IJCSE International Journal of Computer Science and Engineering Open Access
Technical Paper Volume-2, Issue-2 E-ISSN: 2347-2693

Pattern Based Cache Management Policies

Namrata Dafre[1*], Urmila Shrawankar[2] and Deepak Kapgate[3]

[1*]*Student of M.Tech. (CSE), GHRAET, Nagpur, Nagpur University(MS), India, namrata17dafre@gmail.com*
[2] *Department of C.S.E., GHRCE, Nagpur, Nagpur University (MS), India, urmila@ieee.org*
[3]*Department of C.S.E., GHRAET, Nagpur, Nagpur University (MS), India,deepakkapgate32@gmail.com*

www.ijcaonline.org

Received: 05 Feb 2014 Revised: 14 Feb 2014 Accepted: 26 Feb 2014 Published: 28 Feb 2014

Abstract— In a computer architecture Cache memory have been introduced to balance performance and cost of the system. To improve the performance of a cache memory in terms of hit ratio and good response time system needs to employ efficient cache replacement policy. Unified Buffer cache management, Program Counter-based Classification, Detection based Adaptive Replacement, Robust Adaptive buffer Cache management scheme and Block Pattern Based Buffer Cache Management are some of the existing policies. But they have some disadvantages like they were not able to exploit both recency and frequency information, some of them could not exploit all type of reference regularities, some of them have high memory overhead. So we require more advanced policies to improve the performance. In this work we are proposing the block access pattern based replacement policy which predicts future request of a block based on history of response time for respective data block. Block access pattern based replacement policy leads to effective improvement in buffer cache hit ratio and reduced response time.

Index Term—*Buffer* Cache, Access Patterns, Cache Replacement Policies, Buffer Cache Management Techniques

I. INTRODUCTION

Cache is high speed memory contains most recently accessed pieces of main memory. It bridges the gap between CPU and Main Memory. Increasing cache size results in better performance but it is very expensive. It is necessary because, time it takes to bring an instruction into the processor is very long when compared to the time to execute the instruction. Cache memory helps to reduce the time it takes to move information to and from the processor. Cache memory improves system performance by following a concept of Locality of Reference. The concept is that at any given time the processor will be accessing memory in a small or localized region of memory, cache memory loads this region allowing the processor to access the memory region faster. The role of Cache is illustrated in the following figure 1. The typical Cache Organization can be shown in figure 2.

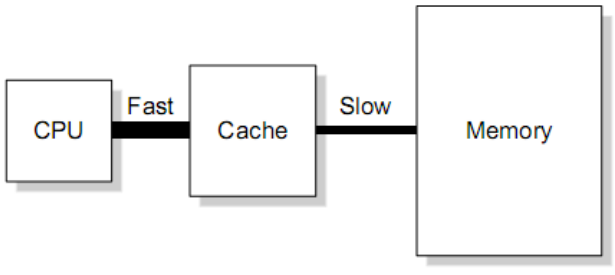

Figure 1: Cache Based Memory System

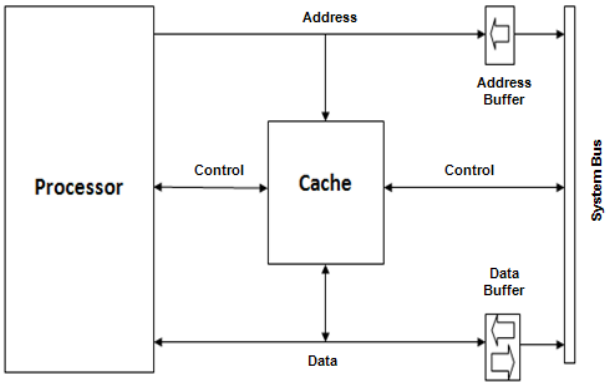

Figure 2: Typical Cache Organization

When a new block is brought into the cache, it needs to replace one of the existing blocks if cache is full. For this purpose we need replacement policies. To provide memory operands to the processor at the speed it can process them is one of the most challenging aspect. To achieve high speed, an efficient replacement policy must be implemented. A number of policies have been introduced. To have maximum hit rate a good caching algorithm must have characteristics such as,

- ✓ Low memory overhead**.**
- ✓ Faster access to data.
- ✓ Low response time.

