DOI 10.29042/2018-4530-4534

Helix Vol. 8(6): 4530- 4534

# A Multilevel Cache Management Policy for Performance Improvement in Distributed System

[1]Manish Motghare, [2]Dr. Preeti Voditel

[1,2] Department of Computer Application. Shri Ramdeobaba College of Engineering and Management, Nagpur, India

*Email: motgharemm@rknec.edu, voditelps2@rknec.edu*

**Received: 20th September 2018, Accepted: 11th October 2018, Published: 31st October 2018**

## Abstract

Caching improves the performance of a system by storing most used recently or frequently accessed data in an upper layer of the multilevel cache hierarchy, results in reducing the delay in I/O system. In parallel and distributed systems, blocks are placed in multilevel cache hierarchy for faster execution of the application. There are many policies; including Hint (information) based which try to improve the gap between various levels of a cache hierarchy in multilevel cache architecture. This information based policy i.e. HINT works on the basis of a number of recent promote and demote operations of cache blocks, i.e. it stores the latest information about the cache block and performs the shifting of cache blocks based on promote and demote operations in various levels of the cache hierarchy. This policy works well for the standalone application, where a number of promote and demote operations occurs rarely. But in a distributed or parallel system, this kind of policy is insufficient because of frequent promote and demote operations. In this paper, we proposed a novel multilevel cache management policy, which keeps a track record of the number of promote and demote operations took place with each cache block for promotion and demotion in a multilevel cache hierarchy. Apart from hint history promote and demote operation, we considered three more parameters for replacement of cache block from upper level to lower level and vice-versa. The proposed policy is able to identify the hot and cold data effectively. The Simulations results show that our proposed policy achieves better performance compared to existing well-known policies like LRU-K, MQ (Multi-Queue) and 2Q.

## Keywords

*Multilevel Cache, Hints, Recency (Most Recently Used), Frequency, Demote, Promote, Cache Performance*

## Introduction

Cache memory is the fastest memory and more expensive than the others. It consists of various levels from L1 to Ln depending upon the system architecture. In distributed multilevel cache hierarchy, upper layer serves as a cache for lower layer cache architecture. The caching architecture includes L1, L2 cache and translation schemes. The upper layer cache i.e. L1 is the closest to the processor or sometimes inbuilt with the processor. The L2 cache, generally larger than the primary cache and placed it as a separate chip. In a multilevel cache architecture, cache blocks move from upper level to lower level and vice-versa depending upon the access pattern of the application. In order to manage these data, Hints are used to identify cold and hot data for improving the performance of the system [1][2]. Based on the different information stored by the Hints, they are classified into three following categories [3][4].

- *Demotion Hints:* The demotion hints are commonly known as demote hints, contains flags which occupies only a few bits. The demote hints shows the information about cache block which is shifted from upper level cache i.e. L1 to lower level L2. The information about this hint is explained in Demote and so on [5][6][7][8].
- *Promotion Hints:* The promotion hints are also known as promote hints, saves information of those blocks which are updated from lower level cache to upper level. The promote policy was mentioned in the Promote policy [16] and Multiple clients Multiple caches [17].
- *Application Hints:* These flags are used to store the various information about the active cache blocks related to various applications. Some of them are static as mentioned [18][19] and rest are dynamic which keep track of the cache blocks based on hints history information [20][21].

Though multilevel cache architecture is good, but it comes up with some disadvantage like, in multilevel cache management policies, it consumes two different positions for the same cache objects which results in wastage of memory. The main problem with multilevel cache architecture is that, for the same data, we require two separate levels of cache which results in memory wastage. The three important parameters which are considered in our proposed reputation algorithm are frequency, recency (most recently used), and size of the cache block along with number of promote and demote operations with each cache block for replacement and shifting. The rest of this paper is organized as follows: section 2 gives a brief introduction of the Reputation policy along with our contribution. The details of the simulator, results, and analysis are mentioned in section 3. Finally, section 4 concludes the work presented in this paper.

