Research Article Impact Factor: 4.226 ISSN: 2319-507X
Ashwini R. Ghule, IJPRET, 2016; Volume 4 (9): 227-237 IJPRET

INTERNATIONAL JOURNAL OF PURE AND APPLIED RESEARCH IN ENGINEERING AND TECHNOLOGY

A PATH FOR HORIZING YOUR INNOVATIVE WORK

MULTI-ATTRIBUTE BASED CACHE MANAGEMENT POLICY FOR PERFORMANCE IMPROVEMENT

MS. ASHWINI R. GHULE[1], PROF. GAURAV D. GULHANE[2], DR. H. R. DESHMUKH[2]

1. PG Scholar, Department of Computer Science & Engineering, Dr. Rajendra Gode Institute of Technology & Research, Amravati.
2. Department of Computer Science & Engineering, Dr. Rajendra Gode Institute of Technology & Research, Amravati.

Accepted Date: 15/03/2016; Published Date: 01/05/2016

Abstract: Parallel or distributed system is famous for their performance. There is a great demand in multilevel cache over a single cache to improve the performance of an I/O system because of its efficiency. Many policies of multilevel cache are present still there is a performance issue. 1)These policies fail to select a victim for replacement i.e. fail to select a cache block optimally and 2)increases redundancy causes cache pollution at lower level. These policies include DEMOTE, LFU, PROMOTE, LRU-K, Multi Queue (MQ). In this paper we introduced Multi-Attribute cache management policy which will address drawbacks of above defined policies by considering three factors together to update or replace cache block in a multilevel cache hierarchy, secondly it prevents cache pollution at lower level cache. First factor is frequency i.e. How repeatedly promotion and demotion of cache block takes place in a cache second is recency of object in a cache i.e. how nearly object is used. Third is considering object size, the object with less recency, largest block size and less frequency will be evicted first. This policy is capable of selecting a cache block that is victim efficiently and thus gives higher hit ratio compared to other present multilevel cache policies.

Keywords: Hints**,** Multilevel Cache, Demote, Recency, Promote, Cache Performance

Corresponding Author: MS. ASHWINI R. GHULE

Access Online On:

www.ijpret.com

How to Cite This Article:

Ashwini R. Ghule, IJPRET, 2016; Volume 4 (9): 227-237

PAPER-QR CODE

INTRODUCTION

Multilevel cache is using more than one level of cache implementation in order to make the speed of cache access almost equal to the speed of the CPU and to hold a large number of cache objects. Based on the content provided in the both the level of the cache it can be classified into two major categories. Regardless of 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. 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 [3]. 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.

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.

A. 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.

Multiple hints: These are the flags used to show the data information in different applications. Multiple 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 multiple level cache memory. It constitutes the feature of the two policies: PROMOTE [11] and DEMOTE. The simulation results are analyzed 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 analyzed against existing multiple level cache policies. Section IV concludes the paper.

II. DESIGN AND MODEL OF MULTIPLE LEVEL METHOD

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 multiple level 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). 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. Multiple Level Method

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 ≤ I ≤ N+1) are from the next upper level Li−1 to the current level Li while (Pi, ≤ I ≤ 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.