In computer architecture, to balance performance and cost of the system Cache and buffer cache have been introduced.

Corresponding Author: *Namrata Dafre*

The hierarchy architecture consists of CPU, RAM and external storage. Cache reduces read latency, while the buffer cache is to reduce writing operations. Cache works between CPU and RAM while cache buffer works between RAM and external storage [1].

Buffer cache is an interface between the main memory and disk drive. It is introduced to reduce the frequency of access made to the secondary storage devices and enhance the system throughput [2]. In a computer system, storage devices such as registers, caches, main memory and secondary memory are present in a hierarchy. They are at various levels. At the top level there are registers which accesses data at the speed of processor usually in one clock cycle. At next higher level there exists cache memory. Primary memory is present next higher level. At higher levels, as data storage space increases, access time and the transfer bandwidth decreases. System cannot access directly to secondary storage for the data. Whenever there is a request for the data, at first the system searches block containing the requested byte in the buffer cache. If the request is not found, the system searches it into the secondary disk, brought into the buffer cache and finally passed to main memory and then to the cache & as the buffer resides in main memory the access time between main memory and buffer cache is negligible.

File Access Patterns
Processor is much faster than DRAM memory, so it is obvious that the number of processor cycles it takes to access main memory has also increased. Thus we must use other cache management technique to make cache more and more efficient for a system which make use of it in their memory hierarchy. Different workloads and programs have different accessing patterns like,

Sequential references: All blocks are accessed one after another, such as file scanning. Blocks are never re-accessed.

Looping-like references: All blocks are accessed periodically.
Temporally-clustered references: Blocks accessed more recently are the ones more likely to be accessed in the near future.

Probabilistic references: Blocks are accessed independently with the associated probabilities.

II. RELATED WORKS

Replacement algorithms are of three types. They are as follows:

1) Replacement algorithms that incorporate longer reference histories than LRU:
As LRU[5] is simple and easy to implement so it is used widely but in some cases it does not perform well so other replacement algorithms such as LRU-K[6], 2Q[13], LRFU[14], EELRU[12], MQ, LIRS[16], and ARC[15] are used which incorporate longer reference histories than LRU. These algorithms make their cache replacement decision by maintaining information of accessed blocks such as recency as well as frequency. But, they cannot exploit regularities such as looping or sequential references.

2) Replacement algorithms that rely on application hints:
Some of the replacement algorithms rely on programmers to insert useful hints such as information about future access patterns but, this technique cannot achieve satisfactory performance level if the I/O access pattern can be known only at runtime. Such application informed caching management schemes are proposed in ACFS [28] and TIP [29].

3) Replacement algorithm that actively detects the I/O access patterns:
Depending on the level at which patterns are detected, the pattern-detection based techniques are of four types:

Block-level patterns
Block level pattern detection policy detects the long sequences of page cache misses and applies the Most Recently Used (MRU) policy to such sequences to avoid scan pollution.

Application-level patterns
At this level, a scheme periodically classifies the pattern of references issued by a single application. DEAR[] observes the patterns of assuming that the I/O pattern of each application is consistent. DEAR (Detection Adaptive Replacement) uses MRU as the replacement policy to manage the cache partitions for looping and sequential patterns, LRU for the partition of the temporally-clustered pattern, and LFU for the partition of the probabilistic pattern.

File-level patterns
At the file level, the UBM (Unified Buffer Management) [23]scheme separates the I/O references according to their target files and automatically classifies the access pattern of each individual file into one of three categories such as sequential references, looping references and other references. It divides the buffer cache into three partitions, one for blocks belonging to each pattern category, and then uses different replacement policies on different partitions. For blocks in the sequentially referenced partition and periodically referenced partition, MRU[7] replacement policy is applied. For blocks that belong to other reference pattern, LRU[5] is used. This approach tends to have good responsiveness and stability due to the fact that most files tend to have stable access patterns, although large database files may show mixed access patterns.

Program-context level patterns
At this level, a scheme separates I/O stream into sub-stream according to single program context and detect the patterns of each sub-stream, assuming that a single program context tends to access files with the same pattern in the future. AMP [21] and PCC [22] are the example of program-context level replacement algorithm. This approach uses program context and has relatively shorter learning period than the file-based approach. While it can make correct classification for new files after training, it classifies the accesses to all files touched by a single program context into the same pattern category, thus it has no detection accuracy.