 

## Motivation and Contributions

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 [9], [10], [11], and [12]. Consider the architecture of two-level caches L1 and L2 as shown in figure 1. In this we compare activeness of four data blocks based on their shifting movement from low to high and high to low-level cache. The movement of a block from low to high level is called the promotion of cache block and movement from high level to low level is called demotion. Block A went through 2 times promote operation and single demote operation and Block B came across only single promote operation. Activeness of any data block is measured on the basis of time spent by data block in the cache memory or current status of presence in the cache. In figure 1, both blocks A and B both is active block because both are present in the L1 cache. But who is more active among them requires complete information to promoting and demote hints [11], [14]. Same fundamental applies with data block C and D. As per the definition of Activeness or Hot data, the block which is having more resident time in a cache memory is said to be active or considered as hot data. But in this case, both A and B as well C and D is active based on latest step hints operation, but finding out who is more active block based on current N-Step history is the problem. The second issue which needs to be addressed is that how to give unified management on promote and demote hints, as both the hints are managed separately in previous research [11], [14], so it becomes difficult to take a decision. The third issue is considering the size of the data block during the promotion and demotion of the data block which is never considered in previous research work. To overcome the drawbacks of the previously mentioned policy, we propose, a new cache block replacement policy for multilevel cache memory, this policy is suitable for parallel and distributed where rapidly promote and demote operation takes place. Our policy considered both promote hint and demote hints history information for shifting of the data block in multilevel cache architecture. When compared to LRU-K [13], which keeps track of only last movement of the cache block in single level cache architecture, REPUTATION policy keeps track of the entire data block in different cache architecture by considering the size during placing of the data block in first level cache i.e. L1. This makes it a unique cache replacement policy which works well for real-time systems where frequently data block is promoted or demoted.

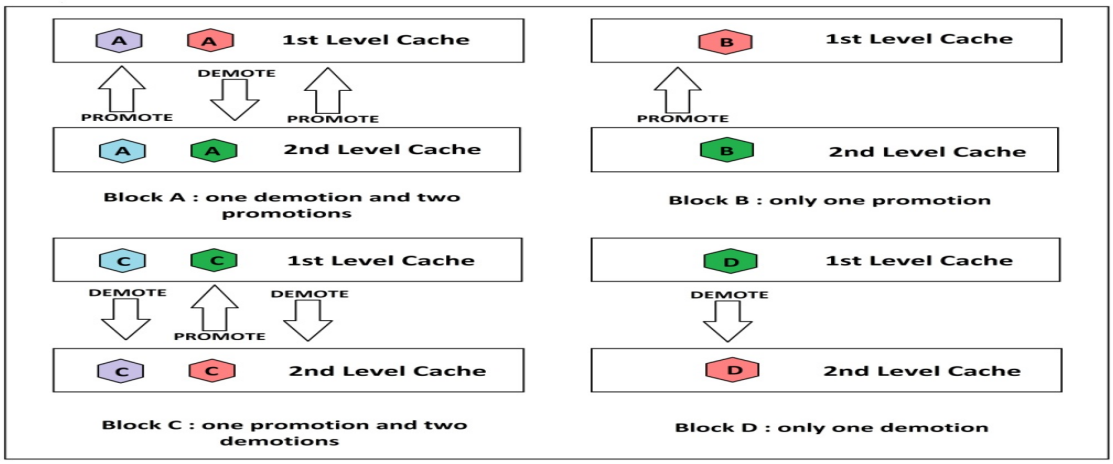


**Figure 1: N-Step Hint Information of Data Blocks**

Our contribution includes: first solve the problem of detecting hot or cold data which is identified using multiple step hints. Second, a unified multilevel cache management policy which considers three parameters for shifting of the cache block in different cache levels i.e. promote hint history; demote hint history and size of the data block. Reputation Cache replacement policy identifies the victim cache block and performs the replacement with the required cache block requested by the application. The policy is simulated using Repcachesim simulator. The cache memory is partitioned, where each level holds a number of cache blocks. To utilize the cache effectively and efficiently multiple parameters are considered for replacement of data block. Reputation aims to overcome the difficulties exhibited by other multilevel cache and single level cache replacement policies.

