International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 5 142 – 145

# RFS-UCM: A Unified Multilevel Cache Management Policy

Ms. Komal Thakre
Department of Computer Science & Engineering
Nuva College of Engineering & Technology
Nagpur, Maharashtra, India
*thakre1990@gmail.com*

Asst. Prof. Shyam Dubey
Department of Computer Science & Engineering
Nuva College of Engineering & Technology
Nagpur, Maharashtra, India
*shyam.nuva@rediffmail.com*

Asst. Prof. Nilesh Choubey
Department of Electronics & Communication Engineering
MIET
Gondia, Maharashtra, India
*Choubey.nilesh@gmail.com*

***Abstract*** **-** Performance plays a vital role in distributed systems. To improve the performance of an I/O system, there is a great demand in multilevel cache over a single cache because of its efficiency. Many policies of multilevel cache are present still there is a performance issue. 1) These policies fail to select optimally cache block for replacement i.e. fail to select a victim and 2)increases redundancy causes cache pollution at lower level. These policies include LFU [1], LRU-K [2], PROMOTE [3], DEMOTE [4], Multi Queue (MQ) [5]. In this paper we introduced compressed cache management policy which will address drawbacks of above defined policies by considering three factors together to replace or update cache block in a multilevel cache hierarchy, secondly it prevents cache pollution at lower level cache. First factor is recency of object in a cache i.e. how nearly object is used, second is frequency i.e. How repeatedly promotion and demotion of cache block takes place in a cache. Third is considering object size, the object with largest block size and less recency and frequency will be evicted first. This policy is capable of selecting a cache block efficiently and thus gives higher hit ratio compared to other present multilevel cache policies

***Index Terms*** **-** *Multilevel-cache, Frequency, recency, promote, demote, cache performance, Distributed Systems.*

*****

## I. INTRODUCTION

As the quick growth of the Internet service, numerous data centers have built large-scale distributed storage systems, where multilevel hierarchical storage systems are used to satisfy the ever increasing high performance I/O demand. In a typical hierarchical structure, the upper level storage server's serves as a cache for the lower level, which forms a distributed multilevel cache system. This cache manages the data which might move among different levels depending on the usage access patterns. To recognize and manage these data, hints [1] [2] are an efficient way to improve the performance of a storage system. In near the beginning research on cooperative caching hints [3] [4] were used to approximate the global view of a storage system.

With the technical development of multilevel caches, especially the progression in the exclusive cache schemes, hints are not only limited to show the status of global management on data blocks, but also the dynamic information of a detailed data block in a storage system. Based on different roles most critical research issues in distributed involve cache replacement as well as cache validation and consistency. Whenever the cache is occupied and the proxy need to cache a new object, it has to decide which object to remove from the cache to put up the new object. The strategy used for the expulsion decision is referred to as the management or replacement policy. In order to evaluate a cache management policy, a general practice is to measure two performance rates, the hit rate (HR - is the proportion of the number of requests that are served by the cache over the total number of requests) and the byte hit rate (BHR – is the proportion of the number of bytes that belongs to the requests served by the cache over the total number of bytes requested).

An elevated HR shows an effective cache management policy and defines an increased user servicing, reducing the average latency. On the other hand, a high BHR improves the network performance (i.e. bandwidth savings, low congestion etc.). Usually, the cache past status is used to predict the future cache substitution actions. Each object is defined by a 'value', the so-called cache efficacy value (CEV), for each object where the objects with the smallest utility outcome will be the first candidates to remove from the cache. Previous cache management policies have been categorized in:

- *Objects Recency-based*: The object's cache efficacy value (CEV) is known at the last reference to that object. LRU (least recently used) is the most suggestive algorithm of this category and has been applied in many proxy caching servers.
- *Objects Size-based*: The object's CEV is known by its size. The most suggestive algorithm of this category is the SIZE [5] which considers objects size as the basic constraint (large objects are driven out first), thinking that users are less likely to re-access large objects because of the high access delay connected.
- *Objects Frequency-based:* The objects CEV is known by the number of requests to that object. Frequency based solutions are variation of the LFU (Least frequently used) policy and they use the popularity of objects for the replacement decision.
- *Objects Function-based*: The object's CEV is known by a cost function, which involves multiple parameters or weights related to a performance metric (BHR and HR). The most suggestive algorithm of this group is the Greedy-Dual size. Generally systems use a objects recency-based policy (LRU), due to its simplicity, but the drawback of such policies is that they do not consider the size of the objects and the delay time in the server transmission servicing, ensuing to high BHR. The objects size-based policies usually result in higher HR since they maintain smaller objects in cache (i.e. more objects reside

in cache). The frequency-based polices keep in cache the most popular objects independently on their size and on their recency status i.e. they avoid keeping in cache recently accessed objects but with a low frequency (such as the so- called "one-timer" objects1). Finally, the function-based policies use complex formulas and usually suffer from profound parameterization and soaring overhead.

