International Journal of Computer Applications (0975 – 8887)
Volume 12– No.12, January 2011

A Modified Algorithm for Buffer Cache Management

Mukesh Kumar Chaudhary, Manoj Kumar, Mayank Rai and Rajendra Kumar Dwivedi
Department of Computer Science and Engineering, M. M. M Engineering College, Gorakhpur, India

ABSTRACT

A fundamental challenge in improving file system performance is to design effective block replacement algorithms to minimize buffer cache misses. In this paper an algorithm is proposed for buffer cache management with prefetching. The buffer cache contains two units, the main cache unit and prefetch unit. The sizes of both the units are fixed. The total sizes of both the units are a constant. Blocks are fetched in one block look ahead prefetch principle. The block placement and replacement policies are defined. The replacement strategy depends on the most recently accessed block and the defined miss count percentage or hit count percentage of the blocks. FIFO algorithm is used for the prefetch unit.

KEYWORDS

Buffer cache management, prefetching, data access. File systems management, operating systems performance.

1. INTRODUCTION

Buffer cache management is a widely studied topic. Many algorithms have been proposed to improve the same. LFU, LRU-k, 2Q, FBR, LRFU, C-LRU (Cooperative -LRU), D-LRU (Distributed-LRU), N-Chance, RobinHood are some of them. In LFU, the least frequently used block is replaced. In LRU-k the algorithm keeps track of the last k references to a page. The page which has shortest interarrival time is retained in the cache. In 2Q two queues are maintained to place pages as either hot or cold. On a re-reference to a page in a queue, it is treated as more likely to be referenced. In FBR, the blocks are maintained in LRU order but are replaced in least frequently used order. In LRFU, a function involving the time of access to the blocks is taken into account to determine the block to be replaced. Prefetching has also been proven to be effective as discussed in [3, 4, 6 and 7]. An alternative to LRU replacement was suggested in [9]. C-LRU, RobinHood algorithm is cooperative algorithm. C-LRU based on the D-LRU et. al [8], the idea is that when a client requests a chunk from another client, a new copy of this chunk will be created by this the importance of the chunk in both clients should be reduced and also RobinHood is based on the N-Chance algorithm, N-Chance for singlet evicted but RobinHood, a singlet (i.e. "poor" chunk) is forwarded to a peer that has a chunk that is cached at many clients (i.e. "rich" chunk). The File system speed has impact on buffer cache management [1]. In [3] an algorithm called W^2 *R* algorithm proposes a method for prefetching in an aggressive manner. The authors propose a method where the cache is considered to be of two units – Weighing and Waiting whose sizes can be changed dynamically. The waiting room has blocks that are prefetched. The weighing room has blocks that have been accessed. The algorithm follows one block look ahead (OBL) for prefetching. The sizes of the two rooms are adjusted based on the time of access of a block from the time it is brought into the Waiting compartment. This paper proposes an algorithm to place and replace blocks in the buffer cache based on OBL principle. The buffer cache consists of two parts – prefetch unit and main cache. The sizes of the two units are fixed. Model with variable sizes is a topic of future research. On a miss, the block is fetched into the main cache. The next sequential block is fetched into the prefetch unit. On a hit in prefetch, the block is brought into the main block. The block with maximum number of misses which is not the most recently accessed is replaced by the algorithm of et al[1] but by this algorithm the maximum miss percentage miss count/(miss count + hit count) * 100) and which is not the most recently accessed is replaced by the new block or maximum hit percentage block is continue. It means which block has the higher hit count, its more chance to in main cache so continue first. On every hit, the corresponding block's hit count is incremented. Simultaneously, the miss count of all the other blocks in the main cache is incremented. This is to say that the rest of the blocks are not useful at that point of time. It is not possible for two blocks to have same hit and miss counts as they are fetched at different points of time. The algorithm keeps track of most recently accessed and the relative usage of the blocks over a period of time. The rest of the paper is organized as follows. Section 2 gives motivating example, section 3 gives the algorithm, section 4 the conclusion, and section 5 references.