## Results and Analysis

This section presents the performance evaluation of Reputation policy through a trace-driven simulation. The performance of a Reputation policy with other multilevel cache management policies as LRU, LFRU, FBR, MQ, and 2Q is made. The

 

evaluation is performed on different workload under varying cache sizes. The results obtained from the comparison are discussed in the following section. The Reputation cache management policy is better to fully utilize the cache memory. Out of the various existing multilevel cache management policies, this policy differentiates the selection of a victim cache block by using various parameters while taking the replacement decision as mentioned above. The number of cache blocks identified as a victim for updating or shifting arranged in LRU queue. The block presents at the bottom of the LRU list is selected as a victim first. If that block is not suitable for updating, then the block just above the previously selected block is selected for updating or demoting and so on. The measure of hit ratio and average response time is used as the primary metric in the performance comparison.

Experimental Setup: Repcachesim simulator is a modified simulator, developed to simulate multilevel cache management policy. The simulator has been widely used by researchers [15] to implement various cache policies. Promotion based single mode cache management policy has been employed. Promotion and Demotion based Dual mode cache policy has been employed. The Hashmap data structure is used to implement the cache. The simulation is performed on the system having 4GB DDR3 RAM, AMDE1 1.7GHz Dual-core processor, and Linux-mint 18.2 operating system. The following parameter is used for simulation.

i. The number of Cache Levels.
ii. Page sizes in bytes.
iii. Aggregate Cache Size (MB).
iv. Frequency and recency of each cache block.
v. A Number of an application running simultaneously.
vi. Number of cache block on various cache levels.
vii. The number of an application running at a time.
viii. The number of data block related to particular application present in the cache block.
ix. The demotions or promotion applied to cache blocks.
x. The total average hit ratio.
xi. The total number of cache hits.
xii. The total number of cache misses.
xiii. Average response time (ms)

The collective hit and miss ratio of present multilevel cache management policy is calculated by using this formula mentioned below:

**Percentage Miss Ratio = (Number of cache miss * Total number of request) * 100**

**Percentage Hit Ratio = (Number of cache hits * Total number of request) * 100**

The results which are shown in Multi-Queue and 2Queue etc. policies give same hit ratio. It is also observed that if the number of the cache levels goes on increasing, the collective hit ratio will increase. Up to four levels ($N \leq 3$), 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 an increase in the overhead in maintaining the large cache.

The Table 1 mentioned below shows a comparison between various available cache management policies which are in current use for distributed and parallel system. Hit ratio under varying cache sizes is given in Table I. Compared to other multilevel cache management policies; Reputation achieves a satisfactory high hit ratio. Use of reference recency and block size information identifies the victim accurately. Multiple cache levels help in achieving a better hit ratio. As the cache block is available on request the time spent in performing the I/O and execution is reduced. Thus system performance is enhanced by adapting to different applications and to the different behavior observed within a single application.

 

| Cache Size | 2MB | 4MB | 8MB | 16MB | 32MB | 64MB | 128MB | 256MB | 512MB | 1024MB |
|---|---|---|---|---|---|---|---|---|---|---|
| LRU | 3.7 | 15.5 | 43.3 | 57.3 | 62.5 | 67 | 71.7 | 77.5 | 82.4 | 87.8 |
| LFRU | 16.4 | 31.2 | 48.4 | 57.2 | 62.1 | 66.5 | 71.2 | 77 | 82.2 | 87.6 |
| FBR | 8.6 | 20.1 | 44.6 | 60.4 | 65.7 | 69.4 | 71.6 | 79.2 | 84.5 | 88.6 |
| 2Q | 17.1 | 32.7 | 50.6 | 60.4 | 65.7 | 69.4 | 73.6 | 78.9 | 84.1 | 88.5 |
| MQ | 22.1 | 36.5 | 55.5 | 63.1 | 6735 | 72.1 | 76.8 | 81.3 | 85.8 | 89.5 |
| REPUTATION | 23.4 | 37.7 | 56 | 64.6 | 68.1 | 73.1 | 76.6 | 81.8 | 86 | 89.2 |

