU principle and works totally in opposite manner. LRU evicts unused page following locality of principle but MRU evicts recently used page as victim. MRU is only suitable when there weak locality of reference, which is worst case of LRU. MRU can be implemented in similar way as LRU by maintaining recency stack. But here front one is removed and bottom one is stored for future use. Hence MRU is only suitable in case of worst locality of reference where LRU could not deal with this effect [6].

Least Frequently Used (LFU) selects a victim page that has not been used often in the past. Instead of using a single recency factor as LRU, additionally LFU maintains frequency of each page, which is equal to number times of the page used. This frequency is calculated throughout the reference stream by maintaining counting information. Frequency count leads to serious problem after a long duration of reference stream. Because when the locality changes, reaction to such certain change will be extremely slow. Assuming that a program either changes its set of active pages, or terminates and is replaced by a completely different program, the frequency count will cause pages in the new locality to be immediately replaced since their frequency is much less than the pages associated with the previous program. Since the context has changed, the pages swapped out will most likely be needed again soon which leads to thrashing. One way to remedy this is to use a popular variant of LFU, which uses frequency counts of a page since it was last loaded rather than from the beginning of the page reference stream. Each time a page is loaded, its frequency counter is reset rather than being allowed to increase indefinitely throughout the execution of the program. LFU still tends to respond slowly to change in locality [7].

The SEQ algorithm [9] can be considered as an adaptive version of LRU that tries to correct the performance loss caused by the presence of linearly sequential memory accesses. When it identifies one or more memory reference sets to numerically adjacent addresses, the algorithm adopts a pseudo-MRU replacement strategy, otherwise the original LRU criterion.

Some algorithms use recency as history information like LRU and Most Recently Used (MRU). These two algorithms can be tuned to form adaptive algorithm called Early Eviction LRU (EELRU) [10], which was proposed as an attempt to mix LRU and MRU, based only on the positions on the LRU queue that concentrate most of the memory references. This queue is only a representation of the main memory using the LRU model, ordered by the recency of each page. EELRU detects potential sequential access patterns analyzing the reuse of pages. One important feature of this algorithm is the detection of non-numerically adjacent sequential memory access patterns. Two tunable parameters used are early eviction point and late eviction point. LRU queue that concentrate most of the memory references when it reaches late eviction point.

Least Frequently Used (LFU) algorithm uses frequency factor for page replacement. LRU and LFU are tuned to form adaptive algorithm called Least Recently Frequently Used (LRFU) [11] that considers both recency and frequency factors.

LRU - K [4] evicts the page that is the one whose backward K-distance is the maximum of all pages in buffer. Backward K-distance bt(p,K) can be defined as the distance backward to the K^{th} most recent reference to page p, where reference string known up to time t is (r1, r2, …,rt). The value of parameter K can be taken as 1, 2 or 3. If K=1, it works as simple LRU algorithm. Highly increasing value of K reduces the overall performance of algorithm. LRU-K can discriminate better between frequently referenced and infrequently referenced pages. Unlike the approach of manually tuning the assignment of page pools to multiple buffer pools, LRU-K does not depend on any external hints. Unlike LFU and its variants, this algorithm copes well with temporally clustered patterns.

2Q [3] algorithm quickly removes sequentially and cyclically referenced block with a long interval. The algorithm uses special buffer queue A1in of size Kin, ghost buffer queue A1out of size Kout, and the main buffer of size Am. Special buffer contains all missed that is first time referenced block. Ghost buffer contains replaced blocks from special buffer. Frequently accessed block are available in main buffer. Hence victim blocks are always from special buffer and main buffer.

Another important algorithm is LIRS. Its objective is to minimizing the deficiencies presented by LRU using an additional criterion named IRR (Inter- Reference Recency) that represents the number of different pages accessed between the last two consecutive accesses to the same page. The algorithm assumes the existence of some behavior inertia and, according to the collected IRRs, replaces the page that will take more time to be referenced again. This means that LIRS does not replace the page that has not been referenced for the longest time, but it uses the access recency information to predict which pages have more probability to be accessed in near future [6].

