Journal of Advances in Computer Research
Quarterly ISSN: 2008-6148
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 3, No. 4, November 2012), Pages: 67-75
www.jacr.iausari.ac.ir

# An Improvement in WRP Block Replacement Policy with Reviewing and Solving its Problems

**Davood Akbari Bengar [1]✉, Hoshang Jazayeri Rad [2], Golnaz Berenjian[3]**

*(1) Department of Computer Sciences, Science and Research Branch, Islamic Azad University, Khouzestan, Iran*
*(2) Department of Computer Sciences, Mahshahr Branch, Islamic Azad University, Mahshahr, Iran*
*(3)Tabari Institute of Higher Education*

akbari.b1980@gmail.com; jazayerirad@put.ac.ir; golnazberenjian@tabari.ac.ir

**Received: 2012/11/14; Accepted: 2012/12/31**

**Abstract**

*One of the most important items for better file system performance is efficient buffering of disk blocks in main memory. Efficient buffering helps to reduce the widespeed gap between main memory and hard disks. In this buffering system, the block replacement policy is one of the most important design decisions that determines which disk block should be replaced when the buffer is full. To overcome the problem of performance gap between hard disks and main memory, there have been lots of proposed policies. Most of these policies are the development of the Least-Recently-Used (LRU) and Least-Frequently-Used (LFU) models. WRP(Weighting Replacement Policy) is a replacement algorithm that its performance is better than the LRU and LFU. The most advantage of this model is it's similarity to both LRU and LFU, which means it has the benefits of both. This research proposes a new block replacement Policy namely, DWRP (Developed Weighting Replacement Policy) which solves the problems of WRPalgorithm and retains its advantages.*

***Keywords:*** *Block replacement policy, System performance, Main memory, Buffering system, Speed gap*

## 1. Introduction

To overcome the speed gap between hard disks and main memory, much research has been performed on buffering disk blocks in main memory. Such a buffering system is called a buffer cache and one of its most important design decisions is the block replacement policy that decides the block to be replaced when the buffer cache is full.

There is a terminology for cache efficiency namely "Hit Rate" (Hit Rate expresses the rapport between the number of addresses accessed from cache and the total number of addresses accessed during that time). The antonym for "Hit Rate" is named "Miss Rate" which can be ascertained by Miss Rate = 1 – Hit Rate formulas. Cache memory uses uniformly sized items which are called pages. The first access will be always a Miss When the cache is empty. Until the cache is filled with pages, the misses may happen. After some cycles, when the cache is filled, this kind of misses wills not happen anymore. Another kind of miss happens when there is an access to a page that is not in the cache and the cache is full of pages, in this situation replacement policy should determine conditions under which a new page will be replaced with an existing one. A good block replacement policy must be easy to implement in hardware and must have good performance with low overhead on the system [20]. In [10] proposed a replacement algorithm that have low overhead on the system and it is easy to implement in hardware. This model is named Weighting Replacement Policy (WRP) which is based on ranking of the pages in the cache according to three factors. Whenever a miss occurs, a page with the lowest rank point is selected to be substituted by the new desired page. In WRP, if weighting value of a page be more, then its rank will be lower. First factor namely $L_i$is the counter which shows the recency of block i in the buffer and second factor namely $F_i$ is the counter which shows the number of times that block i in buffer has been referenced. Third factor is the time difference ΔTi = Tci - Tpi where Tci is the time of last access and Tpi is the time of penultimate access. In this model, the weighting value of block i compute by (1).

$$W_i = \frac{L_i}{F_i * \Delta T_i} \qquad (1)$$

WRP algorithm has two basic problems. First problem of this algorithm is at above function that has been explained with an example. suppose that there are two pages A and B in cache memory and the value of $L_A/F_A$ for page A is equal to the value $L_B/F_B$ for page B and this value correspond to Y. meanwhile, $\Delta T_A$ be equal to 2 and $\Delta T_B$ be equal to 10. If a miss occurs in this condition, then the page A is selected to be substituted by the new desired page because the weighting value of page A is more than page B. but this selection is not true because $\Delta T_A$ is less than $\Delta T_B$ ($\Delta T_A < \Delta T_B$) and this means that the time difference between two last access of page B is more than page A. then the probability of referencing to page A is more than page B in future, there for the page B must be selected for replacement and not A . The second problem of WRP algorithm is that it considers only the time difference between two last accesses and doesn’t consider the previous accesses. In this research, a new page replacement algorithm namely DWRP has been proposed which solves these two problems and other trivial problems.