III. EXISTING CACHE REPLACEMENT POLICIES

LRU[5] algorithm is used widely because of its simplicity**.** LRU algorithm keeps track of the cache lines according to time they have been used. The pages which have not been used for longer time are to be replaced. LRU has some limitations such as inability to cope with access patterns with weak locality and scan pollution. LRU causes thrashing for the workloads larger than the L2 cache. When the LRU policy is used for memory intensive workload, lines that are inserted in the cache will be referenced in the future but due to the capacity misses, they will be replaced by new lines before being re-referenced.

LRU-K [6] is the improvement over LRU, it dynamically records the K'th backward distance. Backward distance of block is number of references in between last and current reference of the block. A block with the maximum K'th backward distance is dropped to make space for missed blocks. LRU-K makes its replacement decision based on the time of the K'th to last reference to the block. Hence, oldest resident block is evicted [4]. In LRU-2 simply K = 2 is taken that is the time of the penultimate reference to a block, LRU-2 quickly removes cold block from the cache.
Early Eviction LRU [12] policy is an improvement of LRU. It attempts to get advantage of both LRU and MRU. It concentrates on the positions of the memory references in the LRU queue. This queue is only a representation of the main memory using the LRU model, ordered by the recency of each page. By analyzing reuse of pages EELRU detects sequential access pattern. EELRU detects non-numerically adjacent sequential memory access patterns.

First In First Out [10] replacement policy uses a replace_ptr to indicate the cache block that is to be replaced when a cache miss occurs. 2Q [13] algorithm performs similar to LRU-K but with considerably lower time complexity. It achieves quick removal of cold blocks from the buffer by using a FIFO queue $A1_{in}$, an LRU queue A_m, and a ghost LRU queue $A1_{out.}$ When a block is newly referenced, it initially kept into $A1_{in}$. When a block is evicted from $A1_{in}$, this block's identifier is added to $A1_{out}$, this queue contents only block identifiers. If a block in $A1_{out}$ or $A1_{in}$ is re-referenced, this block is promoted to A_m [5].

Second Chance is an improvement of FIFO. It uses a reference bit for each cache block. The reference bit will be set to 1 each time the cache block is accessed. SC uses a queue, where the head of the queue represents the next cache block to be replaced upon a cache miss. LFU [5] algorithm keeps track of cache lines which are used frequently and the information which is not used frequently is discarded. It uses a bit called LFU count. MRU [7] algorithm discards, in contrast to LRU algorithm does, replaces most recently used information. Optimal replacement [9] algorithm is also known as clairvoyant algorithm. This is the most efficient algorithm which discards the information that will not be needed for longer time in future.

Least Recently/Frequently used (LRFU) [14] policy consider both recency as well as frequency information of a block. It uses a CRF value on the basis of which it makes the replacement decision. It uses a weighing function which uses all the past CRF values of that block.

In Dueling CLOCK [19], Cache is logically implemented as circular queue. It is an adaptive replacement policy which alternates between the CLOCK algorithm and the scan resistant version of the CLOCK algorithm. This policy uses hit bit, replace_ptr and circular buffer. In CLOCK algorithm , the hitbit associated with each page is set to 0 initially, whenever the page is referred hitbit is set to 1.On miss event the replace_ptr is incremented that is moved to next cache line and corresponding hitbit is checked, if it is 1 then it is set to 0 and replace_ptr is incremented and check next cache line's hitbit, if it is 0 then that cache line is replaced. This policy requires log(N) memory bits for Replace pointer to indicate which page is to be replaced and N bits for array of hitbit, one for each cache line. Therefore, the total memory overhead is N + log(N) bits. Scan Resistant version of CLOCK works same as that of CLOCK, only the difference is, on miss event it force replace_ptr not to advance to next cache line. Dueling CLOCK policy have low overhead cost, it captures recency and frequency information and it is scan resistant [29].