The clock-based approximations, such as CLOCK [12], CLOCK-PRO [13], and CAR [14], usually cannot achieve the high hit ratio compared to their corresponding original algorithms (LRU, LIRS, ARC [15] respectively). They organize buffer pages into circular list, and use a reference bit or a reference counter to record access information for each buffer page. When a page is hit in the buffer, the clock-based approximations set the reference bit or increment the counter, instead of modifying the circular list themselves. As a lock is not required for these operations, their caching performance is scalable. However, the clock-based approximations can record only limited history access information, i.e. whether a page has been accessed or how many times it has been accessed but not in what order the accesses occur. The lack of richer history information can hurt their hit ratios. Moreover, many sophisticated replacement algorithms do not have clock based

approximations since the access information they need cannot be approximated by the clock structure [16].

Three other algorithms, DEAR [17], AFC [18] and UBM [19], analyze the memory accesses looking for some specific patterns, including sequential accesses. They adopt a different replacement criterion for each pattern. For example, DEAR applies MRU for sequential accesses and LRU or LFU for other patterns.

Recent adaptive algorithms use Artificial Intelligence techniques for adaptation. For example the FPR [20] and FAPR [21] algorithms apply fuzzy inference techniques to manage the replacement priorities of the resident pages. All these proposals bring important conceptual benefits to the traditional page replacement algorithms, but they also present more complex implementations. In many cases additional data structures to hold nonresident pages are necessary which leads to increased space requirements. Some algorithms require data update in every memory access, making impracticable its real implementation.

3. DESCRIPTION OF RPP

RPP combines the recency and prior probability of the pages to select the victim frame from memory. Main theme of RPP is to prevent pages having higher probability being replaced even if it has highest recency value. Suppose that a computer system having memory of M pages and is running N processes in pseudo parallel fashion by using round-robin scheduling algorithm (It can be assumed using round-robin scheduler without loss of generality). Since the device is executing N processes, there should be N page tables, one page table for each process. And each process contains M/N pages in average.

When a process takes its turn on CPU, definitely a page containing page table will be used by the process but other pages associated with process may or may not be used. Since there are M pages in total, probability of using a page is 1/M. But there are N page table pages and they are used every time when a process takes turn on CPU, therefore probability of using a page-table page is N/M. LRU does not differentiate between page table pages and non-page table pages while selecting a victim frame. Based on this idea, it is not fair to treat page table and non-page table pages equally while page replacement since the probability of accessing page-table pages is higher than the non page-table pages. Thus, the motive of RPP is to give higher priority to the pages having higher probability and lower priority to the pages having lower probability. If a page has probability N times higher than another page, RPP gives log(n) more chances to such pages before being replaced. Thus when a page table page (page having higher probability) has highest recency, instead of replacing the page RPP sets it's recency to zero and decrements the probability of the page by half. This means RPP replaces the page only when

Recency * prior probability ≤ 1

This equation gives equal weights to both recency and prior probability. If a page has probability k times higher than another page the equation satisfies only when first page have actual recency k times higher than second page and hence gets more priority over second page. Here, actual recency means recency without resetting the recency of higher probability page to zero when it is has highest recency among all pages in cache.

Algorithm of RPP is given below

1. Begin
2. Read new page, say b
3. If b is available in Queue,. Page hit occurs.
   - 3.1. Then
   - 3.2. Move b to front of Queue
4. Else
   - 4.1. If b is not available in Queue, page miss occurs
   - 4.2. If queue is full, Examine page at rear
     - 4.2.1. Then
     - 4.2.2. If recency * prior probability ≤ 1
       - 4.2.2.1. Then
       - 4.2.2.2. Remove the page at rear
     - 4.2.3. Else
       - 4.2.3.1. Move page at rear of queue to front
       - 4.2.3.2. Decrease prior probability by half
       - 4.2.3.3. Go to step 4.2
   - 4.3. Else
     - 4.3.1. Insert new page at front of queue
   - 3.4. End if
5. End if
6. End

4. IMPLEMENTATION

RPP has been implemented by using the data structure similar to LRU. Doubly linked list is used because the position of a node in the doubly linked list can be changed in O(1) time, in case of page hit. If there is page fault and no free memory is available, LRU can replace the page that is at rear of queue in O(1) time. But RPP may need O(K) time in worst case to find a victim page frame, where k in the number of processes that are executing in pseudo parallel. All memory traces are generated by assuming that Round-Robin Process scheduler is used and each process uses eight pages on average when it gets CPU.

