**"Content Replacement in Cache: A Survey on available Algorithms"**

Mobashirah Nasir, Munim Ali Shah

COMSATS Institute of Information Technology

Park Road, Chak Shahzad

Islamabad, Pakistan

mobashirahnasir@gmail.com, mshah@comsats.edu.pk

**Abstract**

**Caching strategies can improve the overall performance of a system. One important factor in caching is the replacement policy. Due to advancement in technology there exist a huge number of Algorithms that were implemented to attain maximum throughput. Using this classification different algorithms and their advantages as well as disadvantages are described in this paper. Furthermore, by conducting survey on algorithms a new solution to attain better performance has been introduced.**

Keywords- Recency, Buffer memory, Paging, Pre-Fetching, LRU, Cache hit and cache miss, Random Replacement, Penalty based, Threshold, Virtual memory

**I. Introduction**

Cache Memory is a small and high speed buffer memory used in modern computers. Cache memory is basically used to store data and information which are currently being used. It takes more time to access data from the main memory, therefore to reduce this time a special memory inside the CPU is reserved to keep small amount of data for some time. Thus, a CPU with cache memory needs less time to wait for an operand and instruction to be fetched from the memory for processing. It takes 300 to 600ns to access contents from main memory whereas it takes 50 to 100ns to obtain information from cache [1]. Absence of Cache memory decreases execution rate which effect the performance of CPU.

Due to small size of Cache Memory it is needed that its content should be replaced per the usage and specific time. A key component of a cache is its replacement policy, which is a decision about evicting a page currently in the cache to make space for new page or data item. Here the problem is that what part of data or information is going to be replaced either the least frequently used data item or randomly any data should be replaced. Different History-Based Prediction Algorithms were developed and implemented. These Algorithms maps pages per their suitability for eviction. Although they have some drawbacks but to achieve better performance they are used.

If contents of cache changed quickly then Data-Finding technique can hardly make a prediction based on history, therefore it is even more expensive by giving us more rate of latency. To minimize the impact of latency and to make Cache beneficial, Cache replacement must be regulated. We must have to slow down the update/replacement rate so that contents of Cache can be utilized and traced by others.

The topic "Content Replacement in Cache" cover two different domains, one is Web Cache and the other is System/Processor Cache. Web Caches have several additional factors like frequency and recentness of pages, cost of fetching documents, and size of documents etc. To implement classical schemes in Web Cache, we must maintain complex data structures and the operations of data structure that leads to net traffic jam [2]. Whereas in Processor Cache, Page size is same therefore the cost of fetching different documents are same but it is not true for Web Cache because document size is not fixed so cost also varies with the size of document.

Cache Aspects

In Cache memory fetching Algorithm decides which data is to be fetched on demand (when it is needed) or pre-fetched (before it is needed). Pre-fetch algorithms decide which information is needed in near future and then obtain this information in advance. These algorithms are also responsible to omit information or data from cache which no longer used by CPU. Caches have a hierarchy, like it has small set of Associative memories, therefore only one associative memory is used to search for a content in a Cache. These small associative memories are referred as Set and the number of element over which associative search is conducted is called Size of that Set. The Replacement Algorithm is used to determine in which set the information/data will be placed. The unit of transfer between Cache and Main Memory is referred as Line size or Block. Selection of Line size is important factor while designing Main Memory.

Cache Hit: Requested data found in Cache which results in data transfer at maximum rate. [3]

Cache Miss: Requested data not found in Cache due to which Processor loads data from Main memory and copies it to Cache, which results in delay called Penalty. [3]

Operations in Cache

In hardware cache memory acts as a Block. Processors and hard drives use it frequently. Cache has several entries, each entry comprises of a datum, along with a specific tag that identifies the datum in backend. Write Policy describes the timing of write.

There are two basic writing techniques.

Write-through: in cache and back storage writing is done synchronously.