2. MOTIVATING EXAMPLE

Consider a list of references. In W^2 *R* algorithm the LRU algorithm is used for replacing blocks in the weighing room. Consider the following scenario. A block b_1 is brought in at time t1. It has 20 misses and 40 hits. It has been in the cache for 60 units of time. Let block b_2 have 30 misses and 70 hits and block b_3 have 7 miss and 20 hit. Let the LRU block is b_1. If the size of the cache is three blocks, then if block b_4 is needed, then block b_1 is evicted. If the future references were for block b_1, then there would be a miss. The et. al[1] algorithm finds the block with maximum number of misses which is not most recently accessed,b_2 is that block. Hence it replaces b_2 Hence the request for b_1 is a hit, but this algorithm the maximum miss percentage is b_1 (33.3%) and b_2 has 30% or just opposite the hit percentage of b_1 is 66.7% and b_2 is 70% and not most recently accessed block replace. So it replace the b_1. The algorithm is based on the concept that blocks with maximum amount of non access over their life in the cache till the current point is good candidates for replacement. This is the motivation.

3. ALGORITHM

This section describes the algorithm for the proposed model. It derives the time complexity of the algorithm and differentiates with other algorithms. The algorithm proceeds as follows the proposed system contain two parts main cache unit and prefetch unit. The sizes of both the units are fixed. Buffers main cache unit and prefetch unit hold the cached and prefetched data. The size of the whole system is a constant. Each block in the main cache unit has two counters miss and hit. The miss counter gives the total number of non references to the block and the hit count gives the total number of references to the block while it is resident in the cache. The following are the steps taken on an access to a block. Each block is identified by its block address.

1 If the block is in main cache its hit count is incremented.
2 If the block is in prefetch unit it is brought into the main cache unit. The placement/replacement strategy is as follows. If there are empty slots in the main cache, the block is placed in it. Else, the block with the maximum miss count percentage i.e. (miss count / (miss count + hit count))*100 and which is not the most recently accessed is replaced. The hit count of the block is set to one and it is marked as the most recently accessed block.
3 If the block is not in prefetch and main cache, it is fetched from the disk to the main cache. The Replacement policy is same as described in step 2, The hit count of the block is set to one and the Miss count of the block is set to zero. The next sequential block is fetched into the prefetch unit.

The time complexity of the algorithm is of the order of sizes of the main and prefetch caches. The worst case is when there is a miss and the whole of main cache and prefetch cache has to be searched before placing the block in the main cache. The following is the conceptual organization of the buffers. The address stream consists of the generated addresses. Each address is depicted by 1 in the Figure 1. It is searched for in the buffer cache depicted by 2. If found, the hit count is incremented. If not, it is searched for in the prefetch unit depicted by 3. If found, it is fetched into the main cache depicted by 4. The hit counter is suitably incremented for the fetched block. In case of a miss, the data is fetched from main memory to the main cache unit depicted by 5. The next physical block is also prefetched into the prefetch unit depicted by 6.

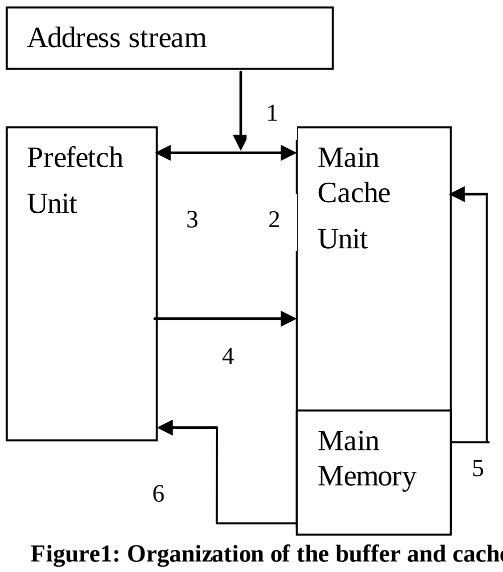

Figure1: Organization of the buffer and cache and sequence of steps to fetch an address.