5. PERFORMANCE ANALYSIS

RPP is evaluated by comparing it with LRU and LRFU by varying the degree of multiprogramming from 32 to 128 in the interval of 32.

Fig 1 shows that performance of LRFU is good in general in comparison to LRU and RPP. Here the degree of multi-programming is 32, therefore there are very few page-table pages (i.e. pages having higher prior probability) which are used more often than other pages. Due to which RPP is not able to take great benefit from pages having higher prior probability.

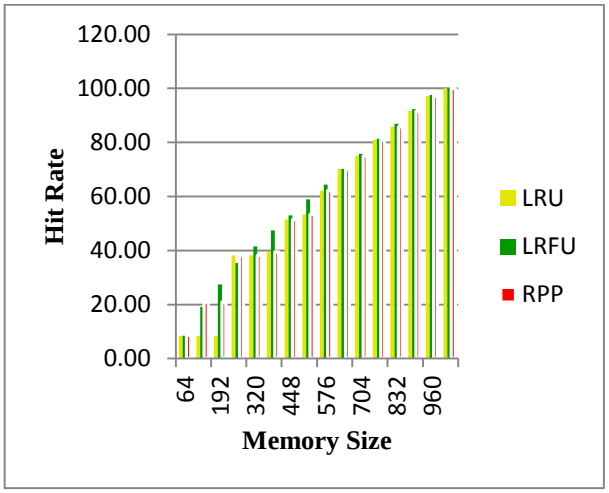

Fig 1 Graph showing performance of algorithms for degree of multiprogramming 32

Following there graphs (Fig 2, 3, 4) shows that performance of RPP is good in general in comparison to LRU and LRFU. This is because, in theses graphs degree of multi-programming is 64, 96, and 128 respectively and hence there are 64, 96, and 128 page-table pages that are used more often than other pages. Therefore RPP takes benefit of such large number of pages having higher prior probability (i.e. page-table pages) and keeps them in memory hence resulting in less number of page faults.