**Write-back:** Writing is done initially to the cache only, writing on back store is postponed until contents of cache should be replaced.

from the point of implementation, write back is much more complex because it needs to keep track of the locations which are going to be replaced by marking them as Dirty for further writing to the backing store. once data in the above locations are written back then they are evicted from the cache which shows an impact of Lazy write.

Write cache eliminates much traffic as a write back cache. [4]

**Cache Coherency:** In uni-processors there exists an inconsistency between write back caches and memory(RAM) but it doesn't cause any problem. But for Multiprocessor systems it should make sure that consistent data is available to all processors that's why cache coherency must be maintained, either by software or hardware techniques. [5]

**II. Related Work**

Rep**lacement Algorithms**

Replacement policies are used to attain optimized usage of cache. When cache is full, then the Replacement policies take action and evict pages according to the specified Algorithms. Replacement policies decide which data must be evicted in order to make space for new data that is currently being used. An efficient Algorithm is that which can take less time and number of cache misses are low and balancing cost. Following are some of the algorithms.

i. Clock with Adaptive Replacement Algorithm (CAR)

CAR is simple to implement and it has very low overhead on cache hits. It shows high performance and it also provides service of self-tuning. It is scan resistant which results in low space over heads that is less than 1%. CAR doesn't care for certain workloads. [6]

ii. Penalty Based Algorithm

Penalty based Algorithms improves the overall Cache hit ratio which results in better performance of a system. In this Algorithms Load and Store operations are predictable. This scheme is combined with other Replacement Algorithms to further improve the performance.

When the cost of replacement is correctly predicted only then Penalty Based Algorithm will give positive results. Since Prediction do not always give 100% result therefore this algorithm gives negative result in some cases.[1]

iii. Least Recently Used Algorithm (LRU)

This Algorithm discards the least recently used item from the cache in order to make space for the new data item. In order to achieve this, history of all data items that is which data item is used when, is kept. A variable known as Aging Bit is used to store this information. Although this Algorithm provides better performance but cost of implementation is much more.

iv. Threshold Parameter

A value known as threshold is required to estimate the expected time needed to completely replace or to fill the Cache content. This threshold is dynamically evaluated based on low and high water marks and current cache size. When Cache size is closer to lower water mark then its threshold has a high value but when Cache size is closer to higher water mark then the value of threshold is smaller.[7]

v. The History Least Frequently Used Algorithm (HLFU)

Cache replacement in typical LFU algorithm is performed by replacing the least frequently requested objects. But in HLFU which is an extension of LFU considers the History function in Cache Replacement process. Respective LFU threshold which is a linear function on the amount of currently used Cache. The HLFU algorithm will replace the cached objects based on Hist value as compared to the defined threshold in LFU. [7]

vi. Least Frequently Used Algorithm (LFU)

This Algorithm counts how often data item has been used. The Data items which are used less are deleted from the Cache first. If all objects have same frequency then this algorithm randomly discards any data item.

vii. Lowest Latency First (LLF)

This Algorithm keeps the average latency to a minimum by first expelling the object with the lowest download latency. This Algorithm gives the best result in cases where the data is retrieved by executing a query against a Relational Database.[8]

viii. Greedy Dual Size (GDS)

In this Algorithm index is calculated according to the size of a file. Larger the file the smaller is the index. File with the smallest index is replaced in this Algorithm. Inflation value is used to keep track of frequently accessed files in the Cache.[9]

ix. Round Robin (RR) Algorithm

In this Algorithm there is an eviction index which is set to a constant value for all files. This provides a lower bound on Performance. There is no other policy which performs poor than this one.[9]

x. Adaptive Replacement Cache (ARC)