Various kernel components on the path from file system Operations to the disk by et. al [2] figure 2

File System accesses through various kernel subsystems. Firstly it accesses the buffer cache which contains two units, the main cache unit and prefetch unit. The sizes of both the units are fixed. The total sizes of both the units are a constant. FIFO is used in prefetch unit .There are two counters which count miss count and hit count which are useful to calculate the miss count percentage and hit count percentage. Kernel I/O makes an I/O request to Kernel I/O clustering unit which then initiates an disk request to Disk Driver to fetch the block from disk to main cache unit.

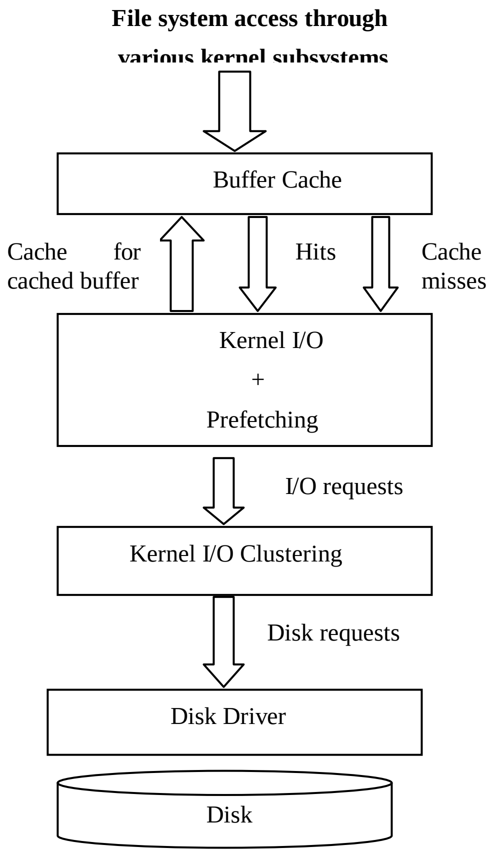

Figure 2: Various kernel components on the path from file system operation to the disk.

The proposed algorithm ensures the following.

1. The number of misses for any two blocks is not the same. This is because blocks are fetched at distinct units of time. While a block is fetched, its hit count is set to one. The incrementing of the miss count takes place when the block is not referenced. Suppose there are two blocks a and b. Let them be fetched at time t_1 and t_2 respectively. Let $t_1 < t_2$. Then assuming there was only one reference to a, the miss count of a is $t_2 – t_1$- 1. The miss count of b will start after t_2. Hence the result.
2. The proposed algorithm differs from LFU. In Least frequently used, the block that was referenced minimum number of times is replaced. In the proposed algorithm, the block could have maximum number of hits and also misses. Hence it could be replaced though it is most frequently used within the time span of its arrival and

current interval. For example suppose there is a request for blocks 1, 4, 5, 1, 5, 5, 4, 4, 5, 5, 4, 1. The following table gives the statistics for this address trace. Let there be a request for block number 6. According to LFU, block number 1 will be replaced. According to the proposed algorithm, block number 1 has the highest miss count percentage but it is the most recently accessed. Hence second block with maximum miss count percentage ,that is block number 4, will be replaced where miss count percentage is calculated by the formula: ((miss count)/(miss count + hit count))*100.

3. The proposed algorithm differs from first in first out (FIFO) algorithm. In FIFO, the block that is fetched first is replaced first irrespective of its last time of reference. In the proposed algorithm if the block is the most recently accessed, it is not replaced.

Table 1: Example for comparison of proposed algorithm with LFU algorithm.

