New Approach of Inter-Cross: An Efficient Multilevel Cache Management Policy

MS Surbhi R.Khare[1], Mr. Ritesh Shrivastava[2]

[1]CT Dept. PIET Nagpur, India
[2]CSE Dept. ACET Nagpur, India

Abstract— *Cache performance has been critical for large scale systems. Until now, many multilevel cache management policies LRU-K, PROMOTE, DEMOTE have been developed but still there is performance issue. Many approaches have been proposed to reduce the gap between different levels such as hint-based multilevel cache. Some approaches like demote or promote are based on the latest cache history information, which is inadequate for applications where there are regular demote and promote operations occur. The major drawback of these policies is selecting a victim. In this paper, the new multilevel cache replacement policy called Inter-cross is implemented to improve the cache performance of a system. The decision of promotion and demotion is based on the block's previous N-step promotion or demotion history and the size and resident time of the block in the cache. Comparative study between inter-cross and existing multilevel policies shows that, existing keeps track on last K references of the block within a last cache level, while inter-cross keeps track of the information of the last K movements of blocks among all the cache levels. Inter-cross algorithms are designed that can efficiently describe the activeness of any blocks in any cache level. Experimental results show that inter-cross achieves better performance compared to existing multilevel cache replacement policies such as LRU-K, PROMOTE, and DEMOTE under different workloads.*

Keywords— Multilevel cache, hints, Regency, demote, promote, cache performance.

I. INTRODUCTION

Memory today is very inexpensive, and becoming increasingly large. Despite the principle of locality, a cache will not function effectively if its size is many orders of magnitude smaller than the memory it is buffering [1]. A natural solution to this problem is to make caches larger as well, perhaps on the order of megabytes instead of kilobytes. Such a cache may be able to hold a sufficient range of information, but is too large to be managed effectively and accessed quickly. We now need a cache for the cache.

This trend leads to the use of multilevel caches: a small cache on the processor chip, and a larger cache on a separate chip nearby. These are often called the Level 1 (L1) cache and the Level 2 (L2) cache, respectively. Level 1(L1) cache is present in the processor and the Level 2 (L2) cache is present is present on the motherboard.

The performance benefit of cache plays significant role in calculating overall system performance [2]. Though cache memories are more expensive than mass storage memory like hard disk, most computing devices are equipped with a cache [3]. Caching is one of the most important methods to bridge the gap between different systems access speed, and it is widely used in database management systems for storing frequently accessed queries, file systems for storing file allocation table (FAT), disk drives, operating systems, data compression [4] etc. A good caching algorithm can cache frequently used data blocks in the buffer pool efficiently and provide faster access to data and further improve the throughput and reduce the response time [5].

Various read caching algorithms have been proposed over last few decades for example, LRU [6], LRFU [7], and LFU [8]. Most of the work in these algorithms has focused on the single layer of cache that separates the producer and consumer of the data. As the size of cache is very small, it is difficult to keep all the data required by the application into the single level cache. A major problem with these approaches is that it fails to address the problems: when the access pattern of the workload changes, the cache policy doesn't adaptively adjust at the same time [9]. So the solution is handling dynamic change in the cache replacement in response to changes in the access pattern [10]. So it is essential to use multiple layers of cache for better cache performance. All the necessary data is available into the cache. The most recently used data is kept in first level and the least recently used data is kept at last level. In real time systems, data travels through multiple cache layers before reaching to an application. It seemed that the performance of single-level cache replacement policy is very poor when used in multilevel caches. Thus multilayer cache management policies like PROMOTE [11][12] and

DEMOTE, LRU-K [13] have been proposed. Hints [14] are used to identify and manage the data in multi level cache. Hints give the latest history of the block in the cache. According to the necessity, cache blocks are promoted from lower level to upper level or demoted form upper level to lower level on the basis of hints.

Based on different roles in a multilevel cache, hints can be classified into three categories:

•**Demote hints:** There are the flags used to show the promoted data from the upper level. Demote hint requires only few bits of memory.

• **Promote hints:** These are the flags used to show the cache hit data which is promoted from the lower level cache.

• **Application hints**: These are the flags used to show the data information in different applications. Application hints may be static or dynamic and well defined based on experienced functions in various access patterns.