Like LRU , this Algorithm is easy to implement. It's running time per request is totally independent of the cache size. ARC has a low space overhead of approximately 0.75% of the size of a cache. ARC is a scan resistant that is , it allows one time sequential requests to pass through without polluting the contents of cache. ARC also leads to self-tuning. This Algorithm continuously balanced Recency and Frequency features by responding to changing access pattern [10]. The basic idea in this Algorithm is to divide the cache into two queues, each is managed by using LRU or CLOCK(CAR) that contains pages accessed only once, while the other contains the pages which are accessed more than one time. Like other Algorithms ARC also has a constant complexity per request. A pre-fetched block is kept in the first queue along with a special Flag, so that on demand the required data block remains in first queue. "Ghost cache" is a special term used in this Algorithm to handle the data element which will be used in near future. [11]

xi. Random Replacement Algorithm

This Algorithm randomly selects any of the data item from the Cache and then replace it with the desired one. This algorithm doesn't need to keep track of the history of data contents. Due to which it consumes less resources, therefore it's cost is less as compared to other algorithms. This algorithm has been used in ARM Processors due to its simplicity. It also admits efficient stochastic simulation [12].

xii. Segmented LRU Algorithm (SLRU)

This Algorithm partitions the cache into two portions, one is unprotected and the other is protected portion(reserved for mostly used objects). When first request for an object has been done then this object is inserted into the unprotected portion. On a cache hit the object is moved into the Protected one. Both portions are managed by LRU technique [13]. But content from unprotected part has been removed and content from protected part has been moved back to the unprotected part as recently used content. This method requires a variable that calculates what percentage of the cache space is reserved for Protected Part.

xiii. Optimal and FIFO Algorithms

The optimal policy selects that on which time, the required page is accessed. It assigns time to the content of cache. Previous results shows that this Algorithm results in fewer number of cache misses [14], [8]. Practically this Algorithm is impossible to implement because it requires that operating system have full-fledged knowledge of the future events. Although this Algorithm will not be implemented but it provides standard to judge the performance of other real world Algorithms.

 The First in First Out Algorithm treats the page as a circular buffer, and pages are removed in a Round Robin style. Only thing that is required is Pointer that circulates through the frames of a page in a process. That's why this is the simplest algorithm to implement. Benefit of using this algorithm is that it replace the page from the memory that has not been used for a long time.[10]

This paper provide comparative analysis of different algorithms that are used to replace contents in Cache Memory of a Processor. And to achieve better performance a beneficial and efficient solution for content replacement in Cache will be introduced. This implementation is practically not implemented.

Recently a lot of work is done on Replacement Policies, some cost effective as well as efficient Algorithms were also proposed, but they are practically not implemented.

Janapsatya and Ignjatovi proposed an algorithm for on-chip L2 cache. They introduce gate level hardware implementation of Clock Algorithm for on-chip caches. Power Analysis and Timing of gate level implementation has shown that Clock along with DC is feasible.[15]

Amit, Kartik, Keval, Manish and Pramila compared different page replacement Algorithms . In this paper not only Traditional algorithms like LRU and Clock were discussed but the recent approaches like Low Inter-reference Recency Set (LIRS), Clock-Pro and Adaptive Replacement Cache (ARC) were also studied.[6]

Olivier Temam\* extended Belady's algorithm which determine the minimum cache miss ratio. He also formally proved the upper bound of local memory performance with metrics. Using this Algorithm he showed that in small memories large block sizes can easily be used without including additional memory.[16]

Qingbo Zhu and Yuanyuan Zhou presented an article in which algorithms related to power-aware storage cache management has been discussed. Dynamic programming technique is used that minimizes the consumption of disk energy. They present a Greedy Algorithm that minimizes cache miss ratio. In this paper they also propose two online power aware algorithms that are PA-LRU and PB-LRU. These algorithms can save up to 22% more disk and provide 64% better average response time. [17]

Tola John and Idowun proposed online page replacement strategy function which is completely self-tuned. There model contains no frequency counts, unlike LFU and FBR. There algorithm doesn't suffer from rescaling requirements. Like LIRS this algorithm doesn't require potentially space bound overheads. Their model has constant time implementation complexity as opposed to the logarithmic implementation of LFU, LRU and LRFU. [4]

Jaeheon and Michel proposed several extensions of LRU which take in account non-uniform cache miss costs. Their implementation is simple but they are very effective in some cases. They first explore the cost of two static cache miss using trace driven simulations to understand the cost effectiveness ratio. They prove that large improvements of cost function are possible in many practical cases. [4]