To minimize the deficiencies presented by LRU, new replacement policy is proposed called as LIRS (Low Inter-reference Recency Set)[16]. To make replacement decision it uses Inter Reference Recency of a block. IRR refers to the number of other blocks accessed between two consecutive references to the block in history. According to the collected IRRs, policy 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 a near

future.The blocks having low IRR are called as LIR blocks and blocks having high IRR are called as HIR blocks. LIRS divides cache into two sets: High Inter-reference Recency (HIR) block set and Low Inter-reference Recency (LIR) store LIR blocks, and the minor part is used to store HIR blocks. A HIR block is replaced when the cache is full.
The pages that were used recently only once, L2 keeps track of pages that were used more than once. That means L1 captures recency while L2 captures frequency. That means it dynamically chooses among LRU and LFU. According to misses the policy will adapt the number of pages allocated for each list. ARC is scan-resistant.. It has constant-time complexity per request.

IV. BUFFER CACHE MANAGEMENT TECHNIQUES

Buffer Cache Management Techniques are classified into three categories. They are given as below.

Access Pattern Based
This technique focuses on prediction of block access pattern. PCC [22], UBM [23] and DEAR [24] are examples of this technique.

Block Pre-fetching
In this mechanism data blocks are read prior and kept into main memory, to deal with the delay associated with the access made to the disk. This mechanism is termed as pre-fetching. User or complier inserted hints are used in informed pre-fetching. I/O request are traced to obtain the information about the system call made by the applications and used in predictive caching. Automatic Pre-fetching And Caching System (APACS) [31] is the examples of block pre-fetching technique.

Distance Based Prediction
Reuse distance of a block is the time difference between two consecutive references to a block. The reuse distance of a block can be obtained by use of a program counter. Re-Reference Interval Prediction (RRIP) technique has suggested Static RRIP (SRRIP) and Dynamic RRIP (DRRIP), Signature Based Hit Predictor (SHiP) [30] are examples of this technique.

Unified Buffer Management (UBM) [23] scheme exploits reference regularities such as sequential and looping references. Reference information of a block in each file is maintained in abstract form. It is maintained in 4-tuple, such as file descriptor, start block number, end block number, and loop period. This scheme works with the three main modules.

Detection module detects sequential and looping references. A reference is categorized as a sequential reference if any of the block is not re-referenced. If references are referred block set. Each block with history information has a status either LIR or HIR. The cache is divided into a major part and a minor part in terms of size. The major part is used to

Adaptive Replacement Cache (ARC) [15] policy implements two additional lists L1 and L2, L1 keeps track periodically then it is categorized as a looping reference. After the detection, block references are classified into sequential, looping, or other references.

Replacement module applies different replacement schemes to the blocks which belong to different reference patterns. For the partition which holds sequential references, MRU replacement policy is used. For looping references, a period-based replacement scheme is used to replace the victim block in decreasing order of their loop periods, and the blocks having same loop period, MRU [7] block replacement scheme is used. For other references LRU [5], LFU [5], LRU-K [6], and LRFU [14] replacement schemes are applied which replaces on the basis of recency, frequency, or a combination of the two factors.

Allocation module allocates the limited buffer cache space among the three partitions corresponding to sequential, looping, and other references. To allocate the blocks in the cache among the three partitions, marginal gain function is used. Marginal gain is defined as, expected number of extra buffer hits per unit time that would be obtained by increasing the number of allocated buffers from (n-1) to n, where n is the expected number of buffer hits per unit time using buffers.

Robust Adaptive buffer Cache management scheme (RACE) [25] is a novel and simple replacement scheme detects an access pattern. It uses two important data structures, a file hash table and a PC hash table. The file hash table records the sequences of consecutive block references and is updated for each block reference. The sequence is identified by the file description (inode), the starting and ending block numbers, the last access time of the first block, and their looping period. The PC hash table records blocks which are fresh and reused. The main process of the RACE scheme works into three steps. First, the file hash table is updated for each block reference. Second, RACE updates the PC hash table by changing the fresh and reused counters. In last step it predicts access pattern based on values of file and PC hash tables. If the file table reports that the currently requested block has been visited before, a looping pattern is returned. If the file table cannot provide any history information of the current block, RACE relies on the PC hash table to make predictions. A PC with its reused counter larger than its fresh counter is considered to show a looping pattern. On the other hand, a PC is classified as sequential if the PC has referenced a certain amount of one-time-use-only blocks and as others if there is no strong supportive evidence to make a prediction.