## 2. Background

Most of replacement policies in use are based on frequency and recency properties, but they fail in some applications. Other new policies may work better, but most of them are not easy to implement. For example, LRU policy that is a recency-based policy, fails badly for big loops. LFU policy that is a frequency-based policy, works badly when various parts of memory have various time–variant patterns. The LFU policy has many problems: it needs logarithmic implementation complexity in cache memory size, pays almost no attention to recent history, and does not conform well to altering access patterns since it collect stale pages with high frequency counts that may no longer be helpful [10]. The method Clock with Adaptive Replacement (CAR) works in much the same way as the Adaptive Replacement Cache (ARC) explained below. However, CAR avoids the last two LRU downfalls of ARC. These downfalls are: 1) the overhead of moving a page to the most recently used position on every page hit; and 2) serialized, in the fact that when moving these pages to the most recently used position the cache is locked so that it doesn’t change during this transition [6].In the Most Frequently Used (MFU) algorithm [1], the blocks that have been referenced more frequently are the candidates for replacement. However, since it doesn’t consider the recency factor, it cannot differentiate between blocks that were once hot but now becoming cold and blocks that are currently hot. If frequency density of a block is high(the references to the block are dense), that block is said to be hot [1]. The memory system consists of a hierarchy of caches. The performance of the entire memory hierarchy will suffer, if each cache in the hierarchy decides which block to replace in an independent way, particularly if inclusion is to be maintained [15]. In these cases a global strategy usually has a limited performance potential. But an optimal global strategy could help to improve the performance. By implementing a global replacement policy for all cache levels which includes both recency and frequency features, the hit ratio of each level will have a minimal influence on the upper level [2, 8]. The LFU-K (K = 1, 2 …) algorithm increases servers' performance. This algorithm keeps the number of times that a page has been referenced and associates a value with each block which is based on a special function. This value quantifies the priority of every block and whenever a miss occurs, a block with the lowest priority is selected to be substituted by the new desired block [4].

The MF-LRU algorithm is a blend of two popular replacement policies namely the LRU and MFU replacement policies. The MF-LRU associates a value with each block called the Recency Frequency Factor (RFF), which quantifies the importance of that block i.e. the likelihood that the block be referenced in the near future. Thus the block with the least RFF is the victim for replacement [1, 21]. A replacement algorithm is proposed in [10], which imposes low overhead on the system and is easy to implement in hardware. This algorithm is named Weighting Replacement Policy (WRP) which is based on ranking of the pages in the cache. Whenever a miss occurs, a page with the lowest rank point is selected to be substituted by the new desired page. In WRP, if weighting value of a page is high, then its rank will be low. Algorithm like Taylor Series Prediction (TSP) [11] is based on recency, frequency, size and cost. This algorithm is suitable for web caching. Many replacement policies have been proposed in the last few years, like LRU-K (K = 1, 2, …) algorithm, the Greedy Dual size (GD-size) policy and Greedy Dual Size Frequency (GDSF) algorithm, which is an enhancement of the GD-size algorithm. Most of these caching algorithms like TSP are based on recency, frequency, size and cost. Most of them are suitable for web caching. Although the LRU replacement algorithm is the most popular algorithm but this algorithm isn't the cheapest in terms of hardware cost [12]. Most of the work done in cache replacement algorithm is local. Some of the proposed algorithms accommodate to the application behavior, but within a single cache. For example, Qureshi et.al suggest retaining some fraction of the working set in the cache so that some fraction of the working set cooperate to the cache hits [13].Cache misses are not of equal importance, and it depends on the amount of memory level parallelism (MLP) [14]. An MLP-aware cache replacement is presented in [19].

## 3. Motivation

A well-known performance enhancement technique that is used in computer systems is Caching. Paged virtual memory, web caching and Processor’s cache are some examples of using this technique. Performance of the system depends on cache mechanism. The time access for main memory is at least 10 times larger than the cache. Increasing the size of cache results in higher performance, but it has a very expensive technology. That’s why the cache size is always smaller than the main memory.

