Hybrid (LRU) Page-Replacement Algorithm

Pooja Khulbe
M.Tech CSE
Uttarakhand Technical university
Dehradun, India

Shruti Pant
M.Tech CSE
Uttarakhand Technical university
Dehradun, India

ABSTRACT

In this paper, a new page replacement algorithm has been represented that has a better performance in average than quondam methods. A major attempt in such an algorithm is to substitute characteristics of some quondam methods engrossed by a new idea. The key concept of proposed scheme is to decrease overall page fault rate for system i.e. to increase the hit ratio of pages. In brief this paper remonstrate an advanced version of Least recently used algorithm , which is referred as Hybrid LRU. The general idea behind Hybrid LRU is to keep track of the total number of references and whenever a page is scrawled (modified to) , a modified bit M will be set for that particular page.

Keywords

Hit ratio, LRU Algorithm, Page Replacement, HYBRID(LRU) Page replacement algorithm

1. INTRODUCTION

The most important part of the operating system is memory management. Main memory is divided into fixed size units called page frames. Each victimized page can be either in secondary memory or in main memory as page frames. The virtual address space [3] is divided into fix size blocks called pages and this division is done by operating system.

A CPU generated address is called logical address or virtual address, where as memory management unit generated address is known as physical address. Before using this logical address, it must be translated to its corresponding physical address. This address translation has been done corresponding to every memory reference, so it is important that it must be fast. A special hardware unit referred to as **Memory Management Unit (MMU)**, is used for such translation. MMU uses address mapping information which is usually located in page tables, to make the translation. If the given virtual address is not mapped to main memory, operating system is trapped by the MMU. This trap is called as **page fault** which gives an opportunity to the operating system to bring the desired page from secondary memory to main memory, and then update the page table correspondingly. In simple words we can say that-**When the processor need to execute a particular page and main memory does not contain that page , this situation is known as PAGE FAULT.** As each and every process has its own virtual address space, the operating system must keep track of all pages and the location of each page used by each process. Whenever any page in main memory is referenced or written to, it is marked accordingly. When a page fault occurs (known as cache miss), the operating system eliminates some page to secondary memory to make space for the incoming page. Whenever a cache miss occurs, the operating system applies page replacement algorithm to choose a page from cache for replacement or eviction to make place for the referenced page. When the selected page is modified while it is in cache (also called as dirty), it must be again written to the RAM also known as writeback . If any page will not modified, no writeback will be need to the RAM, which results less overhead.

A **paging algorithm or page replacement algorithm[4]** is needed to manage paging. A paging algorithm evicts pages from and to the page table when it becomes full. Many algorithms has been developed for such a swapping in which the best-case scenario is to be replaced the page that is not going to be used for the longest time. Since it is difficult to predict the future reference, a better way to choose the correct algorithm is by looking at characteristics of the processes. It is necessary to influence which page should be replace. Evicting a page that may be required in close future can degrade the system performance; because it imposes a reloading time overhead.

Virtual memory system requires effective page replacement algorithms in case of a page fault, to decide which pages should be expelled from memory. Since long many algorithms have been proposed for page replacement. Each algorithm tries to reduce the page fault rate while acquiring minimum overhead.

2. PREVIOUS WORK

2.1 Page Replacement Algorithm

In a computer operating system paging is used for virtual memory management and **page replacement algorithms** are used to decide which memory pages to page out (swap out, write to disk) when a page of memory requires to be allocated. Paging encounters when a page fault occurs and a free page cannot be used to fulfill that allocation, either because there are no free pages, or because the number of free pages is less than some brink.

When any page that was hand-picked for eviction from memory and paged out, is referenced again it has to be rewrite in memory and this includes waiting for I/O culmination.
There are a variety of page replacement algorithms. Some of them are described as follows:-

2.1.1 First in, first out (FIFO)

The first-in first-out algorithm[2] is the simplest and oldest algorithm. The idea behind FIFO is to replace a page that is the oldest page in main memory from all the pages. ‘Replace the page which has been resident from longest period of time.’ FIFO focuses on the length of a time a page has been in memory rather than how much the page is being used. Figure.1 illustrates an example of the FIFO algorithm. Here it is important to note that since the page table is initially empty; directly three page faults appear to fill the table and after that, a page fault occurs only when any required page is not present in the table at that time and so on.

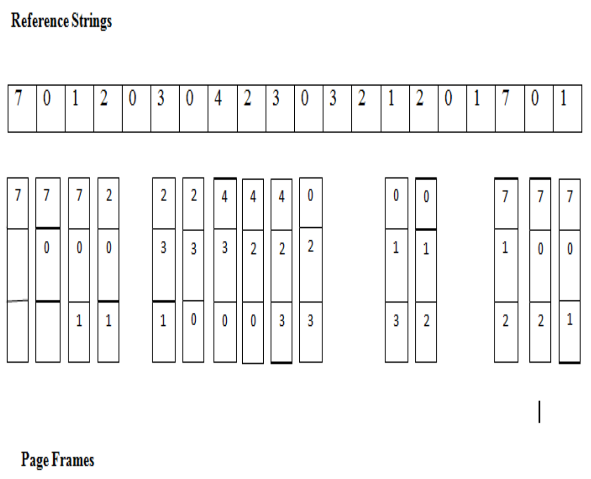

Figure.1: FIFO representation for 3 frames

2.1.2 *The Optimal Algorithm*

Among all page replacement an algorithm, the optimal algorithm has the best performance i.e. has the lowest page fault rate of all the algorithms but its implementation is not realistic till now. In this algorithm, whenever any new page is requested, if there is any vacant frame, requested page will be allocate to that free frame. But the main problem is to choose a frame to be evicted when there is no empty frame. This algorithm simply replace the page that will not be used for the longest period of time. So it is difficult to implement because it requires future knowledge of strings.