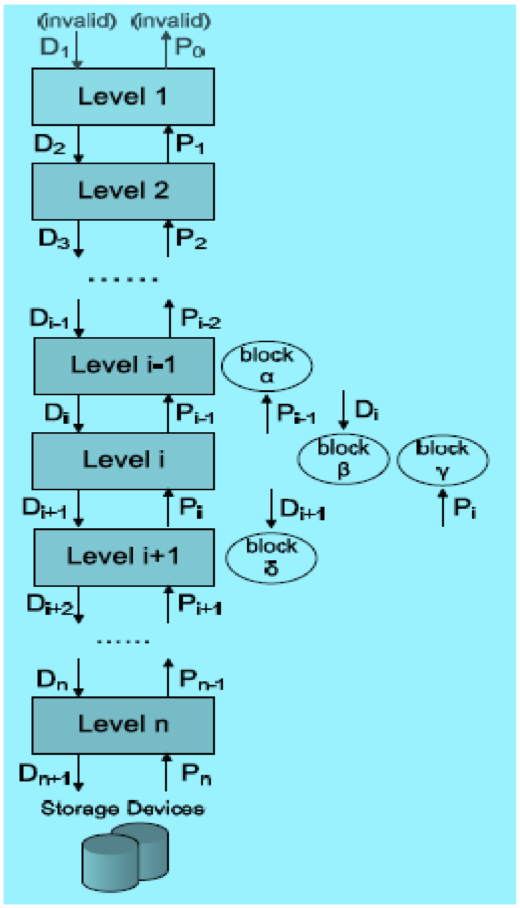


Fig. 1: N-step hint for a multiple level approach.

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. For each level in this multiple level 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.

III. RESULT DISCUSSION

In this section we discuss performance result of multiple level and compare it with other multilevel cache management policies. The performance of the policy is evaluated based on three parameters:

- Hit ratio
- Total Execution Time
- Time Elapsed in I/O.

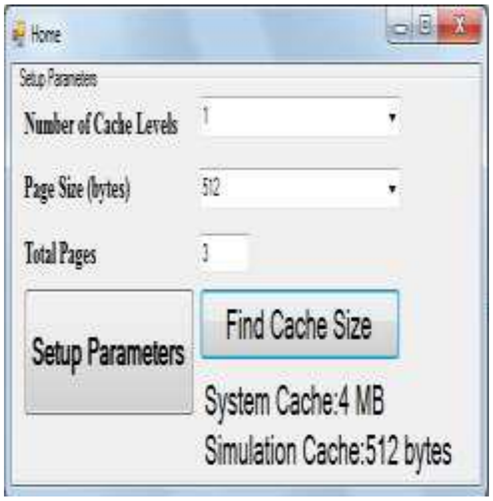
This section presents the performance evaluation of multiple level through a trace-driven simulation. The performance of multiple level with other multilevel cache management policies as LRU-K, MQ, and 2Q is made. The evaluation is performed on different workload under varying cache sizes. The results obtained from the comparison are discussed in the following section.

A. 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.

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

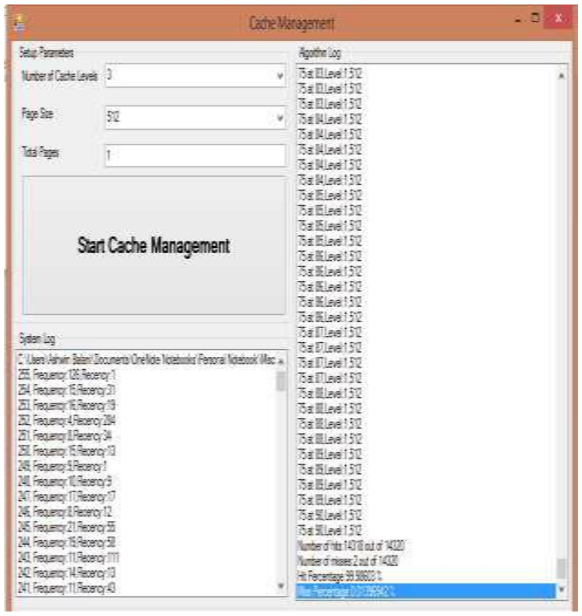
2) Load of 1500 instruction, each having maximum frequency of 15 occurrences.



Number of Cache Levels 1
Page Size (bytes) 512
Total Pages 3
System Cache:4 MB
Simulation Cache:512 bytes

Fig. 2: Output Screen showing Parameters and System Cache Memory Size.

As shown in the above diagram, this is the first output screen showing the parameters like number of cache level, page size in bytes and the total Pages. User will set these parameters according to their need. After clicking on the Find cache Size Button, we get the System cache Size. As shown above, the Size is 4MB, and the Simulation cache Size will be 512 bytes. We can the Number of cache level up to 3 as we divide the cache into 3 level.