According to access time of the cache memory, fetching data from cache memory has an important role in system performance. Other techniques should be used until make cache more efficient for systems and the block replacement policy is an important technique for doing this task. Most of policies are the development of the Least-Recently-Used (LRU) and Least-Frequently-Used (LFU) models such as WRP (Weighting Replacement Policy).WRP is a replacement policy that its performance is better than the LRU and LFU but it has some problems. In this research, a new page replacement policy namely DWRP has been proposed which solves the problems of WRP.

## 4. DWRP Replacement Algorithm

In this section, the proposed replacement algorithm has been explained. The size of the blocks has been assumed that is equal and the replacement algorithm is used to manage a memory that holds a finite number of these blocks. A hit will occur when there is a reference to a block in the buffer. When a reference to a block not in the buffer is done, a miss will occur and referenced block must be added to buffer. When buffer is full and a miss occurs, an exciting block must be evicted to make room for a new one. The proposed replacement algorithm is a powerful tool that can increases the system performance with considering three parameters recency, frequency and reference rate of every block. In fact, it performs both LRU and LFU algorithms and gives a weight to each block according to three said parameters. First parameter namely $RB_i$is the counter which shows the recency of block i in the buffer and second parameter namely $FB_i$ is the counter which shows the number of times that block i in buffer has been referenced . Third parameter namely $\delta_{tk}(i)$ is the average of time distances for block i from begin to tk time (a time distance namely ΔT is the time difference between two sequential access of a block). If block i referenced at time tk then third parameter computes for block i by (2).

$$\delta_{tk}(i) = \frac{\delta_{t(k-1)}(i)+\Delta T_{k-1}}{2} \quad (2)$$

Now suppose that block i referenced at times t0,t1, t2 and t3 (Fig. 1) and $\delta_{t0}(i)$ be equal to 1 then $\delta_{t3}(i)$ computes by (3), (4), (5).

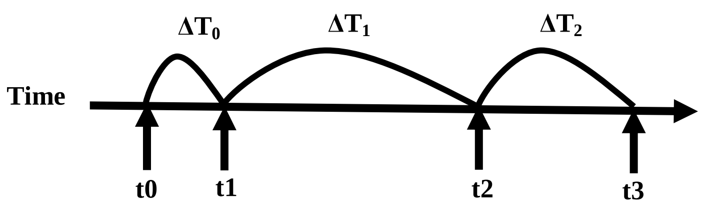

***Figure 1. Reference to block i at times t0, t1, t2 and t3***

$$\delta_{t1}(i) = \frac{\delta_{t0}(i)+\Delta T_0}{2} \quad (3)$$

$$\delta_{t2}(i) = \frac{\delta_{t1}(i)+\Delta T_1}{2} \quad (4)$$

$$\delta_{t3}(i) = \frac{\delta_{t2}(i)+\Delta T_2}{2} \quad (5)$$

Based on three said parameters, the weighting value of block i at time tk compute by (6).

$$WB_i = \frac{RB_i}{FB_i} * \delta_{tk}(i) \qquad (6)$$

$RB_i$ will work like LRU counter, when the new block i is placed in the buffer then all of the parameters in weighting function must be set, and it will be followed by setting $RB_i$ to 1 , $FB_i$ to 1 and $\delta_{t0}(i)$ to 1. $FB_i$ has been set to 1 because it means that the block i has been used once and $\delta_{t0}(i)$ has been assumed that be equal to 1 because the time between each reference to a block would be at least one in its minimum case. In every access to buffer at time tk, if referenced block j is in the buffer then a hit is occurred and the proposed algorithm will work in this way:

1) $RB_i = RB_i+1$ ، for every $i \neq j$
2) $\delta_{tk}(j) = \frac{\delta_{t(k-1)}(j)+RB_j}{2}$ , $FB_j = FB_j+1$ and $RB_j = 1$

In accessing to buffer at time tk if referenced block j is not in the buffer a miss occurs and the algorithm will choose the block in buffer which the value of its weighting function is greater than the others. Searching for block with greatest weighting value will be started in the buffer from top to down. In this way, if values of some blocks are equal to each other, the block which has been referenced less frequently will be chosen to evict from buffer. It means that the proposed algorithm follows LFU low in its nature. Let assume that a miss has been occurred and block t has the greatest weighting value and it must be evicted from buffer, thus the proposed algorithm will work in this way:

1) $RB_i = RB_i+1$ ، for every $i \neq j$
2) $FB_t = 1$
3) $FB_j = FB_j+1$ ، $\delta_{tk}(j) = \frac{\delta_{t(k-1)}(j)+RB_j}{2}$ and $RB_j = 1$