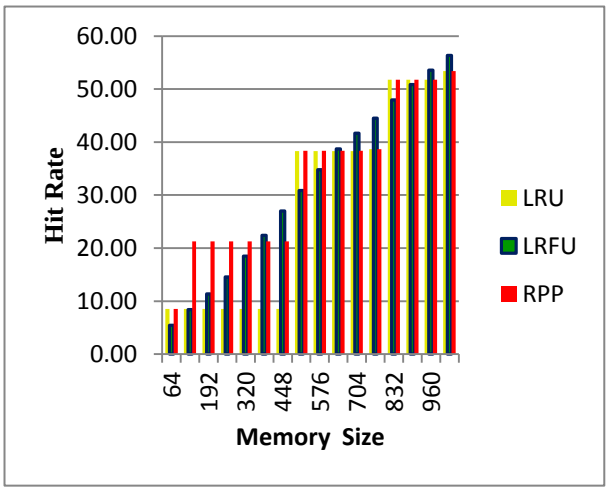

Fig 2 Graph showing performance of algorithms for degree of multiprogramming 64

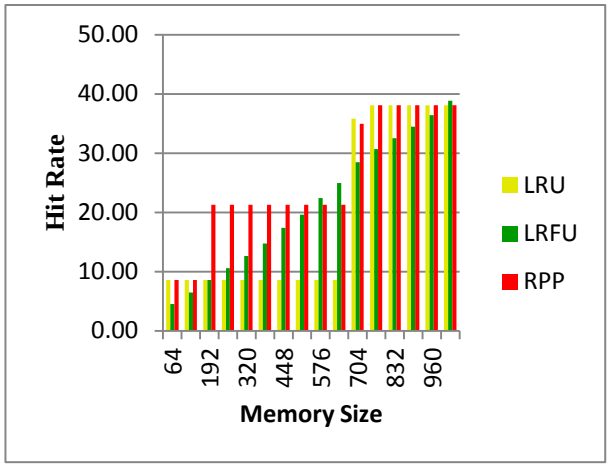

Fig 3 Graph showing performance of algorithms for degree of multiprogramming 96

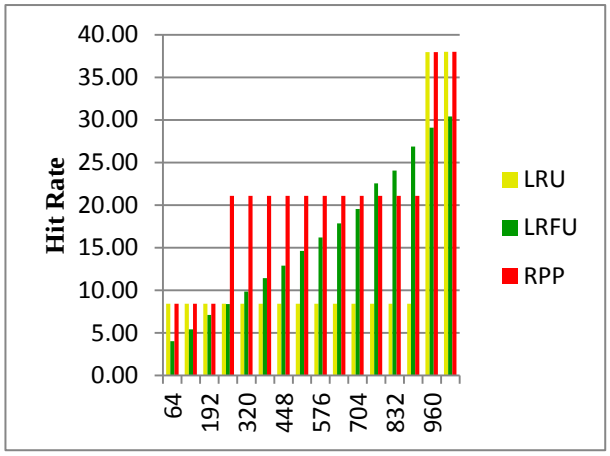

Fig 4 Graph showing performance of algorithms for degree of multiprogramming 128

Following graph summarizes above all graphs by showing average hit rates of all considered page replacement algorithms. It shows that average performance of RPP is less than other algorithms when degree of multiprogramming is 32 but its performance is becoming more better if degree of multiprogramming is increased to 64, 96, and 128.

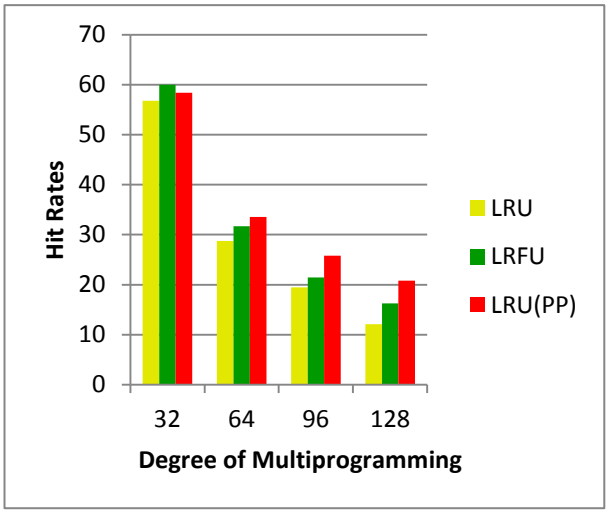

Fig 5 Graph showing average performance of algorithms for varying degree of multiprogramming

6. CONCLUSION

LRU and its variants are used in many systems due to its simplicity during implementation and better hit rates with strong locality workloads. But LRU performs weak with weak locality workloads. LRFU and other variants of LRU enhance the performance with weak locality workloads by combining frequency with recency. To calculate frequency of a page, memory management systems need to remember deep history information which adds overhead to the system. Therefore rather than combining recency and frequency, RPP combines recency with prior probability associated with the page for which systems do not need to remember past history of the page access and can be calculated easily.

From the analysis, it is found that performance of RPP is weaker than LRFU but better than LRU when degree of multiprogramming is 32. But, as the degree of multiprogramming is increased (see Fig. 2, Fig. 3 and Fig. 4) performance of RPP is becoming better than LRU as well as LRFU in average. Therefore, it can be concluded that when large number of process are executed in pseudo parallel, RPP performs better than LRFU with weak locality workloads having probabilistic pattern. The reason behind this is large number of pages having higher prior probability (i.e. page-table pages) which are in favor of RPP page replacement policy.

Worst case complexity of RPP is O(k), where k is the degree of multiprogramming, but from empirical analysis, it is believed that its complexity is quite below than O(k). Therefore calculation of exact analysis of RPP is of future research. Again, looking at graph of Fig. 4, LRFU shows anomalous behavior. Thus, analyzing this behavior of LRFU may be another future research.