There is a problem in using hints in correctly identifying most important or less important data, and then quickly promoting more important data to the upper level(s) and demoting less important data to the lower level(s).These hints provide just a block's latest hint information at last level, but lack some important hint history, which reflects a block's past movement among various cache levels. Another problem is giving a unified management on demote and promote hints. These hints are managed separately which may bring an incomplete view on a data and an additional management cost.

In this paper, a new cache block replacement policy is proposed for multilevel cache memory. It constitutes the feature of the two policies: PROMOTE [11] and DEMOTE. The simulation results are analysed for performance analysis of this policy by hit ratio and average response time. In the remaining part, design and algorithms of proposed policy in section II. In section III, the simulation results are analysed against existing multilevel cache policies. Section IV concludes the paper.

II. DESIGN AND MODELLING OF INTERCROSS

The main purpose of this design is to improve the overall cache performance from the application point of view by putting more active data closer to the application which is the upper level of cache hierarchy. To achieve this objective, a multilevel cache management policy is implemented that makes the decision whether to promote a data block or demote a data block based on N-step history information known as hints as well as size and recency of the block in the cache. This policy uses the concept of compressed caching [12]. The data is stored in the cache memory in compressed form therefore more data can be stored than the single level cache [4].

It combines two existing policies: DEMOTE /PROMOTE [11]. The replacement decision is based on hints and size and recency based insertion. In this paper, the focus is more general on demote and promote hints to carry additional information of data blocks from the upper level(s) or the lower level(s) [16] .It is assumed that the cache memory has number of cache levels. The size of the cache level goes on increasing. The cache level nearer to processor is smallest in size; the second level is larger than first one and so on up to last level. The problem with compressed cache is fixed sized cache block. If the particular data is to be stored in cache which is having small size than cache block, then the remaining memory space is consumed by that block cannot be used by other application. Therefore cache block [17] in this policy is having variable size.

The selection of the cache block to be replaced with the main memory block is done by considering various factors like number of promotion and demotion, size of the block, and recency of the block in the cache memory [14]. Inter-cross policy uses the combination of two existing cache management policy: PROMOTE and DEMOTE for detecting the number of promotions and demotions of the block [15]. Usually, promotions are more preferred than demotion of the block. Two types of hints [7] are used here: demote and promote hints. It focuses on demote hints to carry additional information of data blocks from the upper level(s) and promote hints to carry additional information of data blocks from the lower level(s).

It is difficult to select a victim by considering size and the recency factor of the block to be replaced [14]. There are four types of blocks while considering these two factors.

1) Small in size and less resident time
2) Small in size and more resident time
3) Large in size and less resident time
4) Large in size and more resident time

It is obvious that type 3 class blocks should be removed from the cache and type 2 class blocks should be kept in cache. The replacement decision on other two types is complex. However, type 1 class blocks can be managed by considering size as an important factor but the difficulty is with type 4 class blocks. They have larger size and more recency value. Large amount of memory space is consumed by such blocks which can be used for other application.

There is another problem in deciding whether the block is small or large. The selection of the threshold value which

decides the small or large block is essential. Selecting large threshold value means re-reference rate is given more weight than the size information, whereas a lower threshold value means size information is given more weight than the re-reference rate [15]. If the value of threshold is decided on the basic of performance of particular application, it is not easy to implement this solution in real systems, because this threshold value varies with every application. In other words, if particular threshold value is best for an application then it can be worst for another application.

A. Inter-cross Modeling

As shown in Figure 1, N-step hint for a random data block, the latest hint is 1st step hint while the oldest hint is Nth step hint. An N-step hint is a sequence which consists of 1st step, 2nd step, 3rd step, and Nth step hints. An Inter-cross technique is proposed in this paper to solve the problem of detecting most active and less active cache block which is solved by using multiple step history hint information. It compares the activeness of data blocks in any cache level and perform unified management on demote and promote hints.

