A Framework for MF-LRU (Most Frequently-Least Recently Used) Replacement Policy

Sabarinathan Sayiraman, Senthil Kumar Dayalan, Shanmugavel Mani Subbiah

School of Computer Science and Engineering,
College of Engineering Guindy, Anna University,
Chennai, India.
Email: {sabarinathan, dsenthil, ssmani}@cs.annauniv,edu

Abstract

Replacement Policies play a crucial part in different aspects of today's high performance computing environments such as Web Caching, Cache Management in Microprocessors, Page Management in Operating Systems, Replication strategies in Distributed Information Systems etc. Since the block misses in these systems severely affect their throughput, the replacement policy for the blocks should be given more importance. In this paper, we propose a new replacement policy known as MF-LRU Replacement Policy. This policy is a blend of two popular Replacement policies namely LRU (Least Recently Used) and MFU (Most Frequently Used) Replacement Policies. Both the policies have their own advantages and disadvantages. This paper aims to provide solutions for eliminating the disadvantages in both these policies retaining their advantages.

1. INTRODUCTION

Recency and Frequency of references are the two important parameters that determine the likelihood of a block to be accessed in the near future. The LRU policy gives importance only to the recency of references. On the other hand MFU considers only the frequency of references. In MFU, the blocks that have been referenced more frequently are the candidates for replacement. However, since it doesn't consider the recency factor, it cannot differentiate between blocks that were once hot but now becoming cold and blocks that are currently hot. A block is said to be hot if its frequency density is high i.e. the references to the block are dense. In general, a block referenced frequently must be replaced only after it has finished all its services, as in loops. Hence, the block that was once hot and now becoming cold would be a better candidate for replacement than the currently hot block. For example let the reference times of two blocks be given by

Block A: {1,2, 3, 4, 5, 7, 10, and 18}
Block B: {6, 9, 11, 12, 13, 14, 15, and 16}

Here though the frequency of references of the two blocks are same, block A would be the better candidate for replacement as it is the one that was once hot but now (at time instance 19 and above) cold. Block B on the other hand is currently hot.

In this paper we propose a block replacement policy known as MF-LRU Replacement Policy, which is a blend of the LRU and MFU policies. It gives importance to both the recency and frequency of references.

2. RELATED WORK

FBR is a frequency-based policy that is similar to LFU but uses the concepts of correlated references [6]. In **LRU-K** policy, replacements are based on the time of the K^{th} to last non-correlated reference to each block [4]. In **2Q** Replacement policy, there is a special buffer called the A1 queue into which a missed block is initially placed. A block in the A1 queue is promoted to the main cache only when it is re-referenced while in the A1 queue. Otherwise it will be evicted when it becomes the LRU block in the A1 queue [5]. **LRFU** (Least Recently / Frequently Used) replacement policy subsumes LRU and LFU replacement strategies [2]. This policy takes the advantage of LFU and LRU replacement schemes. **LRU-SP** is a size adjusted and popularity-aware extension to Least Recently Used (LRU) for caching web objects [3].

3. MF-LRU REPLACEMENT POLICY

Unlike the LRU or MFU policies that consider either recency or frequency only, the MF-LRU policy takes both of them into account in the replacement decision.

The MF-LRU associates a value with each block called the RFF (Recency Frequency Factor), which quantifies the importance of that block i.e. the likelihood that the block be referenced in the near future. Thus the block with the least RFF is the victim for replacement.

The RFF for every block should change such that every access to that block increases its recency component and decreases its frequency component. For every block that is not accessed, its recency component should decrease. The RFF varies according to the weights attached to these components.

The main objective of MF-LRU policy is to replace an old frequency dense block. The function to calculate RFF for each block b accessed at the current time, t is given by

$$RFF_{b,t} = \sum_i (F(t-i)) - \{G(\delta) RFF_{b,last}\}$$

where,

- $F(x) = (1/p)^{\lambda x}, \; p > 1$
- λ decides the weights for recency and frequency, $\lambda \in [0,1]$
- $i = \{x \mid x \text{ is an access time of the block}\}$
- δ = (Current time t – last access time of blocks b)
- $G(\delta)$ is a function saturating at 1 with values in [0,1]
- $RFF_{b,last}$ is the RFF of the block b accessed at time 'last'.