Block Pattern Based Buffer Cache Management [26,27] is a methodology which analyzes past access behavior and program context from I/O request and helps to predict block access patterns. It uses data structure which has values such as File hash table and PC hash table of a block. In the process of block access pattern identification three modules are involved. The three modules works as,

Detection module updates File hash table and PC hash table of the respective block as the block is accessed by the I/O request, which is then used to identify the block access pattern. The block being referenced, may be newly referenced or re-referred. So that data structure either enters new values or the existing values are updated. Then the threshold is used to avoid conflicts and the pattern is detected. Allocation Module allocates space to the block being referenced, in the partition corresponding to the pattern identified. The partition space allocated to each identified pattern is calculated and movement of the block among the several partitions, is managed dynamically through the use of marginal gain function. Replacement module replaces block from the partition whenever there is no space to allocate for a block. Based upon the pattern identified by the detection module the replacement policy is applied. Program-Counter based Classification (PCC) [22] is a prediction technique used in pattern based buffer caching. This technique identifies the access pattern among the blocks accessed by I/O operations triggered by a call instruction in the application. Operating system correlates the I/O operations with the program context in which they are issued via the program counters of the call instructions that trigger the I/O requests. PCC also performs classification more quickly as per-PC pattern just needs to be learned once. Detection based Adaptive Replacement (DEAR) [24] buffer management scheme detects the block reference pattern of applications and classifies the reference pattern as sequential, looping, temporally clustered, or probabilistic. After detection, the scheme applies an appropriate replacement policy to the application. In this technique two attributes associated with blocks such as frequency and backward distance are used. Backward distance is defined as the time interval between the current time and the time of a last reference. This scheme employs a procedure which invokes periodically and detects the correct reference pattern. The procedure first finds the backward distance and frequency of the blocks and two ordered lists are created one according to backward distance and another according to frequency. Then these two lists are divided into sub list, all are of same size. After that it calculates average forward distance for all the sub lists and checks various conditions to detect correct reference pattern.

V. COMPARISON AMONG CACHE REPLACEMENT POLICIES

Name of algorithm	performance	Access Time	Scan Resistant	Memory Overhead	Parameter used	Adaptive
LRU	Good for working set less in size than cache size[5]	Fast	No[15]	Low	Recency	No[5]
LRU-2 [6]	Better Than LRU	Slow	No	Low	Recency	No
EELRU	Better Than LRU for regular access pattern[12]	Fast	Upto Some Extent[12]	Low	Recency	Yes[12]
LFU	Good[5]	Fast	No	Low	Frequency	No[5]
MRU	Good	Fast	No[7]	Low	Recency	No[7]
FIFO	Poor[5]	Fast	No[15]	Low	_	No[15]
2Q	Poor[16]	Slow	No[13]	Low	Recency	No[13]
ARC [15]	Good	Fast	Yes[15]	High	Recency, Frequency	Yes[15]
LRFU	Good	50 times Slower than LRU & ARC[15]	No	High	CRF(Recency & Frequency)	Yes[14]
Dueling CLOCK	Good	Fast	Yes[19]	High[19]	-	Yes[19]
LIRS	Good	Fast	No	High comparing to LRU[16]	IRR(Inter Reference Recency)	Yes[16]

Hence, by comparing all the above cache replacement policies we can say that LRU policy is simple and easy to implement. It works well except at some situations. Limitations of LRU are, it has a high overhead cost of moving cache blocks into the most recently used position each time a cache block is accessed, does not exploit frequency information of memory accesses and it is prone to cache pollution when a sequence of single-use memory accesses that are larger than the cache size is fetched from memory. In short it is not scan-resistant and thrash-resistant. LRU2 is an improvement over LRU policy, but it does not work well for the blocks having no significant difference in reference frequencies. In addition, LRU-2 has high overhead as each block access requires log(N) operations to manipulate a priority queue, where N is the number of blocks in the cache.

EELRU is again an improvement over LRU to achieve adaptability and sensitivity to access pattern change. It achieves the desired goal but behaves pathological after some extent for loop access pattern. EELRU cannot quickly respond to the changing access patterns. Without spatial or temporal detections

LFU algorithm does not exploit recency information of the block.

OPT policy can not implemented practically, as it requires future reference information of a block and we can not predict the future reference.

FIFO does not record recency information nor does it exploit the frequency of memory accesses and it is known to have lower performance than LRU.