- This cache model consists of N levels L1, L2...Ln. For a random cache level Li, demote hints (denoted by Di, $2 \leq I \leq N+1$) are from the next upper level Li−1 to the current level Li while (Pi, $\leq I \leq N$) delegates promote hints from the next lower level Li+1 to the current level. In this design, each block can be promoted or demoted by one level in a single transaction [1]. Therefore, initially the more important data blocks placed in a higher cache level. This approach focuses on read I/O requests and writes requests can be handled by other separate hints. To record the movements of active data blocks among various cache levels step hint value (SHV) are used.
- N-step Hint Values (NHVs) are used to identify the status of a data block. Based on the all NHV's of any cache level, demotion or promotion policies will be applied when the NHV of a data block is small or large.
- If the block is demoted from upper level cache then its step hint value (SHV) is set to 1 and if promoted to upper level cache then its SHV is set to 0 in NHV.

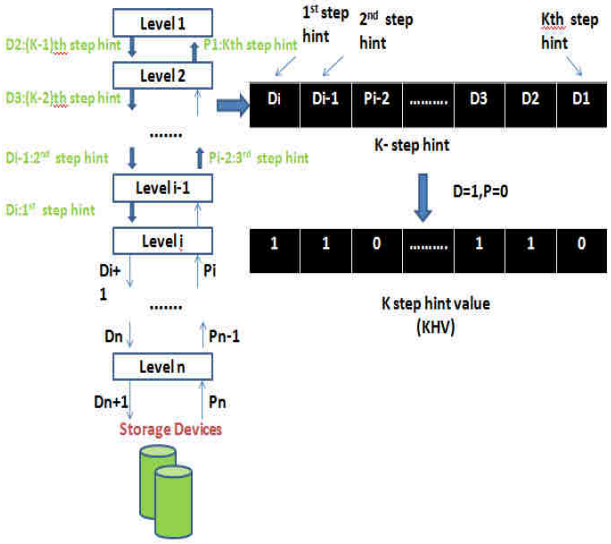


Fig. 1:N-step Hint.

$$NHV = \sum_{n=0}^{N} \mathrm{SHV}(n) \ldots\ldots\ldots (1)$$

The promotion condition of the block depends upon activeness if the block. It is determined by calculating its N-step hint value (NHV).The NHV is the sum of all SHV at a particular level. The block having maximum NHV is more active and selected for promotion and the block with minimum NHV is less active and selected for demotion. Equation 1 can easily identify the most active blocks in a random level *L* for promotion and the least active blocks for demotion by checking the NHVs.

B. Intercross Algorithm

For each level in this multilevel cache design, N-step hints are added into single level cache algorithms, which can be any existing cache algorithms. For testing purpose, Hint-N [1] algorithms are used to describe the interaction among cache levels while using LRU [17] to characterize a block within a specific level. Hint-N algorithms are developed, which have the following process on NHV's and two policies to decide whether a data block should be demoted or promoted. Here, the Least Recently Used policy is used to select a victim LRU list for demotion if required.

This policy constitutes three algorithms:

A. Initialization

B. Promotion

C. Demotion

- ***Initialization and Update of NHVs***

The following rules are used to initialize and update NHVs in the Hint-N algorithms as shown in Algorithm.

1) Initially all the levels of the cache are empty, KHV will be set to 0.

2) If the block is not present in cache then put all of its NHV's to NULL.
3) If the block is in cache memory, then update its NHV according to its latest hint information.

- ***Promotion Policy***

When a data block in L_i becomes more active than the least active block in L_{i-1}, it should be promoted. If there is a read request to a block which is not present in the upper level cache then:

1) Calculate minimum NHV (NHV min (i)) for the level (Li) and send this value to next lower level cache (L_{i-1}).
2) In the next lower level cache, check whether the block is present or not otherwise repeat step 1.
3) If the block is present then check whether it satisfies the promotion condition:

$$NHV(block) \geq \sum_{n=}^{N} NHV(n) \ for\ all\ cache\ blocks$$

If YES:

a) If there is space in upper level cache, promote the block to upper level and update its NHV.
b) If upper level cache is full then Replace this block with the block which have largest size and lowest locality of reference and which is present at the bottom of LRU [19] list and update NHV of both the blocks.

If NO, go to step 1.

4) Else fetch the block from main memory [6]. In this way, promotion takes place.

- ***Demotion Policy***

If there is read request to a block from processor and is need to replace a block in upper level with the required block, then the demote algorithm works.

