Volume 4, Issue 9, September 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Analytical Study of CAR, CLOCK and LRU Page Replacement Algorithms in Operating System

Indu Bala*
CSE, MRIU,
India

Ochin Sharma
CSE, MRIU,
India

Aftab Alam
CSE, MRIU,
India

Abstract— In a computer architecture cache memory have been introduced to balance performance and cost of the system. To improve the performance of a cache memory in terms of hit ratio and good response time system needs to employ efficient page replacement policy. And all replacement of pages has been occurred in cache memory. Page replacement algorithm decides which memory pages to page out when a page of memory needs to be allocated. Paging happens when a page fault occurs and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold. In this paper, we are comparing LRU, Clock and CAR algorithm which gives results in the efficient improvement of hit ratio and reduced response time.

Keywords— LRU, Clock, ARC, CAR

I. INTRODUCTION

Cache is high speed memory contains most recently accessed pieces of main memory. It bridges the gap between CPU and Main Memory. Increasing cache size results in better performance but it is very expensive. It is necessary because, time it takes to bring an instruction into the processor is very long when compared to the time to execute the instruction. Cache memory helps to reduce the time it takes to move information to and from the processor. Cache memory improves system performance by following a concept of Locality of Reference. The concept is that at any given time the processor will be accessing memory in a small or localized region of memory, cache memory loads this region allowing the processor to access the memory region faster. The role of cache is illustrated in the following figure 1.

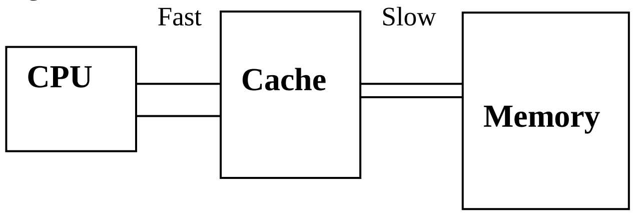

Figure 1: Cache Based Memory System

When a new page is brought into the cache, it needs to replace one of the existing pages if cache is full. For this purpose we need replacement policies. To provide memory operands to the processor at the speed it can process them is one of the most challenging aspect. To achieve high speed, an efficient replacement policy must be implemented. A number of policies have been introduced. To have maximum hit rate a good page replacement algorithm must have characteristics such as,

- Low memory overhead.
- Faster access to data.
- Low response time.

II. RELATED WORK

A. LRU: Advantages and Disadvantages

The LRU (Least Recently Used) algorithm is used widely because of its simplicity. It keeps track of the cache lines according to time they have been used. The pages which have not been used for longer time are to be replaced. The advantages of LRU are that it is extremely simple to implement, has constant time and space overhead, and captures "recency" or "clustered locality of reference" that is common to many workloads. In fact, under a certain Stack Depth Distribution (SDD) assumption for workloads, LRU is the optimal page replacement policy.

The algorithm LRU has many disadvantages:

1. On every hit to a cache page it must be moved to the most recently used (MRU) position. In an asynchronous computing environment where multiple threads may be trying to move pages to the MRU position, the MRU position is protected by a lock to ensure consistency and correctness. This lock typically leads to a great amount of contention, since all cache hits are serialized behind this lock. Such contention is often unacceptable in high

performance and high throughput environments such as virtual memory, databases, file systems, and storage controllers.

2. In a virtual memory setting, the overhead of moving a page to the MRU position on every page hit is unacceptable.
3. While LRU captures the "recency" features of a workload, it does not capture and exploit the "frequency" features of a workload. More generally, if some pages are often rerequested, but the temporal distance between consecutive requests is larger than the cache size, then LRU cannot take advantage of such pages with "long-term utility".
4. LRU can be easily polluted by a scan, that is, by sequence of one-time use only page requests leading to lower performance.