### a. RELATED WORK

There is numerous different cache schemes have been proposed over the past decades, which can be grouped into two main categories: single cache level cache and multilevel cache level.

### *A. Single Level Cache*

Least recently used i.e. LRU [5] is widely used in cache management. Since the 2000s, many variants of LRU aim to improve the performance of single-level cache. LRU-K [7] keeps record of the last K references to popular pages, adapts in real time to the changing access patterns, and performs better than the conventional buffer cache algorithm. The optimality proof of LRU-K was given in the later research [8].We only list some famous LRU-based algorithms: such as frequency based i.e. FBR [19], multiple queue 2Q [9], unified buffer management UBM [10], least recently and frequently used LRFU [11], LIRS [12], STOW [13], and so on.

### *B. Multilevel Cache*

There is a few multilevel cache policy come out to optimize the collective performance of distributed systems, which also shows the difference between the RFS and other solutions.

TABLE-I: Comparative Study of Multilevel cache policy

| Multi-level Cache Policies | Cache Level | E1 | D2 | P3 | Application-based policies |
|---|---|---|---|---|---|
| MQ | 2nd | NO | NO | NO | Access-based placement |
| DEMOTE | 2nd | YES | YES | NO | * |
| ULC | 2nd | YES | YES | NO | Eviction-based placement |
| X-RAY | 2nd | YES | YES | NO | Eviction-reference placement |
| MC2 | All | YES | YES | NO | Approximate application hints |

Notes:
E1: Exclusive caching.
D2: Demote-like policies
P3: Promote policies
*: not discussed but may be supported if needed
All levels: at least perform well in the 1st level and 2nd level

## II. PROBLEM FORMULATION

A cache management problem involves some constraint that is used to monitor the cached objects replacement process. In practice, each cache management policy defines a CEV assigned to each cached object such that the object with the appropriate CEV will be driven out from cache. In this paper, the Web cache content is modeled by a linked list in which each node is associated with a particular cached object. Therefore, the number of nodes is bounded by the number of cached objects. Here, we consider each cached object to be identified by its corresponding stored object filename; along with a number of related attributes (e.g. object's size, time of object's request etc.). The basic goal of the proposed cache replacement problem is to maintain in cache the objects with the largest CEV in order to achieve high HR [18][19] [20].

The total cache size is computed in bytes and is of fixed size (S). The total set of objects that are stored in the cache is not fixed, but it depends on which objects are cached each time (denoted by Ni). Let oi be the Web object which is requested at the i-th request. If oi is in the cache, then we have a cache hit. Otherwise, we have a cache miss and the object oi should be inserted into the cache. In case that there is not enough space in cache, there is a need to evict one or more objects from the cache in order to free sufficient space[14] [15].

For each cached object x we keep track of: the object id (ox), the object size (sx) in order to handle the *size* balance, the number of accesses since the last time object x was accessed in order to support *recency*, and the node id which is updated according to its *frequency*.

It is important that the cache replacement process should guarantee enough space for the incoming objects. In this context, there are two actions related with the replacement process (i.e. either the object will remain stored in cache or it will be evicted from cache). We define a function to identify the action that should be taken for each cached object [16] [17].

## III. THE RFS-UCM UNIFIED CACHE MANAGEMENT POLICY

The proposed cache management policy is called SFR (*Recency, Frequency and Size Cache* Management), and it involves two stages:

### *A. Stage 1- Cache Segmentation:*

Partition the cache into a number of segments which are in accordance to the objects' sizes. Therefore the objects are stored in cache into certain variable-sized segments.

### *B. Stage 2- Cache replacement:*

All of the defined cache-segments will apply a cache replacement policy.

### A. Cache Segmentation

The motivation for a cache-segmentation is the flexibility that it offers to manage the cached objects, since the cache segments may grow and shrink deliberately (according to request streams) and also at each segment a separate replacement policy may be applied. In this paper we segment the cache in a way inspired from the work presented in

[21][22]. Next, we propose hierarchical pyramid-like cache segmentation PSS (Pyramidal Selection Scheme [1]) depending on objects' sizes such that objects are placed in the appropriate segment according to theirs sizes.

- ❖ Object with high cache efficacy value (CEV) and having small block size will be stored at L1 cache, high efficacy value indicates that it is more frequently used.

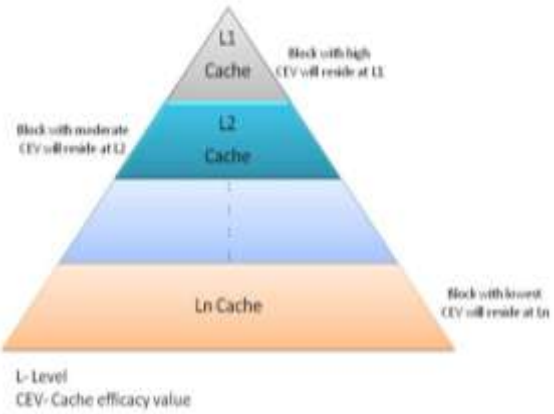


Fig.1. Pyramidal Cache Segmentation

- ❖ Object with moderate cache efficacy value (CEV) and having middle size will be stored at L2 cache.
- ❖ Object with very less cache efficacy value (CEV) and having bigger cache size will be stored at last cache level.