**Table 1. Performance Evaluation of Various Cache Management Policies with Reputation Policy**

**Conclusion**

The proposed policy efficiently performs the cache management by replacing the most accurate cache block with the victim cache block. The decision of which cache blocks to be replaced is taken by considering the frequency, recency and size of the block hence the replacement is more accurate. This policy shows improved performance compared to LRU-K, MQ (Multi-Queue) and 2Q by improving the hit ratio, and reducing the time spent in performing the I/O and overall execution. This policy differs from other multilevel cache management policies as it considers the latest cache block information of all the cache levels while making replacement decision. It also considers the parameters like size and resident time which are most essential for the correct block to be replaced with total cache utilization. LRU-K, 2Q, and MQ (Multi-Queue) do not consider these parameters. The concept of compressed caching is used in this policy which is not used by other multilevel cache management policies. This helps in saving essential cache space. In this policy cache replacement decision is based on the frequency of data block, size, and recency. Use of various parameters in making replacement decision helps in increases the hit ratio of each individual sub-cache partition which in turn improves the overall hit ratio. This leads to effective and efficient utilization of the available cache memory space.

**References**

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

[2] R. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, “Informed Prefetching and Caching,” Proc. 1 5th ACM Symp. Operating System Principles, Dec. 1995.

[3] P. Sarkar and J. Hartman, “Efficient Cooperative Caching Using Hints,” Proc. Second USENIX Symp. Operating Systems Design and Implementation, Oct. 1996.

[4] P. Sarkar and J. Hartman, “Hint-Based Cooperative Caching,” ACM Trans. Computer Systems, vol. 18, no. 4, pp. 387-419, 2000.

[5] T. Wong and J. Wilkes, “My Cache or Yours? Making Storage More Exclusive,” Proc. USENIX Ann. Technical Conf., June 2002.

[6] Z. Chen, Y. Zhou, and K. Li, “Eviction Based Cache Placement for Storage Caches,” Proc. USENIX Ann. Technical Conf., June 2003.

[7] 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.

[8] L. 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.

[9] G.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.

[10]L. Bairavasundaram; M. Sivathanu; A. C. Arpaci-Dusseau; R. H. Arpaci-Dusseau, “X-RAY: A Non-Invasive Exclusive Caching Mechanism for RAIDs,” 31st Annual International Symposium on Computer Architecture, 176-187, 2004.

[11] T. Wong and J. Wilkes, “My cache or yours? Making storage more exclusive”, in proc. of the 2002 USENIX Annual Technical Conf., Monetary, CA, June 2002.

[12] Y. Zhou, J.Philbin, and K. Li, “Second-level buffer cache management”, IEEE transactions on Parallel and Distributed Systems, 15(6): 505-519, 2004.

 

[13] B. Gill, “On multilevel exclusive caching: offline optimality and why promotions are better than demotions”, In Proc. Of the 6th USENIX Conf. on File and Storage Technologies, San Jose, CA, February 2008.
[14] E. O’Neil, P. O’Neil, and G. Weikum, “The LRU-K Page Replacement Algorithm for Database Disk Buffering,” Proc. ACM SIGMOD Int’l Conf. Management of Data, New York, NY, USA: ACM Press, 1993, pp. 297–306. May 1993.
[15] T. Wong and J. Wilkes, “My cache or yours? Making storage more exclusive”, In Proc. of the 2002 USENIX Annual Technical Conf., Monterey, CA, June 2002.
[16] B. Gill, “On Multi-Level Exclusive Caching: Offline Optimality and Why Promotions are Better than Demotions,” Proc. Sixth USENIX Conf. File and Storage Technologies, Feb. 2008.
[17] G. Yadgar, M. Factor, K. Li, and A. Schuster, “MC2: Multiple Clients on a Multilevel Cache,” Proc. 28th Int’l Conf. Distributed Computing Systems, June 2008.
[18] 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.
[19] X. Li, A. Aboulnaga, K. Salem, A. Sachedina, and S. Gao, “Second-Tier Cache Management Using Write Hints,” Proc. Fourth USENIX Conf. File and Storage Technologies, Dec. 2005.
[20] X. Liu, A. Aboulnaga, K. Salem, and X. Li, “CLIC: Client-Informed Caching for Storage Servers,” Proc. Seventh USENIX Conf. File and Storage Technologies, Feb. 2009.
[21] 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.

 