The weighting value of blocks that are in buffer will update in every access to cache. As in simulation section has been shown, the proposed algorithm will work better than WRP, LRU, LFU and FIFO with different cache sizes. One of the important concepts in replacement algorithms is its overhead in the systems. The proposed algorithm needs three counter to work and will add space overhead to system: first, algorithm needs a space for counter $RB_i$ second, it needs a space for counter $FB_i$ and third it needs a counter for $\delta_{tk}(i)$. The last and maximum space that it needs is a space for $WB_i$, which is as weighing value for each block in the buffer. Calculating weighting function value for each block after every access to cache will cause a time overhead to system. Although the $WB_i$ is considered as an integer number to decrease the time and memory overheads.

## 5. Simulation

To evaluate the proposed algorithm experimentally, the proposed algorithm has been simulated in C programming language and has been compared with other algorithms like WRP, LRU, LFU and FIFO. The simulator program was designed to run thousands trace of memory addresses by proposed algorithm and four said algorithms. The obtained hit ratio depends on the replacement algorithm, cache size and the locality of reference for cache requests. An address trace is simply a list of one hundred numbers between 0 and 10 that generate randomly by a program. In fact, every of these 11 numbers are explanatory of one memory address.

### *5.1 Simulation Results*

The simulation program has been executed twice and with 4 different cache sizes until evaluates the performance of the proposed algorithm. At first time, 1000 different address traces have been considered as input of the simulation program. The simulator acquires the hit ratio for each of 1000 traces and then computes the average of hit ratios. The average of hit ratios for each of cache sizes has been compared with WRP, LRU, LFU and FIFO. As it is indicated in Fig. 2, if 1000 different address traces are executed by simulation program (NT=1000) then on the average, the proposed weighting replacement algorithm works about 0.4% better than WRP and about 0.22% Better than LRU. In Fig. 2, the proposed algorithm has been compared with WRP, LRU, LFU and FIFO by thousands address traces. The simulator computed the accurate $WB_i$ for each reference, but for implementing there is no necessarily to use float value and it can be simplified to an integer number. At second time, the simulator received 2000 different address traces as input and then computed the average of hit ratios. As it is indicated in Fig. 3, the maximum average of hit ratios for DWRP is 68.315%, and is about 0.13% more than WRP and about 0.14% better than FIFO. In the worst case the result is equal to the maximum average of hit ratios of WRP, LRU, LFU and FIFO. Fig. 3 shows the comparison and the results for 2000 address traces. This figure shows that the proposed algorithm is always better than four other algorithms.

***Table 1. A Comparison between hit ratio of DWRP, WRP, LRU, LFU and FIFO for 4 different cache sizes (NT=1000). In this case, an enhancement compare to WRP, LRU, LFU and FIFO for DWRP in all cache sizes is seen***

| Cache Size (# of Blocks) | DWRP % | WRP % | LRU % | LFU % | FIFO % |
|---|---|---|---|---|---|
| 2 | 17.944 | 17.765 | 17.894 | 17.913 | 17.831 |
| 4 | 35.514 | 35.161 | 35.276 | 35.37 | 35.385 |
| 6 | 52.57 | 52.042 | 52.212 | 52.327 | 52.324 |
| 8 | 68.68 | 68.418 | 68.47 | 68.334 | 68.456 |

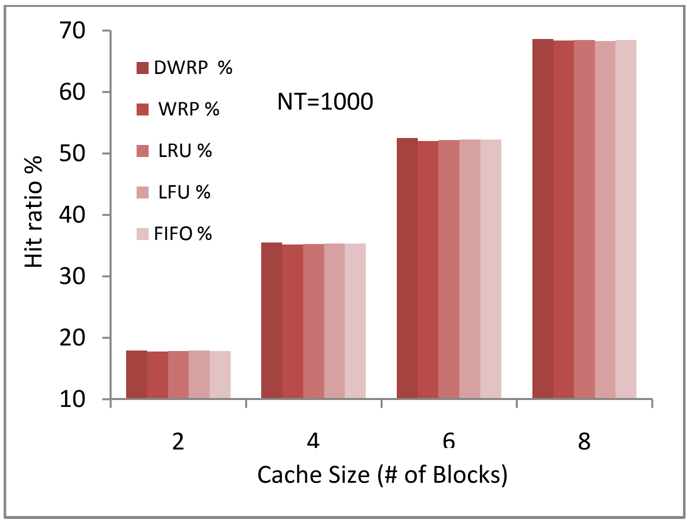

***Figure 2.Performance of DWRP, WRP, LRU, LFU and FIFO with different cache sizes for NT=1000. DWRP performs better than other algorithms***