**III. Performance and Evaluation**

In this research paper thirteen different algorithms were studied and after that, comparison has been made in the form of table which shows different performance matrix based on certain criteria. The most important factors that affect the hit ratio of cache depends on cache size, replacement algorithm and reference of a location for each cache request.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **CAR** | **LRU** | **HLFU** | **LFU** | **LLF** | **GDS** | **ARC** | **Random Replacement** | **SLRU** | **FIFO** |
| **Cache miss** | t | 4.4% on 16KB cache size |  | 769k |  |  |  | 5.0% on 16KB cache size | 293k |  |
| **Cache hit****ratio** | 42.43 % [9] | 38.41% |  | 1077k[8] |  |  | 39.17% |  | 1553k[18] | 33.8% |
| **Resource** **utilization** | High resource utilization | Need to track down Recent list , update overhead | Extra storage is require to keep history | Extra storage is require to keep history | Minimum resources are utilized | Minimum resources are utilized because block with minimum cost is replaced | High resource utilization | Almost no or very small of resources are required | Cost of segmentation  |  |
| **latency** |  |  |  |  | Calculates the average latency[18] |  |  |  |  |  |
| **queuing** | Over head of two circular buffers | Complexity fluctuate between constant and logarithmic values | Single queue is used  | Complexity fluctuate between constant and logarithmic values | Single queue is used to implement this algorithm | Single queue is used | Partition the cache into two queues |  | Single as well as circular queues are implemented  |  |
| **Maximum** **gain** | 8.75% At cache size 60 [19] | 17.4% at cache size 90 |  |  |  |  |  |  | Provides optimum result | 11.47%at cache size 120 |
| **Page** **movement** **overhead** | Very low | high |  | low | Almost low |  | Very high  |  | Very low because there are more than one segments  |  |
| **Recency/** **Frequency** | Keep track of frequency[] | Doesn't exploit [19] | Frequency variable should be there | Pay no attention to Recency | No need to tack frequency | Each block is associated with its miss ratio [17] | Content of cache don't change during transition |  | Keep track of frequency of cache elements | No need to have a frequency or Recency parameter |
| **Size** **of physical** **memory**  |  | Complexity increases when size increases[24] |  | Logarithmic implementation in cache size | With the increase in size latency also increases | Size of cache doesn't matter | Constant complexity [19] | Doesn't matter because contents has to be replaced randomly | Causes complexity because different segments has to be managed accordingly | Ignore usage pattern and paging overhead due to large size |
| **Extra** **Parameter** |  |  | Needs a parameter to controlling references | Needs a tunable parameter to controlling references[10] | Calculation of latency parameter | Cost parameter |  | Relative cost should be calculated in order to gain best results[17] | Maintenance of different segments |  |

Position figures and tables at the tops and bottoms of columns, if possible. Large figures and tables may span both columns. Figure captions should be below the figures; table captions should be above the tables. Try to place the figures and tables after their first mention in the text. Use the abbreviation (e.g., “Fig. 1”) even at the beginning of a sentence.

All half-tone illustrations (pictures/photographs) should be clear black and white prints. Do not use photocopies. These illustrations should be furnished within the copy. Make certain to include a caption in the paper for the illustration as well as to label the illustration on the back.

**V. Conclusion**

Despite the advancement in technology, different types of replacement algorithms are now being introduced. But it totally depends on the scenario in which algorithm should be implemented, there are certain factors like cost, performance, time, efficiency, accuracy which must be addressed. In above comparison table ARC gives the best result, but its cost is much high. LRU is frequently used, which is cost effective.

**References**

[1] A. J. Smith, “Cache Memories,” ACM Comput. Surv., vol. 14, no. 3, pp. 473–530, Sep. 1982.

[2] a. Bhattacharjee and B. K. Debnath, “A new web cache replacement algorithm,” PACRIM. 2005 IEEE Pacific Rim Conf. Commun. Comput. signal Process. 2005., pp. 420–423, 2005.