The deficiency of SC is it needs to keep cache blocks moving from the head of the queue to the tail. ARC policy performs very good but it has high space overhead. LRFU combines LRU and LFU, but it is not effective on workload with a looping pattern. Dueling Clock policy has high memory overhead. LIRS uses independent recency events of each block to effectively characterize their references. It achieves simplicity, adaptability but LIRS stack may grow arbitrarily large, and hence, it needs to be required large memory overhead. This policy does not perform well for sequential access pattern.

VI. COMPARISON AMONG BUFFER CACHE MANAGEMENT TECHNIQUES

Name of policy	Access level	Comment
UBM [23]	Files	UBM scheme is very effective in detecting sequential and looping references. It shows substantial performance improvement. Have good responsiveness and stability as files generally tend to have same access pattern. It has no classification accuracy.
PCC [22]	Program counter Call instruction	It is not sensitive to pattern change over an individual file, as its pattern classification decision based on aggregate statistical information.
DEAR [24]	Application	As application exhibits mixture of access patterns, it may fail to detect local patterns, but it can detect global access pattern correctly.
RACE [25]	File and program counter	It overcomes all the limitations of LRU, but its overhead is it requires program counter signatures.
Block Pattern Based Buffer Cache Management [26]	File and program counter	Buffer cache hit ratio is improved by reducing total elapsed time in servicing I/O services. Overhead is it requires program counter signatures.

VII. CONCLUSION

A cache replacement policy is considered as efficient if it is able to exploit any type of reference regularities which improves hit ratio. In this paper several replacement policies and cache management techniques are discussed of which UBM has no classification accuracy, PCC is not sensitive to pattern change, DEAR policy failed to detect local access pattern correctly and some other has memory overhead as they need to store information such as program counter signatures. This leads to high miss ratio and increased response time. As response time is important factor, it must be minimum. Above discussed schemes can be used according to system requirement. According to the comparison, some policies have limitations so they could not perform well for all type of reference regularities because of this performance is degraded. So there is need to develop a policy which perform better than existing one.

REFERENCES

[1] Hou Fang, zhao Yue-long, Hou fang, "A cache management algorithm based on page miss cost", in proceedings of International conference on Information Engineering and computer science, ICIECS, ISBN: **978-1-4244-4994-1** pp. **1-4**, **2009**.

[2] Prof.P. K. Biswas, "Lecture series on digital computer organization," Internet: http://nptel.iitm.ac.in, Sep **2009**.

[3] M. J. Bach, "Operating system the design of the UNIX", Upper Saddle River, NJ, USA: Prentice-Hall, Inc., **1986**.

[4] A. S. Tanenbaum, A. S.Woodhull, "Operating systems design and implementation", Upper Saddle River, NJ, USA: Prentice-Hall, Inc., **1987**.

[5] Donghee Lee, et.al.,."On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies",SIGMETRICS'**99** Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages **134-143**, NY, USA, **1999**.

[6]E. J. O'Neil, P. E. O'Neil, and G. Weikum., "The LRU-K Page Replacement Algorithm for Database Disk Buffering" , ACM Conference on SIGMOD, pg. no. **297–306**, **1993**.

[7] K. So and R. N. Rechtschaffen, "Cache operations by MRU change." IEEE Trans. Computers, vol. **37**, no. **6**, pp. **700–709**, **1988**.

[8] L. A. Belady, "A study of replacement algorithms **for** a virtual-storage computer," IBM Syst. J., vol. **5**, no. **2**, pp. **78-101**, **1966**.

[9] Alfred V. Aho, Jeffrey D. Ullman, et.al., "Principles of Optimal Page Replacement", Journal of ACM, Vol **18**, Issue **1**, Pages **80-93**,Jan **1971**.

[10] R. Turner and H. Levy, "Segmented FIFO Page Replacement", In Proceedings of SIGMETRICS ,**1981**.

[11] A. Dan and D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes", in Proceedings of ACM SIGMETRICS, Boulder, Colorado, United States, pp. **143—152**,**1990**.

[12] Y. Smaragdakis, S. Kaplan, and P. Wilson, "EELRU: simple and effective adaptive page replacement," in Proceedings of ACM SIGMETRICS international conference on Measurement and modeling of computer systems, New York, USA, pg. no. **122–133**, **1999**.