***Table 2.A Comparison between hit ratio of DWRP, WRP, LRU, LFU and FIFO for 4 different cache sizes (NT=2000). In this case, an enhancement compare to WRP, LRU, LFU and FIFO for DWRP in all cache sizes is seen***

| Cache Size (# of Blocks) | DWRP % | WRP % | LRU % | LFU % | FIFO % |
|---|---|---|---|---|---|
| 2 | 17.8905 | 17.878 | 17.8875 | 17.826 | 17.8725 |
| 4 | 35.3185 | 35.2615 | 35.297 | 35.2275 | 35.2425 |
| 6 | 52.2440 | 52.011 | 52.1595 | 52.172 | 51.979 |
| 8 | 68.315 | 68.1225 | 68.221 | 68.2775 | 68.132 |

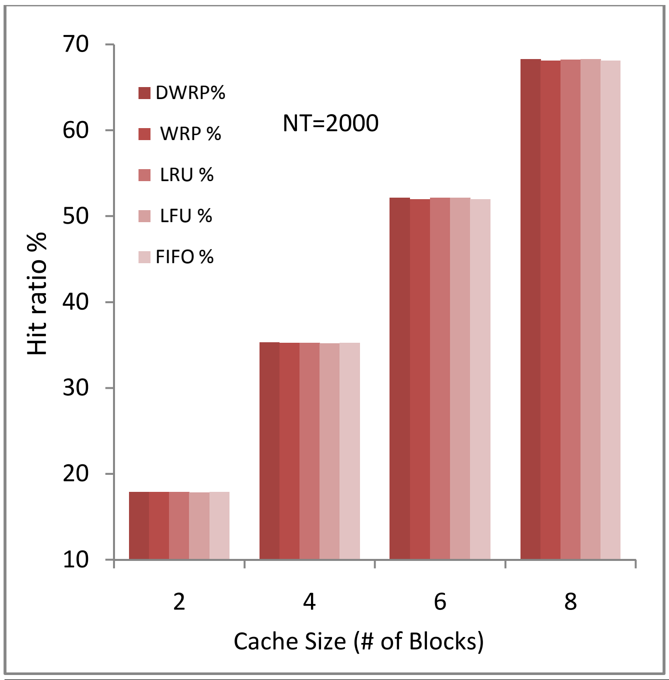

***Figure 3.Performance of DWRP, WRP, LRU, LFU and FIFO with different cache sizes for NT=2000. DWRP performs better than other algorithms***

## 6. Conclusion

In this article, a new caching replacement algorithm has been proposed that is the improvement of the WRP algorithm. The WRP algorithm has two basic problems. The first problem of this algorithm happens when the result of division of two factors (recency and frequency) for one block is equal to another block. The second problem is that it considers the only time distance between two last access for $\Delta T_i$ and doesn't consider the previous accesses. In this article by proposing a new function, removed this two problems and the other little problems. The DWRP has been simulated with two traces and has been compared with WRP, LRU, LFU and FIFO. Simulation shows that this new algorithm works better than four algorithms WRP, LRU, LFU and FIFO. Considering the additional parameters and factors which describe features of objects in the buffer would help to improvement of proposed algorithm, for instance, considering the cost and size of each object in the cache that will make DWRP suitable for applications like web caching.

## 7. References

[1] Sabarinathan Sayiraman, Senthil Kumar Dayalan, Shanmugavel Mani Subbiah, "A Framework for MF-LRU Replacement Policy", School of Computer Science and Engineering ,College of Engineering Guindy, Anna University ,Chennai, India, 2002.

[2] Richa Gupta, SanjivTokekar," A Novel Pair of Replacement Algorithms onL1 and L2 Cache for FFT",Richa Gupta et al / International Journal on Computer Science and Engineering Vol.2(1), 2010, pp.92-97.