B. CLOCK as a one-bit approximation to LRU:

CLOCK removes disadvantages D1 and D2 of LRU. The algorithm CLOCK maintains a "page reference bit" with every page. When a page is first brought into the cache, its page reference bit is set to zero. The pages in the cache are organized as a circular buffer known as a *clock*. On a hit to a page, its page reference bit is set to one. Replacement is done by moving a *clock hand* through the circular buffer. The clock hand can only replace a page with page reference bit set to zero. However, while the clock hand is traversing to find the victim page, if it encounters a page with page reference bit of one, then it resets the bit to zero. Since, on a page hit, there is no need to move the page to the MRU position, no serialization of hits occurs. Moreover, in virtual memory applications, the page reference bit can be turned on by the hardware. Furthermore, performance of CLOCK is usually quite comparable to LRU

C. ARC:

Suppose that the cache can hold c pages. The policy ARC maintains a cache directory that contains 2c pages–c pages in the cache and c history pages. The cache directory of ARC, which was referred to as DBL in, maintains two lists: L1 and L2. The first list contains pages that have been seen only once recently, while the latter contains pages that have been seen at least twice recently. The list L1 is thought of as "recency" and L2 as "frequency". A more precise interpretation would have been to think of L1 as "short-term utility" and L2 as "long-term utility". The replacement policy for managing DBL is: Replace the LRU page in L1, if |L1| = c; otherwise, replace the LRU page in L2. The policy ARC builds on DBL by carefully selecting c pages from the 2c pages in DBL. The basic idea is to divide L1 into top T1 and bottom B1 and to divide L2 into top T2 and bottom B2. The pages in T1 and T2 are in the cache directory, while the history pages in B1 and B2 are in the cache directory but not in the cache. The pages evicted from T1 (resp. T2) are put on the history list B1 (resp. B2). The algorithm sets a target size p for the list T1. The replacement policy is simple: Replace the LRU page in T1, if |T1| >= p; otherwise, replace the LRU page in T2. The adaption comes from the fact that the target size p is continuously varied in response to an observed workload. The adaption rule is also simple: Increase p, if a hit in the history B1 is observed; similarly, decrease p, if a hit in the history B2 is observed. This completes our brief description of ARC.

D. CAR:

Our policy CAR is inspired by ARC. Hence, for the sake of consistency, we have chosen to use the same notation as that in [10] so as to facilitate an easy comparison of similarities and differences between the two policies. For a visual description of CAR, see Figure 2. We now explain the intuition behind the algorithm. For concreteness, let c denote the cache size in pages. The policy CAR maintains four doubly linked lists: T1, T2, B1, and B2. The lists T1 and T2 contain the pages in cache, while the lists B1 and B2 maintain history information about the recently evicted pages. For each page in the cache, that is, in T1 or T2, we will maintain a page reference bit that can be set to either one or zero. Let T^0_1 denote the pages in T1 with a page reference bit of zero and let T^1_0 denote the pages in T1 with a page reference bit of one. The lists T^0_1 and T^1_1 are introduced for expository reasons only–they will not be required explicitly in our algorithm. Not maintaining either of these lists or their sizes was a key insight that allowed us to simplify ARC to CAR.

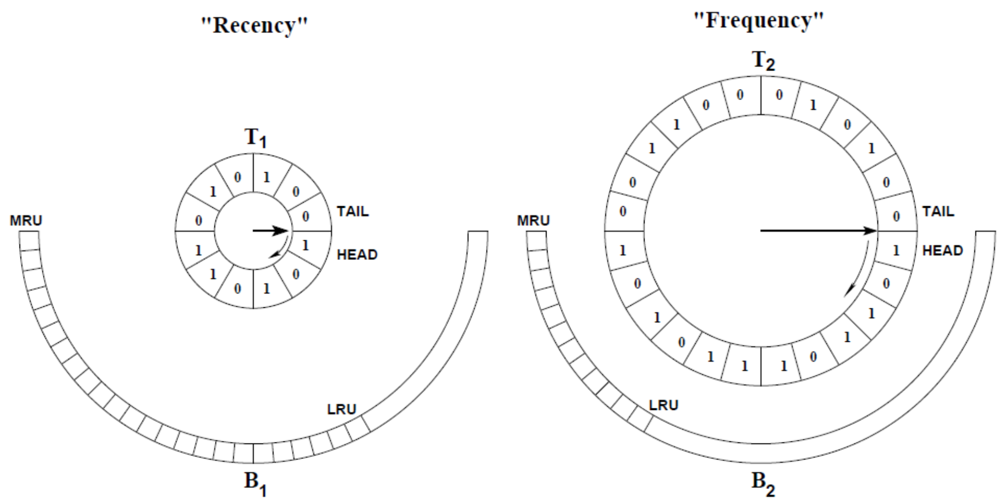

Figure 2. A visual description of CAR [25]