1) Calculate minimum NHV (NHV min) for that level.
2) Check how many blocks having their NHV equal to minimum NHV.
3) Among these blocks, select a block which is at the bottom of LRU list.
4) Check whether the block belongs to which type
a) If the block belongs to type *1*, go to step *e*.
b) If the block belongs to type *3*, go to step *5*.
c) If the block belongs to type *2*, go to step *e*.
d) If the block belongs to type *4*, check the block for threshold. If the block is smaller than threshold value, go to step e
e) Select the block which is above the previous block in LRU list. Go to step 4.
5) Replace the requisite block with the current block and update NHV's of both the blocks.

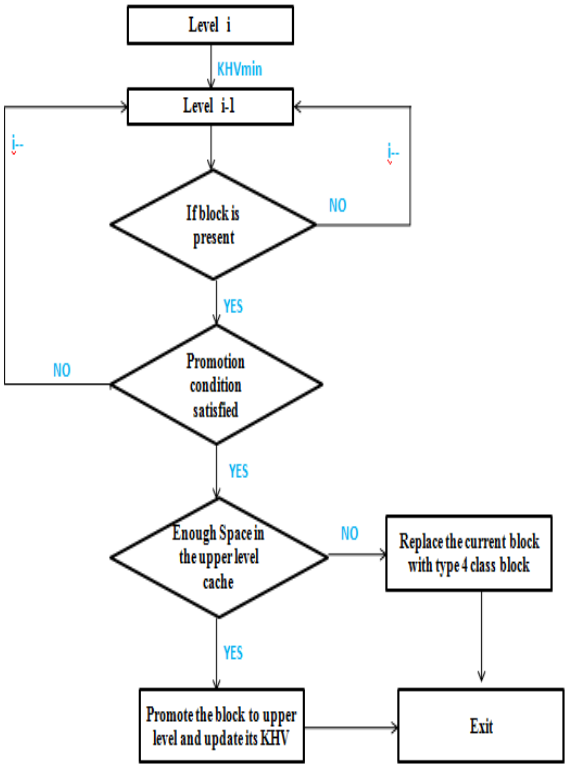


Fig.2: Flow of Promotion Policy

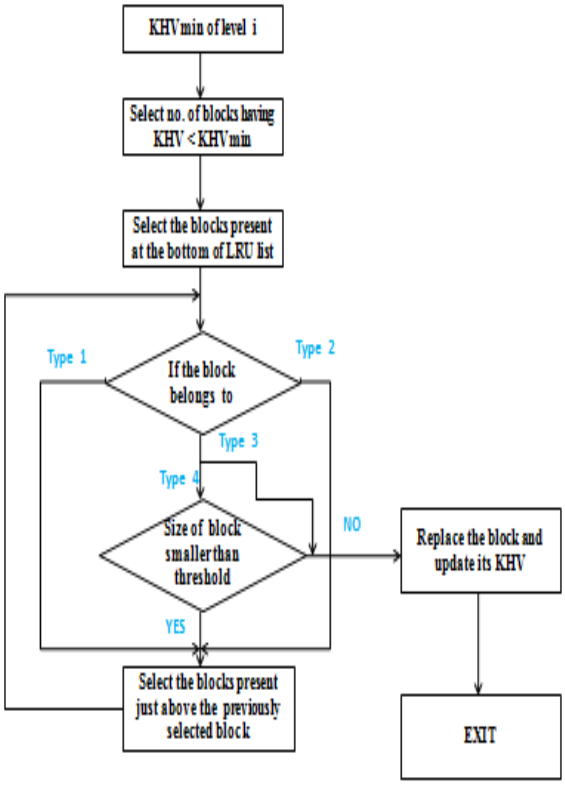


Fig.3:Flow of Demotion Policy

In this way, this inter-cross algorithm gives better cache performance than other cache management policy [20]. The above algorithm can be simulated on cache simulator

to see the performance. It shows better hit ratio than other policies.

III. SIMULATION RESULTS AND ANALYSIS

To verify the effectiveness of Inter-cross policy, trace driven simulation are used to evaluate the Inter-cross algorithms and compare them with three popular multilevel cache like FIFO [21], LRU-K [22], DEMOTE [11], PROMOTE [20] under different workloads as shown in Figure 3.