B. Hybrid Cache Replacement Policy

A typical replacement approach involves updating the cache content under a certain criterion or over a considered time period. Here, we consider that a cache replacement is occurred when there is a need for cache disk space. Specifically, whenever an object is requested and it is a cache hit, we update the place where the requested object is stored in the list (namely, we re-order the list by moving the requested object towards to the tail of the list). On the other hand, in case of a cache miss, the requested object is located to the appropriate cache segment with respect to its size.

Thus, regarding to the amount of free cache space, we have two options. If there is enough free space to accommodate the object 'Ox' in the cache, then we locate the requested object to the appropriate cache segment. If not, we should make space by evicting from the cache one or more objects. Then, we locate the requested object to the appropriate cache segment.

C. Implementation Details

The below flowchart indicates how the hybrid cache management policy works. Whenever the data is requested it is first searched into the cache i.e. L1. If the data is found in level1 cache it called as hit and data is returned from cache. On the other hand if data is not found in L1 cache then, search is carried on L2 cache, if data found it will return the data.

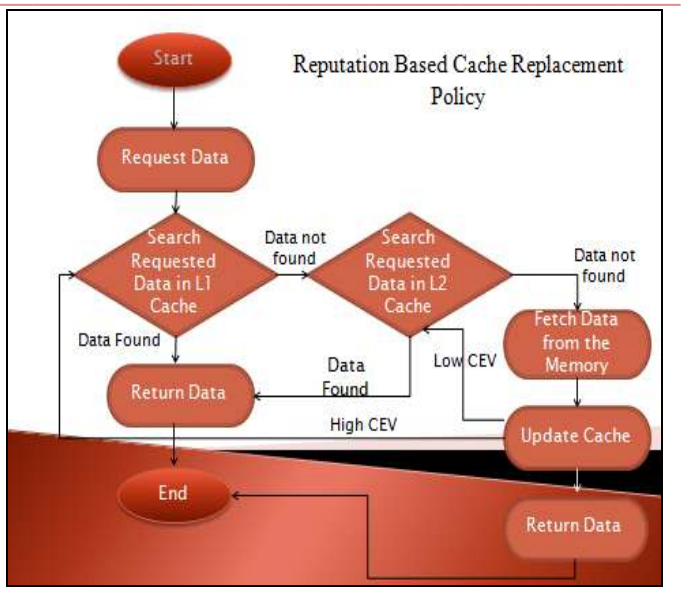


Fig.2. Flowchart Showing Hybrid Cache Replacement Policy

If data is not found in L2 cache then it is searched in secondary memory and data is returned from that. The cache is updated according to function based policy. As level1 cache is small in size so object with smaller size and high cache efficacy value is stored in L1. This way unified cache management policy gives improved hit ratio and reduces the shifting of cache blocks.

## IV. ANALYSIS AND DISCUSSIONS

The table's shows the RFS-UCM performance enhancement over a variable cache size. The plus sign shows improved performance and minus sign shows worse performance. This also indicates our proposed policy will attain a 9% higher HR comparing with variable sized LRU, for variable sized objects. Though, RFS-UCM sacrifices 3.5% of Byte Hit Ratio (BHR) in order to have a good improvement in HR. Unified cache management policy when compared with Pyramidal algorithm, it shows a small improvement in HR and BHR.

TABLE-II Comparative Quantitative Analysis

| ***RFS- UCM*** | ***PSS*** | | ***LRU -¥*** | |
|---|---|---|---|---|
| **Cache Size** | **HR** | **BHR** | **HR** | **BHR** |
| Large (15%) | +0.22% | -0.27% | +8.91% | -5.73% |
| Medium (10%) | +0.17% | +0.27% | +9.62% | -3.54% |
| Small (5%) | -0.10% | +0.26% | +9.44% | -0.91% |
| *Average* | +0.09% | +0.06% | +9.32% | -3.45% |

## V. CONCLUSION

Caching is a typical approach for improving the performance of distributed systems. In this paper, we introduced the unified cache management policy, where the decision about which objects should be ejected is determined by considering three object's factors: frequency, recency and

size. The proposed works is shown to perform well if applied practically in real word scenario under particular situations by the means of proper simulations. RFS- UCM help optimized cache utilization and achieves higher hit rates compared with the other cache management policies. An increased HR shows an effective cache management policy and defines an increased user servicing, reducing the average latency. On the other hand, a high BHR improves the network performance.

## REFERENCES

[1] 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.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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