S.NO.	**Block No.**	**Miss count**	**Hit count**	**% Miss count**
1	1	7	3	70
2	4	8	4	66.67
3	5	7	5	58.33

The proposed algorithm differs from LRU. In

5. LRU, the least recently used block is placed. In the proposed algorithm it could be the case that the LRU block has lesser misses than others. Hence it won’t be replaced in this case.

6. The proposed algorithm differs from MRU. In MRU, the most recently used block is replaced. This is not the case in the proposed algorithm. Consider the mathematical aspects of the proposed model. The input stream can be
   a. Sequential
   b. Random
   c. Mixed

For sequential input, if a block b_i is in main cache unit; its successor b_{i+1} will be in the prefetch unit. Hence there will be miss only for the first block and odd numbered blocks. All other accesses are hits in the prefetch unit. For random input, suppose that block b_i in the main cache unit. Let the probability of the next reference being b_{i+1} be p_1, b_{i-1} be p_3, bj where j>i+1 be p_2, b_k, k<i-1 be p_4. Then the probability of the next block is a hit is p_1+p_3 and miss is p_2+p_4. Based on the random function used, these can be determined.

4. CONCLUSION

A new algorithm to achieve buffer cache management based on prefetching has been proposed in this paper. The algorithm assumes two units main cache unit and prefetch unit in the memory. The main cache unit is the buffer cache and the prefetch unit has the prefetched blocks. The algorithm increments the hit count of a block on every access and increments the miss count of a block for every non reference to it. The most recently accessed block is not chosen for replacement. The block with maximum miss count percentage that is not the most recently used is used for replacement in case of conflict. The time taken to execute the algorithm is O(fetch time + prefetch time) in the worst case of a miss in the main and prefetch units. For the W^2R algorithm the time complexity is also O(fetch time + prefetch time) and the time complexity of et. al [1] is also O(fetch time + prefetch time). The algorithm has been compared with W^2R algorithm. The algorithm gives the same performance as W^2R algorithm for sequential inputs and performs better in terms of number of hits for random inputs.

5. ACKNOWLEDGEMENTS

We (all three members) are very thankful to our eminent teacher Lect. Rajendra Kumar Dwivedi, who has inspired and motivated us to achieve such type of daunting work, He has constantly accelerated us to finish our work within time.

6. REFERENCES

[1] S.Subha, An Algorithm for Buffer Cache Management, Sixth International Conference on Information Technology: New Generations, 2009.

[2] Ali R.Butt, Charis Gniady and Y. Charlie Hu,The Perfomance Impact of kernel Prefetching on Buffer Cache Replacement Algorithms, ACM SIGMETRICS, 2005.

[3] H. Seok Jeon, Sam H.noh, A Database Disk Buffer Management Algorithm based on Prefetching, Proceedings of the seventh international conference of Information and Knowledge Management 1998 pp-167-174 .

[4] Hui Lei, Dan Ducha mp, An Analytical Approch to File Prefetching, Proceedings of the USENIX 1997 Annual Technical Conference, pp-275-288, 1997.

[5] Mark palmer, Stanley B. Zdonik, FIDO, A Cache that learns to Fetch, Proceeding of 17th International Conference on Very Large Databases, pp-255-264, 1991.

[6] Pei Cao, Edward W. Felten, Anna R. Karlin, Kai Li, A Study of Integrated Prefetching and Caching Strategies, Measurement and Modeling of Computer Systems, 1995.

[7] Pei Cao, Edward W. Felten, Anna R. Karlin, Kai Li, Implementation And Performance of Integrated Application-Controlled File Caching, Prefetching and Disk Scheduling, ACM Transactions on Computer Systems, 1996.

[8] Eric Anderson, Christopher Hoover, Xiaozhou Li, New Algorithm for File System Cooperative Caching, ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2010.

[9] Theodore Johnson, Dennis Shasha, 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, Proceedings of the Twentieth International Conference on Very Large Databases, 1994.

[10] Song Jiang, Xiaodong Zhang, Workloads: A Novel Replacement Algorithm to Improve Buffer Cache Performance, Transactions on Computers, 2005.