A. Simulation Methodology

The new .net based simulator is developed, *hybridcachesim*. Within each cache level, LRU policy is used. The experiment results show the % hit ratio of the Inter-cross against existing multilevel cache policies. Four different traces are used in this simulation as shown in Figure 4:

1) Load of 1000 instruction, each having maximum frequency of 20 occurrences.
2) Load of 1500 instruction, each having maximum frequency of 15 occurrences.

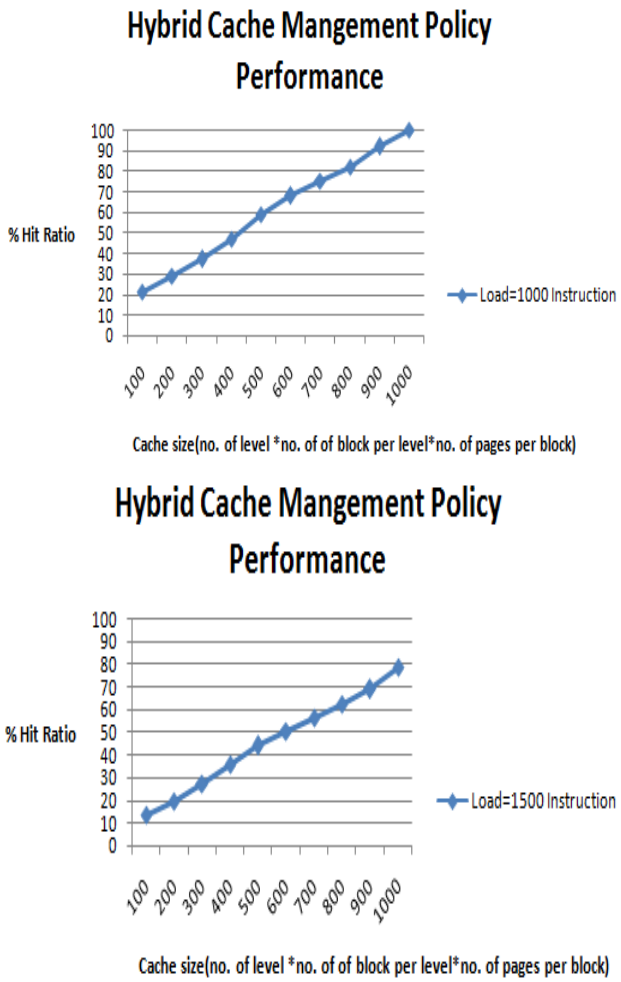


Fig.4:Performance of Inter-cross policy under different workload

Inter-cross approach is concerned with the applications which have many blocks active among different cache levels, so the system is set to warm-up in hybridcachesim so that the enough data blocks have been flooded to all cache levels [19]. Unless stated, the default parameters used are: two cache levels (n = 3), number of blocks per levels (10), and block size (Sb=*10* KB). The aggregate cache size is the product of all cache level, number of blocks per level, and block size. Write requests are ignored in this simulation. Based on the default settings of *hybridcachesim*, the average access times of cache and disk are 0.25ms and 10ms, respectively.

B. Results and Analysis

1) Number of Demote/Promote operations

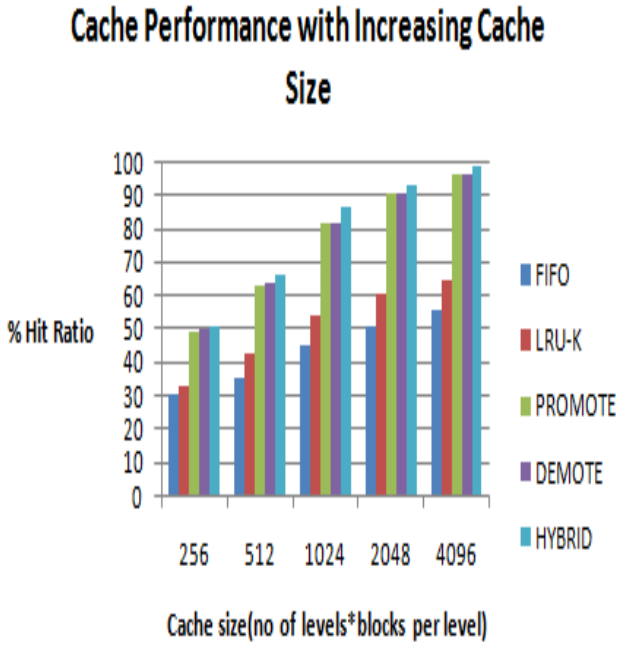
Experimental results shows that the number of promote/demote operations decreases with increased cache size because the possibility of replacement of a block is decreased when the cache becomes larger. The number of operations also decreases with increased block size when the number of blocks is reduced. It is observed that Inter-cross reduces up to 15% of Demote/Promote operations compared to the PROMOTE [20][5] algorithm. This prevents unnecessary movement of cache blocks among different cache levels by keeping the data at the most appropriate level.