The CLOCKS T1 and T2 contain those pages that are in the cache and the lists B1 and B2 contain history pages that were recently evicted from the cache. The CLOCK T1 captures "recency" while the CLOCK T2 captures "frequency." The lists B1 and B2 are simple LRU lists. Pages evicted from T1 are placed on B1, and those evicted
From T2 are placed on B2. The algorithm strives to keep B1 to roughly the same size as T2 and B2 to roughly the same size as T1. The algorithm also limits |T1| + |B1| from exceeding the cache size. The sizes of the CLOCKs T1 and T2 are adapted continuously in response to a varying workload. Whenever a hit in B1 is observed, the target size of T1 is incremented; similarly, whenever a hit in B2 is observed, the target size of T1 is decremented. The new pages are inserted in either T1 or T2 immediately behind the clock hands which are shown to rotate clockwise. The page reference bit of new pages is set to 0. Upon a cache hit to any page in T1 ∪ T2, the page reference bit associated with the page is simply set to 1. Whenever the T1 clock hand encounters a page with a page reference bit of 1, the clock hand moves the page behind the T2 clock hand and resets the page reference bit to 0. Whenever the T1 clock hand encounters a page with a page reference bit of 0, the page is evicted and is placed at the MRU position in B1. Whenever the T2 clock hand encounters a page with a page reference bit of 1, the page reference bit is reset to 0. Whenever the T2 clock hand encounters a page with a page reference bit of 0, the page is evicted and is placed at the MRU position in B2.
The cache history replacement policy is simple as well:
If |T1| + |B1| contain exactly c pages, then remove a history page from B1, else remove a history page from B2.
Once again, for a better approximation to ARC, the cache history replacement policy should have been: If $|T^0{}_1|$ + |B1| contain exactly c pages, then remove a history page from B1, else remove a history page from B2. However, this would require maintaining the size of $T^0{}_1$ which would require additional processing on a hit, defeating the very purpose of avoiding lock contention.

III. COMPARISIONS AMONG LRU, CLOCK AND CAR PAGE REPLACEMENT ALGORITHMS

In our dissertation, we have compared three page replacement algorithms – CAR, CLOCK and LRU. CAR policy outperforms LRU and how it combines best features of CLOCK and ARC by removing all the disadvantages of LRU. The best page replacement policy is one which will generate maximum hit ratio
Drawbacks of LRU and Clock :-

A. LRU

D1 On every hit to a cache page it must be moved to the most recently used (MRU) position. In an asynchronous computing environment where multiple threads may be trying to move pages to the MRU position, the MRU position is protected by a lock to ensure consistency and correctness. This lock typically leads to a great amount of contention, since all cache hits are serialized behind this lock. Such contention is often unacceptable in high performance and high throughput environments such as virtual memory, databases, file systems, and storage controllers.
D2 In a virtual memory setting, the overhead of moving a page to the MRU position on every page hit is unacceptable.
D3 While LRU captures the "recency" features of a workload, it does not capture and exploit the "frequency" features of a workload. More generally, if some pages are often rerequested, but the temporal distance between consecutive requests is larger than the cache size, then LRU cannot take advantage of such pages with "long-term utility".
D4 LRU can be easily polluted by a scan, that is, by sequence of one-time use only page requests leading to lower performance.

B. CLOCK

i. It has no frequency factor like LRU
ii. Susceptibility to scan
iii. Low performance

C. Advantages of CAR

i. CAR removes cache hit serialization problem of LRU and ARC.
ii. CAR has very low overhead on cache hits and is simple to implement
iii. CAR is self-tuning and has high performance
iv. CAR is scan-resistant and has low space overhead less than 1%

D. Language and Tools Used

To compare CAR, LRU and Clock page replacement algorithm, we have chosen C programming language and MS-Visual Studio10.
We compared the performance of above three algorithms with different cache sizes. The obtained hit ratio depends on the replacement algorithm, cache size and the locality of reference for cache requests.