[3] K. Psounis, B. Prabhakar, and C. Science, “A Randomized Web-Cache Replacement Scheme,” pp. 1407–1415, 2001.

[4] N. P. Jouppi, “Cache write policies and performance,” ACM SIGARCH Comput. Archit. News, vol. 21, no. 2, pp. 191–201, May 1993.

[5] “Cache Memories Krishna M . Kavi The University of Alabama in Huntsville Cache Memories In Uniprocessors .”

[6] O. Temam, “Investigating optimal local memory performance,” ACM SIGOPS Oper. Syst. Rev., vol. 32, no. 5, pp. 218–227, Dec. 1998.

[7] S. Santhosh and W. Shi, “A Semantic-based Cache Replacement Algorithm for Mobile File Access.”

[8] A. R. Butt, C. Gniady, and Y. C. Hu, “The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms,” vol. 56, no. 7, pp. 889–908, 2007.

[9] S. M. S. Daula, K. E. S. Murthy, and G. A. Khan, “A Throughput Analysis on Page Replacement Algorithms in Cache Memory Management,” vol. 2, no. 2, pp. 126–130, 2012.

[10] S. Modha, “Outperforming LRU with an Adaptive,” 2004.

[11] P. M. H. Lipasti, “Cache Replacement Policies Cache Design : Four Key Issues ECE / CS 752 : Advanced Computer Architecture I Cache Miss Rates : 3 C ’ s [ Hill ] Optimal Replacement Policy ? Least-Recently Used Practical Pseudo-LRU ECE / CS 752 : Advanced Computer Architectu,” pp. 1–5, 2012.

[12] A. Janapsatya and A. Ignjatovi, “Dueling CLOCK : Adaptive Cache Replacement Policy Based on The CLOCK Algorithm,” pp. 2–7, 2010.

[13] A. S. Chavan, K. R. Nayak, K. D. Vora, M. D. Purohit, P. M. Chawan, and A. B. Min, “A Comparison of Page Replacement Algorithms,” vol. 3, no. 2, pp. 171–174, 2011.

[14] S. Podlipnig and L. Böszörmenyi, “A survey of Web cache replacement strategies,” ACM Comput. Surv., vol. 35, no. 4, pp. 374–398, 2003.

[15] Q. Zhu and Y. Zhou, “Power-Aware Storage Cache Management,” IEEE Trans. Comput., vol. 54, no. 5, pp. 587–602, May 2005.

[16] T. J. Odule and I. A. Osinuga, “Dynamically Self-adjusting Cache Replacement Algorithm,” vol. 6, no. 1, pp. 25–34, 2013.

[17] J. Jeong and M. Dubois, “Cost-Sensitive Cache Replacement Algorithms Jaeheon Jeong and Michel Dubois.”

[18] A. R. Butt, C. Gniady, Y. C. Hu, and W. Lafayette, “The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms,” 2005.

[19] D. Swain, B. Paikaray, and D. Swain, “AWRP : Adaptive Weight Ranking Policy,” vol. 3, no. 2, 2011.

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” *Phil. Trans. Roy. Soc. London,* vol. A247, pp. 529-551, April 1955.

1. J. Clerk Maxwell, *A Treaties on Electricity and Magnetism*, 3rd ed., Vol. 2, Oxford: Clarendon Press, 1892, pp. 68-73.
2. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,'' in *Magnetism*, Vol. III, G. T. Rado and H. Suhl, Eds., New York: Academic Press, 1963, pp. 271-350.
3. M. Smith, “Title of paper optional here,” unpublished
4. K. Rose, “Title of paper with only first word capitalized,” in press.
5. Y. Yorozu, M. Hirano, K. Oka, and Y. Tanigawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” *IEEE Trans. J. Magn.*, vol. 2, pp. 740-741, August 1987 [*Digests 9th Annual conf. Magn.*, p. 3012, 1982]