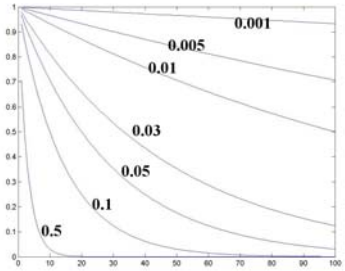


Fig 1: F(x) for various λ

The first term additively increases the importance of the block based on its recency. This is achieved by the function F(x), a saturating function which has the maximum value of 1. Due to the monotonic decreasing nature of F(x), for a set of access times i the value of $\Sigma(F(t-i))$ decreases as t increases. For instance, let a block be accessed at instances i={1,3,5,7}.

At t =10,

$$\Sigma F(10-i) = F(10-1) + F(10-3) + F(10-5) + F(10-7)$$
$$= F(9)+F(7)+F(5)+F(3)$$

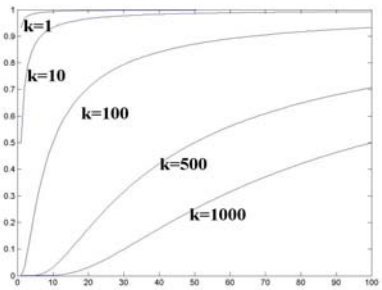
At t =12,

$$\Sigma F(12-i) = F(12-1) + F(12-3) + F(12-5) + F(12-7)$$
$$= F(11)+F(9)+F(7)+F(5)$$
$$= F(2)\ \Sigma F(10-i)$$

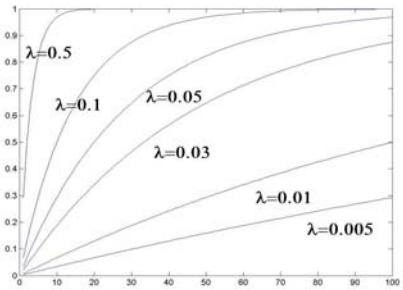
The second term penalizes the RFF of a most frequently, less recently used block i.e. the penalty on the RFF will be higher if it is accessed more frequently once upon a time but not now. The function G(δ) in the second term is made to serve this purpose.

G(δ) can be chosen from a class of monotonically increasing functions, each increasing at different rates. Some possible candidates for G(δ) are $p^{-k\lambda/\delta}$, $(1-p^{-\lambda\delta})$, $(1-p^{-\lambda\delta}) / (1+p^{-\lambda\delta})$. 'k' in the first candidate is a positive integer which controls the slope of G(δ). Higher the value of k, lower will be the increasing rate of G(δ) i.e. higher the k value, higher will be the chances of survival of an old frequency dense block.

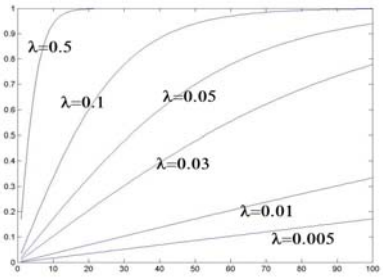
$G(\delta)=p^{-k\lambda/\delta}$ for various k (λ=0.1)



$G(\delta)=1-p^{-\lambda\delta}$ for various λ



$G(\delta)= (1-2^{-\lambda\delta}) / (1+2^{-\lambda\delta})$ for various λ



The penalty imposed on the RFF of a block for not being accessed can be inferred from the plots for G(δ).

4. AN IMPLEMENTATION MODEL

This cache replacement policy should require an efficient implementation to reduce the number of computations. As the block with the least RFF needs to be replaced, we need to maintain a data structure that makes it easy to efficiently return the block with the least RFF. Maintaining all the access times for a block is tedious. For that, we define RF(Recency factor) which stores the value of $\Sigma(F(t-i))$. The new RF can be calculated from the previous RF by the formula, $\mathbf{RF_{new} = F(\delta) * RF_{old}}$.

We use a heap in which the children always have RFF greater than their parent. Thus the element at the root has the least RFF among those in the heap. The manipulations associated with the heap can be reduced due to the following property.