Hit Ratio = Total No. of Hit Count / Total No. of Reference Count

Table1 compares CAR hit ratio to the hit ratios of LRU and CLOCK. All hit ratios are cold starts and are reported in percentages. From the table1, CAR page replacement policy is better replacement policy than the other two replacement policies – LRU and CLOCK.

Table1: Data Table for Workload

Cache Size	**CAR (Hit %)**	**CLOCK (Hit %)**	**LRU (Hit %)**
1	0	0	0
2	0	0	0
3	9.71	3.88	3.88
4	23.30	6.80	5.83
5	34.95	14.56	12.62
6	39.81	23.30	22.33
7	48.54	31.07	29.13
8	56.31	33.98	35.92
9	61.17	44.66	39.81
10	62.14	45.63	44.66
11	62.14	49.51	48.54
12	62.14	54.37	50.49

In figure 3, we graphically compare the hit ratios of CAR to the CLOCK and LRU. The performance of CAR is better than CLOCK and LRU. It can be clearly seen in a workload and cache sizes CAR outperforms CLOCK and LRU – sometimes quite dramatically.

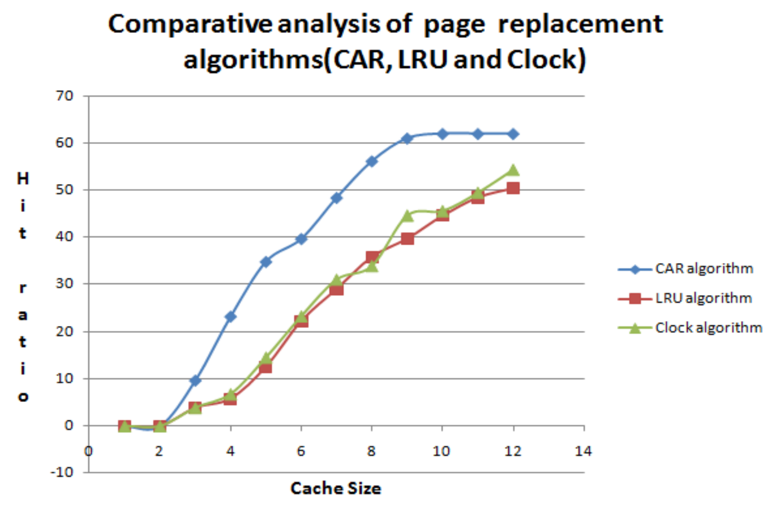

Figure 3: Workload Graph

IV. CONCLUSIONS

A page replacement policy is considered as efficient if it is able to exploit any type of reference regularities which improves hit ratio. In this paper, we compare the CAR, CLOCK and LRU page replacement algorithms. And by comparing we conclude that CAR generate maximum hit ratio among all three algorithms.CAR removes the cache hit serialization problem of LRU and ARC. The CAR attempts to merge the adaptive policy of ARC with the implementation efficiency of CLOCK. The self-tuning nature of CAR makes it very attractive for deployment in environments where no a priori knowledge of the workloads is available. CAR is scan-resistant. A scan is any sequence of one-time use requests.

REFERENCES

[1] Nimrod Megiddo Dharmendra S. Modha "Outperforming LRU with an Adaptive Replacement Cache Algorithm "IBM Almaden Research Center, 0018-9162/04 2004 IEEE58 Computer.

[2] Namrata Dafre, Urmila Shrawankar and Deepak Kapgate "Pattern Based Cache Management Policies" International Journal of Computer Sciences and Engineering ,Vol.-2(2), pp (28-35) Feb 2014, E-ISSN: 2347-2693

[3] Yifeng Zhu, Member, IEEE, and Hong Jiang, Member, IEEE" A Robust Adaptive Caching Strategy for Buffer Cache" Digital Object Indentifier 10.1109/TC.2007.70788 0018-9340 2007 IEEE

[4] S.M. Shamsheer Daula, Dr.K.E Sreenivasa Murthy,G Amjad Khan " A Throughput Analysis on Page Replacement Algorithms in Cache Memory Management" International Journal of Engineering Research and Applications(IJERA),Vol.2,Issuue2,Mar-Apr 2012,pp.126-130,ISSN:2248-9622

[5] A. S. Sumant, and P. M. Chawan, ―Virtual Memory Management Techniques in 2.6 Linux kernel and challenges, IASCIT International Journal of Engineering and Technology, pp. 157- 160, 2010.