7. REFERENCES

[1] P. Cao, E. W. Felten, and K. Li, "Application-controlled File Caching Policies", *Proc. USENIX Summer 1994 Technical Conf.*, pp.171-182, June 1994.

[2] R. Karedla, J.S. Love, and B.G. Wherry. Caching strategies to improve disk performance. IEEE Computere, 27(3):38 46, March 1994.

[3] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," *Proc. 20th Int'l Conf. Very Large Data Bases*, pp. 439-450, Sept. 1994.

[4] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering", *Proc. 1993 ACM SIGMOD Int'l Conf. Management of Data*, pp. 297-306, May 1993.

[5] J.T. Robinson and N.V. Devarakonda, "Data Cache Management Using Frequency-Based Replacement," *Proc. 1990 ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems*, pp. 134-142, May 1990.

[6] Song Jiang and Xiaodong Zhang, Making LRU Friendly to Weak Locality Workloads: A Novel Replacement Algorithm to Improve Buffer Cache Performance, IEEE Transactions on Computers, Vol. 54, and No. 8, August 2005, pp 939-952.

[7] D. Lee, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim, "On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies", *Proc. 1999 ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems*, pp. 134-143, May 1999.

[8] Kirby McMaster, Samuel Sambasivam, Nicole Anderson 2010, A case study of Belady's anomaly and binomial distribution, Proceedings of Informing Science & IT Education Conference (InSITE)

[9] Glass, G. and Cao, P. 1997. Adaptive Page Replacement Based on Memory Reference Behavior, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'97), pp 115-126.

[10] Smaragdakis, Kaplan, S., and Wilson, P. 1999. EELRU: Simple and Effective Adaptive Page Replacement, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'99), Atlanta, pp 122-133.

[11] Gyan Prakash Joshi, 2007, Calculation Of Control Parameter λ That Results Into Optimal Performance In Terms Of Page Fault Rate In The Algorithm Least Recently Frequently Used(LRFU) For Page Replacement, Master's Thesis, Tribhuvan University, Central Department of Computer Science and Information Technology.

[12] Corbató, F. J. 1968. A paging experiment with the Multics system. In Honor of P. M. Morse, pp 217–228, MIT Press, 1969. Also as MIT Project MAC Report MAC-M-384.

[13] Jiang, S., Chen, F., Zhang, X. 2005. CLOCK-Pro: An effective improvement of the CLOCK replacement. In Proceedings of the 10th Annual USENIX Technical

[14] Bansal, S. and Modha, D. S. 2004. CAR: Clock with Adaptive Replacement, In Proceedings of the USENIX Conference on File and Storage Technologies (FAST'04), San Francisco, pp 187-200

[15] Megiddo, N. and Modha, D. S. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache, In Proceedings of the USENIX Conference on File and Storage Technologies (FAST'03), San Francisco, pp 115-130.

[16] BP-Wrapper: A System Framework Making Any Replacement Algorithms (Almost) Lock Contention Free Xiaoning Ding, Song Jiang, Xiaodong Zhang, pp 370.

[17] Choi, J. et al. 1999. An Implementation Study of a Detection-Based Adaptive Block Replacement Scheme, USENIX Annual Technical Conference, 239-252.

[18] Choi, J. et al. 2000. Towards application/file-level characterization of block references: a case for fine-grained buffer management. In: Proceeding ot the 25th International Conference on Measurement and Modeling of Computer Systems, Santa Clara, CA. (SIGMETRICS'00), pp 286-295.

[19] Sabeghil, M. and Yaghmaee, M. H. 2006. Using fuzzy logic to improve cache replacement decisions. IJCSNS International Journal of Computer Science and Network.

[20] Kim, J.M. et al. 2000. A low-overhead high-performance unified buffer management scheme that exploit sequential and looping references. In Symposium on Operating System Design and Implementation, San Diego. OSDI' 2000 USENIX, pp 119-134.

[21] Bagchi, S., Nygaard, M. 2004. A Fuzzy Adaptive Algorithm for Fine Grained Cache Paging. 8th International Workshop (SCOPES'04), Netherlands, pp 200-213.