2) Aggregate hit ratio of different multilevel cache policies

Next, the aggregate hit ratios of different multilevel cache management policies are calculated.

$$\% \text{ Hit Ratio} = (\text{No.of Cache hits} * \text{Total no.of request}) * 100 \ldots \quad (3)$$

$$\% \text{ Miss Ratio} = (\text{No.of Cache miss} * \text{Total no.of request}) * 100 \ldots \quad (4)$$



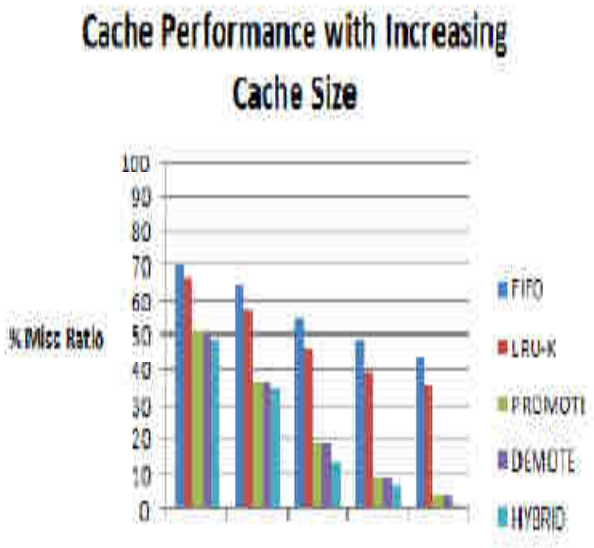


Fig.5: Comparison of performance of Inter-Cross policy with different multilevel

The results are shown in Table I. PROMOTE and DEMOTE [1] policies gives same hit ratio. It is observed that if number of cache levels going on increasing, the aggregate hit ratio increase. Up to four levels ($N \leq 4$), the performance benefit is more as compared to cache overhead problems like cache coherence. But after the number of levels reaches to five, the aggregate hit ration remains constant and there is increase in the overhead in maintaining the large cache as shown in the Figure 2. From these results, Inter-cross achieves higher aggregate hit ratio than existing multilevel cache policy.

3) Average response time of different multilevel cache policies

The average response time and average hit ratio of multi level cache replacement policy is higher than those for single level. It is a simple and efficient approach to handle demotes and promotes hints. When more hint information is used, a better decision can be made whether to demote or promote a block. When the number of cache increases above four the access overhead of cache table increases and the performance degrades.

TABLE III POLICIES COMPARISON ON THE BASIS OF DIFFERENT WORKLOAD

Cache Size(Kilo bytes)	% Hit ratio			
	LRU-K	PROMOTE	DEMOTE	INTERCROSS
256	32.83	49.45	49.60	50.80
512	42.47	63.25	63.46	65.79
1024	53.65	81.45	81.46	86.19
2048	60.70	90.52	90.64	92.64
4096	64.63	96.12	96.32	98.75

IV. CONCLUSION

In this paper Inter-cross multilevel cache management policy is implemented, to keep track of last N-step history information about the movement of a data block among multiple cache levels. The activeness of a block is determined by the frequency of the demote/promote operations of a block, size and resident time of the block at the particular cache level. Inter-cross promotes active data to the upper cache level while demotes passive data to the lower level more efficiently. An Inter-cross model is developed to easily identify the activeness of blocks. The results show that Inter-cross achieves better performance compared to FIFO, LRU-K DEMOTES and PROMOTE algorithms under different I/O workloads.