[6] D. Lee et al., ―LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies, IEEE Trans. Computers, vol. 50, no.2, 2010, pp. 1352-1360.

[7] Y. Zhou and J.F. Philbin, ―The Multi-Queue Replacement Algorithm for Second-Level Buffer Caches, Proc. Usenix Ann. Tech. Conf. (Usenix 2010), Usenix, 2010, pp. 91-104.

[8] W.W. Hsu, A.J. Smith, and H.C. Young, The Automatic Improvement of Locality in Storage Systems, tech. report, Computer Science Division, Univ. of California, Berkeley, 2010.

[9] N. Megiddo and D.S. Modha, ―ARC: A Self- Tuning, Low Overhead Replacement Cache, Proc. Usenix Conf. File and Storage Technologies (FAST 2009), Usenix, 2010, pp. 115-130.

[10] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, ―LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies, IEEE Trans. Computers, vol. 50, no. 12, pp. 1352–1360, 2009.

[11] Sorav Bansal and Dharmendra S. Modha -CAR: Clock with Adaptive Replacement, Stanford University, IBM Almaden Research Center.

[12] Amit S. Chavan, Kartik R. Nayak, Keval D. Vora, Manish D. Purohit and Pramila M. Chawan - A Comparison of Page Replacement Algorithms, IACSIT International Journal of Engineering and Technology, Vol.3, No.2, April 2011

[13] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies," IEEE Trans. Computers, vol. 50, no. 12, pp. 1352–1360, 2001.

[14] A. Janapsatya, A. Ignjatovic, J. Peddersen and S. Parameswaran, "Dueling CLOCK: Adaptive cache replacement policy based on the CLOCK algorithm", Design, Automation and Test in Europe Conference and Exhibition, pp. 920-925, 2010.

[15] S. Jiang, and X. Zhang, "LIRS: An Efficient Policy to improve Buffer Cache Performance", IEEE Transactions on Computers, pp. 939-952, 2005.

[16] S. Jiang, X. Zhang, and F. Chen, "CLOCK-Pro: An Effective Improvement of the CLOCK Replacement", ATEC '05 Proceedings of the annual conference on USENIX Annual Technical Conference, pp. 35, 2005.

[17] A. S. Sumant, and P. M. Chawan, "Virtual Memory Management Techniques in 2.6 Linux kernel and challenges", IASCIT International Journal of Engineering and Technology, pp. 157-160, 2010.

[18] LI Zhan-sheng, et.al.,"CRFP: A Novel Adaptive Replacement Policy Combined the LRU and LFU", IEEE 8th International Conference on Computer and Information Technology Workshops, 2008 .

[19] Yifeng Zhu, Hong Jiang,"RACE: A Robust Adaptive Caching Strategy for Buffer Cache", IEEE Transaction on computers, 2007 .

[20] Urmila Shrawankar, Reetu Gupta, "Block Pattern Based Buffer Cache Management", The 8th International Conference on Computer Science and Education, April 26-28 , Colombo, 2013 .

[21] Reetu Gupta, Urmila Shrawankar, "Managing Buffer Cache by Block Access Pattern", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 6, November 2012 .

[22] A. Jaleel, C. Jean, S. C. Steely, "ShiP: Signature Based Hit Predictor for High Performance Caching", ACM International symposium on Computer Architectur, pg 430-431 , 2011 .

[23] Hou Fang, zhao Yue-long, Hou fang, "A cache management algorithm based on page miss cost", in proceedings of International conference on Information Engineering and computer science, ICIECS, ISBN: 978- 1-4244-4994-1 pp. 1-4, 2009 .

[24] Y. Zhou and J.F. Philbin, ―The Multi-Queue Replace-ment Algorithm for Second-Level Buffer Caches,‖ Proc. Usenix Ann. Tech. Conf. (Usenix 2010), Usenix, 2010, pp. 91-104.

[25] Amit S.Chavan, Kartik R. Nayak – "The Comparision Of Page Replacement Algorithms", in proceeding of International Journal of Engineering and Technology, IACSIT, Vol.3, No.2, April 2011.