[3] Seon-yeong Park ,Dawoon Jung ,Jeong-uk Kang ,Jin-soo Kim, and Joonwon Lee," CFLRU: A Replacement Algorithm for Flash Memory", CASES '06 Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems, January 31, 2006 .

[4] Vladimir V. Prischepa," An Efficient Web Caching Algorithm based on LFU-K replacement policy", Proceedings of the Spring Young Researcher's Colloquium on Database and Information Systems SYRCoDIS, St.-Petersburg, Russia, 2004.

[5] Sangyeun Cho , Lory Al Moakar ," Augmented Fifo Cache Replacement Policies for Low-Power Embedded Processors", Journal of Circuits, Systems, and Computers Vol. 18, No. 6 (2009), pp.1081–1092.

[6] Sorav Bansal and Dharmendra S. Modha," CAR: Clock with Adaptive Replacement ", FAST '04 Proceedings of the 3rd USENIX Conference on File and Storage Technologies, USENIX Association Berkeley, CA, USA©2004.

[7] N. Megiddo and D. S. Modha, "ARC: A Self-Tunning, Low Overhead Replacement Cache" ,Proc. Usenix Conf. File and Storage Technologies (FAST 2003), Usenix, 2003, pp.115-130.

[8] Richa Gupta ,Sanjiv Tokekar ," Proficient Pair of Replacement Algorithms on L1 and L2 Cache for Merge Sort, "Journal of Computing , Volume 2, Issue 3, March 2010, Issn 2151-9617.

[9] Qingbo Zhu, Asim Shankar and Yuanyuan Zhou," PB-LRU:A SelfTuning Power Aware Storage Cache Replacement Algorithm for Conserving Disk Energy", Department of Computer Science University of Illinois at Urbana Champaign Urbaba, IL 61801, ICS’04, June 26–July 1, 2004, Malo, France.

[10] Kaveh Samiee, "A Replacement Algorithm Based on Weighting and Ranking Cache Objects", International Journal of Hybrid Information Technology Vol.2, No.2, April, 2009.

[11] Q. Yang, H. H. Zhang and H. Zhang, " Taylor Series Prediction: A Cache Replacement Policy Based on Second-Order Trend Analysis," Proc. 34th Hawaii Conf. System Science, 2001.

[12] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic, "Performance evaluation of cache replacement policies for the spec cpu2000 benchmark suite ", in Proc. 42nd ACM Southeast Conference, 2004.

[13] M. Qureshi, A. Jaleel, Y. Patt, S. S. Jr., and J. Emer, "Adaptive insertion policies for high performance caching", in Proc. 34th Int’l Symposium on Computer Architecture, 2007.

[14] T. Puzak, A. Hartstein, P. Emma, and V. Srinivasan, "Measuring the cost of a cache miss", in Workshop on Modeling, Benchmarking and Simulation (MoBS), 2006.
[15] Mohamed Zahran. " Cache Replacement Policy Revisited," In Proceedings of the 6th Workshop on Duplicating Decon-structing, and Debugging, San Diego, CA, USA, June 2007.
[16] S. Jihang and X. Zhang, "LIRS: An Efficient Low Inter Reference Recency Set Replacement Policy to Improve Buffer Cache Performance," Proc. ACM Sigmetrics Conf., ACM Pres, pp. 31-42, 2002.
[17] Jianliang Xu, Qinglong Hu, Wang-Chien Lee and DikLun Lee, "Performance Evaluation of an Optimal Cache Replacement Policy for Wireless Data Dissemination "Ieee transactions on knowledge and data engineering, vol. 16, No. 1,January 2004.
[18] Jaeheon Jeong and Michel Dubois, "Cost-Sensitive Cache Replacement Algorithms", High-Performance Computer Architecture, 2003. HPCA-9 2003.
[19] M. Qureshi, D. Lynch, O. Mutlu, and Y. Patt, "A case for mlp-aware cache replacement", in Proc. 33rd Int’l Symposium on Computer Architecture, 2006.
[20] Debabala Swain, Bijay Paikaray and Debabrata Swain," AWRP: Adaptive Weight Ranking Policy for Improving Cache Performance". Journal Of Computing, Volume 3, Issue 2, February 2011, ISSN 2151-9617.
[21] Mingwei Lin, Shuyu Chen and Guiping Wang, " Greedy page replacement algorithm for flash-aware swap system", IEEETransactions on Consumer Electronics, Volume 58, Issue 2, May 2012, pp. 435-440.