[13] T. Johnson and D. Shasha, "2Q : A Low Overhead High Performance Buffer Management Replacement Algorithm" , In Proceedings of the **20**th International Conference on VLDB, pg. no. **439–450**, **1994**.

[14] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim, "LRFU: A Spectrum of Policies that Subsumes the LRU and LFU Policies", IEEE Transactions on Computers, vol. **50**, Issue no. **12**, pp.**1352-1361**, Dec. **2001**

[15] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST), pg no. **115–130**, Mar **2003**.

[16] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pg. no. **31–42**, June **2002**.

[17] Seon-yeong Park, et.al., "CFLRU: A Replacement Algorithm for Flash Memory", CASES'06, October **23–25**, Seoul, Korea ,**2006**.

[18] LI Zhan-sheng, et.al.,"CRFP: A Novel Adaptive Replacement Policy Combined the LRU and LFU", IEEE 8th International Conference on Computer and Information Technology Workshops, **2008**.

[19] Andhi Janapsatya, Aleksandar Ignjatovi´c, et.al., "Dueling CLOCK: Adaptive Cache Replacement Policy Based on The CLOCK Algorithm", **2010**.

[20] T. Puzak, et.al,"Analysis of cache replacement algorithms," Ph.D. dissertation, Dep. Elec. Comput. Eng., Univ. Massachusetts, Feb.**1985**.

[21] F. Zhou, R. von Behren, and E. Brewer, "AMP: Program context specific buffer caching," in Proceedings of the USENIX Technical Conference, Apr. **2005**

[22] C. Gniady, A. R. Butt, and Y. C. Hu, "Program-counter based pattern classification in buffer caching", in Proceedings of **6**th Symposium on Operating System Design and Implementation ,pg. no. **395–408**, Dec. **2004**.

[23] J. M. Kim, J. Choi, J. Kim, S. H. Noh, S. L.Min, Y. Cho, and C. S. Kim,"A low-overhead, high-performance unified buffer management scheme that exploits sequential and looping references," in **4**th Symposium on Operating System Design and Implementation ,pg. no. **119–134**, Oct. **2000**.

[24] Jongmoo Choiy, Sam H. Nohz, Sang LyulMiny, Yookun Cho, "An Adaptive Block Management Scheme Using On-Line Detection of Block Reference Patterns", International Workshop on Multi-Media Database Management Systems, Proceedings , pg no. **172 – 179**, Dayton, Aug **1998**.

[25] Yifeng Zhu, Hong Jiang,"RACE: A Robust Adaptive Caching Strategy for Buffer Cache", IEEE Transaction on computers, **2007**.

[26] Urmila Shrawankar, Reetu Gupta, "Block Pattern Based Buffer Cache Management", The **8**th International Conference on Computer Science and Education, April **26-28**, Colombo, **2013**.

[27] Reetu Gupta, Urmila Shrawankar, "Managing Buffer Cache by Block Access Pattern", IJCSI International Journal of Computer Science Issues, Vol. **9**, Issue **6**, November **2012**.

[28] P. Cao, et.al., "Implementation and performance of integrated application-controlled file caching, prefetching, and disk scheduling," ACM Transactions on Computer Systems, vol. **14**, Issue **4**, pp. **311–343**, **1996**.

[29] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in Proceedings of the fifteenth ACM symposium on Operating systems principles (SOSP), New York, USA: ACM Press, **1995**.

[30] A. Jaleel, C. Jean, S. C. Steely, "ShiP: Signature Based Hit Predictor for High Performance Caching", ACM International symposium on Computer Architectur, pg **430-431**, **2011**.

[31] Zhiyang Ding,et.al., "An Automatic Prefetching and Caching System", IEEE, **2010**.
[32] Heung Seok Jeon, "Practical Buffer Cache Management Scheme based on Simple Prefetching", IEEE Transactions on Consumer Electronics, Volume.- **52**, Issue - **3**, August **2006**.
[33] G. Keramidas, P. Petoumenou, S. kaxiras, "Cache Replacement Based on Reuse Distance Prediction", Computer Design IEEE International Conference, page no- **245-250**, Oct **2007**.
[34] G. Keramidas, P. Petoumenou, S. kaxiras, " Instruction Based Reuse Distance Prediction for Efficient Cache Management", Pet International Symposium on System, Architecture, Modeling and Simulations, page no. **48-49**, July **2009**.