REFERENCES

[1] Chentao Wu, Xubin He, Qiang Cao, Changsheng Xie, and Shenggang Wan.” Crossbreed: An Efficient Multilevel Cache Using N-step Hints,”IEEE Transactions on Parallel and Distributed Systems, Mar 2013.

[2] U. Shrawankar and R. Gupta, “Block Pattern Based Buffer Cache Management.”In 8th International Conference on Computer Science & Education (ICCSE 2013.

[3] B. Gill, M. Ko, B. Debnath, and W. Belluomini, “STOW: A spatially and temporally optimized write caching algorithm.” In Proc. of the 2009 USENIX Annual Technical Conf., San Diego, CA, June 2009.

[4] Yadgar, M. Factor, and A. Schuster, Karma: “Know-it-all replacement for a multilevel cache.” In Proc. of the 5th USENIX Conf. on File and Storage Technologies, San Jose, CA, February 2007.

[5] Lampson, “Hints for Computer System Design,” Proc. Ninth ACM Symp. Operating System Principles, Oct. 1983

[6] R. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, “Informed Prefetching and Caching,” Proc. 15th ACM Symp Operating System Principles, Dec. 1995.

[7] B. Gill. “Systems and methods for multi-level exclusive caching using hints”. US Patent No. 7761664 B2, July 2010.

[8] B. Gill, “On multi-level exclusive caching: Offline optimality and why promotions are better than demotions”. In Proc. of the6th USENIX Conf. on

File and Storage Technologies, San Jose, CA,February 2008.

[9] Yadgar, M. Factor, K. Li, and A. Schuster, "Management of multilevel, multiclient cache hierarchies with application hints."ACM Transactions on Computer Systems, 29(2): Article 5, 2011.

[10] Y. Zhu and H. Jiang, RACE: "A robust adaptive caching strategy for buffer cache. IEEE Transactions on Computers, 57(1):25–40, 2007.

[11] k, Chikhale,U.Shrwankar,"Hybridmulti-level cache management policy", IEEE conference on communication systems and network topologies, 978-1-4799-3070, march 2014.

[12] Bairavasundaram, M. Sivathanu, A. Arpaci-Dusseau, and R.Arpaci-Dusseau, "X-RAY: A Non-Invasive Exclusive Caching Mechanism for RAIDs," Proc. 31th Ann. Int'l Symp. Computer Architecture, June 2004.

[13] Wu, X. He, Q. Cao, and C. Xie, "Hint-K: An Efficient Multi-Level Cache Using K-Step Hints," Proc. 39th Int'l Conf. Parallel Processing, Sept. 2010.

[14] G. Yadgar, M. Factor, K. Li, and A. Schuster, "Management of Multilevel, Multiclient Cache Hierarchies with Application Hints," ACM Trans. Computer Systems, vol. 29, no. 2, Article 5,2011.

[15] G. Yadgar, M. Factor, and A. Schuster, "Karma: Know-it-All Replacement for a Multilevel Cache," Proc. Fifth USENIX Conf. File and Storage Technologies, Feb. 2007.

[16] Zhou, B. Behren, and E. Brewer, "AMP: Program Context Specific Buffer Caching," Proc. USENIX Ann. Technical Conf., Apr. 2005.

[17] Y. Zhou, Z. Chen, and K. Li, "Second-Level Buffer Cache Management," IEEE Trans. Parallel and Distributed Systems, vol. 15, no. 6, pp. 505-519, June 2004.

[18] S. Jiang and X. Zhang, "LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance," Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems, June 2002.

[19] S. Jiang and X. Zhang, "ULC: A File Block Placement and Replacement Protocol to Effectively Exploit Hierarchical Locality in Multi-Level Buffer Caches," Proc. 24th Int'l Conf. Distributed Computing Systems, Mar. 2004.

[20] J. Robinson and M. Devarakonda, "Data Cache Management Using Frequency-Based Replacement," Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems, May 1990.

[21] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," Proc. 20th Int'l Conf. Very Large Databases, Sept. 1994.

[22] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim, "LRFU: A Spectrum Of Policies That Subsumes the Least Recently Used and Least Frequently Used Polices," IEEE Trans. Computers, vol. 50, no. 12, pp. 1352-1361, Dec. 2001.