Property as in [2]: For the function $F(x)=(1/p)^{\lambda x}$, there exists d_{thresh} such that

$$\forall\ d \geq d_{thresh},\ 1 > \Sigma F(i), i \in [d, \infty)$$

The significance of this property is that it distinguishes a set of blocks, only whose RFF can be higher than the RFF of the currently accessed block. If a block has not been accessed for the past d_{thresh} references, its RFF cannot exceed the RFF of the currently referenced block. So in order to minimize the operations in the heap we can maintain a list containing blocks, which are referenced before d_{thresh}. The list is maintained so that all its elements have RFFs lesser than that of the root of the heap. Blocks in the list move towards the tail as their RFFs decrease. The block at the tail of the list has the least RFF and is the candidate for replacement.

Replacement Algorithm

1. If a new block b is referenced
   - $RFF_{b,t} = 1$
   - Demote the block at the root to the head of the list.
   - Insert the block at the root and restore the heap.
2. If block b is in list
   - $RFF_{b,new} = F(0) + RF_{b,last} * F(\delta) - G(\delta) RFF_{b,last}$
   - Demote the block at the root to the head of the list.
   - Insert the block at the root and restore the heap.
3. If block b is in the heap
   - $RFF_{b,new} = F(0) + RF_{b,last} * F(\delta) - G(\delta) RFF_{b,last}$
   - If $RFF_{b,new} > RFF_{b,last}$
     - Restore the sub tree of the heap with the node containing block b as root.
   - Else
     - call Reorder(b);

Reorder(b)

- While $RFF_b < RFF_{b \rightarrow parent}$ { $b \rightarrow$parent : represents the parent of b }
  - Swap (b, $b \rightarrow$parent);
- Restore the heap from the current block b

The main parameters, which affect the RFF of a block, are λ and δ. The effect of each of these can be studied by their contribution to the RFF. The parameter λ specifies the weights given to the recency and frequency components.

4.1 AN ILLUSTRATION

Let the number of blocks be 5. Let the access pattern at successive times (t =1 to 15) be AP ={1,1,1,1,2,2,3,2,3,4,5,3,2,5,5}. The $RFF_{t=16}$ for each block (1,2,3,4,5) is found to be [-1.8384] [0.4654] [0.0222] [-0.5303] [1.8170]

The calculation suggests that block 1 is the first victim. This is in accordance with the MF-LRU policy as block 1 was once hot but has now become cold. The next candidate as shown by the calculation is block 4 due its lower recency. Block 5 has the highest RFF indicating that it's hot now. The RFF of the remaining blocks can be observed using the same semantics.

5.CONCLUSION

In this paper we presented a new replacement policy known as MF-LRU (Most Frequently – Least Recently Used) Replacement Policy. It exploits the advantages of MFU when it is combined with LRU by identifying, which blocks are currently hot and which blocks are not. This replacement policy has a lot of potential applications in various aspects of modern day high performance computing environments.

6.REFERENCES

1. John.L Hennessey, David. A. Patterson. "Computer Architecture: A Quantitative Approach", *Morgan Kaufmann Publishers, 2000.*
2. Donghee Lee, Jongmoo Choi, Jong Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, Chong Sang Kim. LRFU (Least Recently/Frequently Used) Replacement Policy: A Spectrum of Block Replacement Policies. *In IEEE Transactions December 2001.*
3. Kai Cheng and Yahiko Kambayashi. LRU-SP: A Size-Adjusted and Popularity-Aware LRU Replacement Algorithm for Web Caching.
4. Elizabeth J.O'Neil, Patrick E. O'Neil, Gerhard Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering". *In Proceedings of the 1993 ACM SIGMOD Conference pp. 297-306, 1993*
5. T. Johnson and D. Shasha. 2Q: A low Overhead High Performance Buffer Management Replacement Algorithm. *In the proceedings of the 20th Intl Conference on Very Large Data Bases, pages 439-450, 1994.*
6. J. T. Robinson and N. V. Devarakonda. Data Cache Management Using Frequency based Replacement. *In the Proceedings of the 1990 ACM SIGMETRICS Conference, pages 134-142, 1990.*