Number of Cache Levels 3
Page Size 512
Total Pages 1

```
C:\Users\Ashwin Balani\Documents\OneNote Notebooks\Personal Notebook\Mac
255, Frequency:126,Recency:1
254, Frequency:15,Recency:31
253, Frequency:16,Recency:19
252, Frequency:4,Recency:284
251, Frequency:8,Recency:34
250, Frequency:15,Recency:13
249, Frequency:5,Recency:1
248, Frequency:10,Recency:9
247, Frequency:17,Recency:17
246, Frequency:8,Recency:12
245, Frequency:21,Recency:55
244, Frequency:19,Recency:58
243, Frequency:11,Recency:111
242, Frequency:14,Recency:13
241, Frequency:11,Recency:43
```

```
75 at 83,Level 1 512
75 at 83,Level 1 512
75 at 83,Level 1 512
75 at 84,Level 1 512
75 at 84,Level 1 512
75 at 84,Level 1 512
75 at 84,Level 1 512
75 at 84,Level 1 512
75 at 85,Level 1 512
75 at 85,Level 1 512
75 at 85,Level 1 512
75 at 85,Level 1 512
75 at 86,Level 1 512
75 at 86,Level 1 512
75 at 86,Level 1 512
75 at 86,Level 1 512
75 at 86,Level 1 512
75 at 87,Level 1 512
75 at 87,Level 1 512
75 at 87,Level 1 512
75 at 87,Level 1 512
75 at 88,Level 1 512
75 at 88,Level 1 512
75 at 88,Level 1 512
75 at 88,Level 1 512
75 at 89,Level 1 512
75 at 89,Level 1 512
75 at 89,Level 1 512
75 at 89,Level 1 512
75 at 90,Level 1 512
75 at 90,Level 1 512
Number of hits 14318 out of 14320
Number of misses 2 out of 14320
Hit Percentage 99.98603 %
Miss Percentage 0.01396942 %
```

Fig. 3: Output screen showing System Log and Algorithm Log.

This is the second output screen, once you find the cache size and set parameters like number of cache level, Page size and total pages after that the system will ask to apply either single cache management policy or dual cache management policy. Once user has chosen the policy after that you have to select the different files from your system and then click on the start cache management button. As soon as the system receive the command of user for starting the cache management the processing will get start and after finishing the processing it will display the system log and algorithm log which give detail idea about the number of bytes present in the files, at what location of memory that particular byte available and at which level. Multiple level 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.

C. Aggregate hit ratio of different multiple levell cache policies

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

% Hit Ratio = (No. of Cache hits$*$ Total no. of request)

$*$100… (3)

% Miss Ratio = (No. of Cache miss$*$ Total no. of request)$*$100 … (4)

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 ≤ 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, multiple level achieves higher aggregate hit ratio than existing multilevel cache policy.

D. Average Response Time of Different Multiple Level 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.

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

Table 1: Policies Comparison on the Basis of Different Workload

IV. CONCLUSION

In this paper, to keep track of last N-step history information about the movement of a data block among multiple cache levels, multiple level cache management policy is implemented. The activeness of a block is determined by resident time of the block at the particular cache level, the frequency of the demote/promote operations of a block, size of block. Multiple level demotes passive data to the lower level while promotes active data to the upper cache level more efficiently. An Multi-attribute based model is developed to easily identify the activeness of blocks. The results show that Inter-cross achieves better performance compared to DEMOTES, FIFO, LRU-K 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. G. Yadgar, M. Factor, and A. Schuster, Karma: "Knowit-all replacement for a multilevel cache." In Proc. Of the 5th USENIX Conf. on File and Storage Technologies, San Jose, CA, February 2007.
5. B. 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, July2010.
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. G. 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. 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.
13. C. 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: Knowit-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.