.

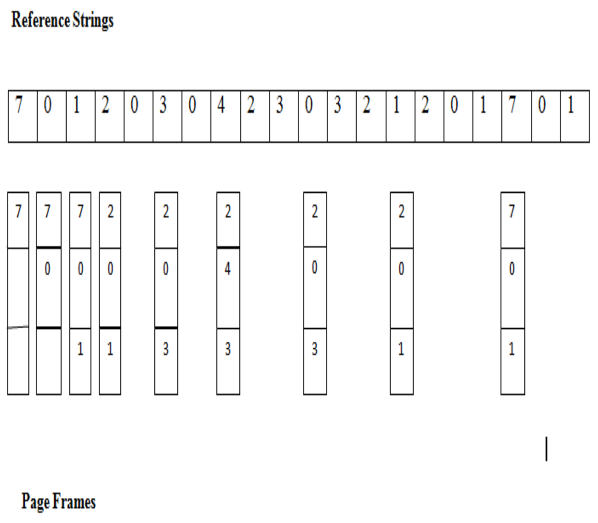

Fig.2: optimal algorithm representation for 3 frames

However, one of reasons for designing such an algorithm, while it is not relevant practically, is its utilization as an ideal reference for assessing and comparing practical(and not theoretical) algorithms. Fig.2 shows execution of optimal algorithm by considering a random page invoking sequence.

2.1.3 *Least-recently-used (LRU)*

This algorithm[2] replaces the page that has not been used for the longest period of time. In general LRU algorithm[1] results better than FIFO algorithm. The reason behind this is that LRU takes into account the patterns of program behavior by presuming that in most distant past used page is slightest likely to be used again later. The least-recently-used algorithm looks backward in the period of time. However, implementation of the LRU algorithm is now possible but it imposes large overhead to the system.

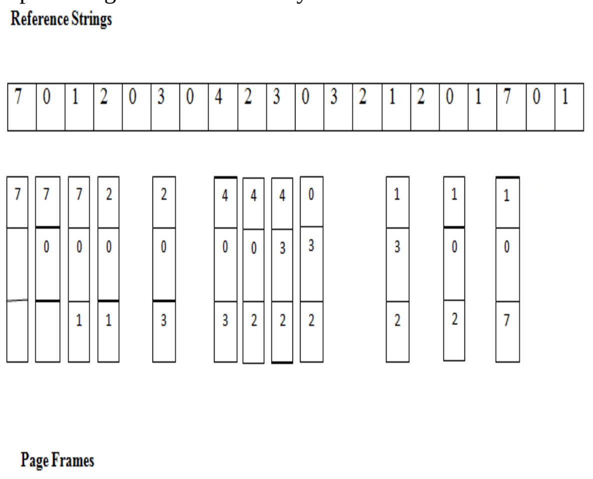

Figure.3: LRU representation for 3 frames

3. PRELIMINARIES

A **cache** is a component that stores data transparently [5] so that requests for that data can be served faster in future. The data that is stored within a cache might be values that have been computed earlier or duplicates of original values that are stored somewhere else. When requested data already exists in the cache (**cache hit**), this request can be completed by merely reading the cache, which is faster in comparison of eviction or allocation of pages. Otherwise (**cache miss**), the data has to be recomputed or fetched from its original storage location, which is comparatively slower. Hit Ratio, is the ratio of the total number of hits to the number of all transactions.

Hit ratio = number_of_requests_that_hit_cache / total_number_of_requests.

# or to represent it as a percentage:

percentage_hit = ratio * 100

# for example:

Let us assume that:-

number_of_requests_that_hit_cache = 12

number_of_requests = 100

ratio = number_of_requests_that_hit_cache / total_number_of_requests

ratio == 0.12

4. PROPOSED ALGORITHM (HYBRID LRU)

The proposed algorithm is based on LRU and will be referred to as Hybrid LRU. This algorithm uses an extra feature that is **total number of references** for each page which will be counted on each referred page. And as well as it uses the concept of **modified reference**, when a page is modified (written to), a modified reference is set i.e. M=1 for that page. Using following steps proposed algorithm can be easily understood:-

Step 1. when a page fault occurs, it will investigate the first parameter, TNR for all pages. If only one page found with minimum TNR value, that page will be selected to evict from memory.

Step 2. Modified reference is contained in each page table entry, and the algorithm initializes it with zero i.e. M =0, for all pages. When the contents of any page change, M will be set, M=1 for that page.

Step 3. If the minimum TNR value is shared between special fractions of pages and M=0 , treat it same as LRU.

Step 4. if the minimum TNR value is shared between special fractions of pages and any one of them is modified then always choose the modified page i.e. **the page with M =1 again for replacement.**

Step 5. if the minimum TNR value is shared between special fractions of pages and more than one or all of them are modified then replace the recently modified page for replacement. And clear the modified bit for each table entry i.e. again set M =0 for each page.

Note- whenever an already modified page will be replaced again, modified bits for each table entry will be set to 0 i.e M=0.

Step 6. Repeat step 1 to 5 until all pages completes their traversing.

Note: In order to represent modification bit, we use the sign ☐ when M=1.

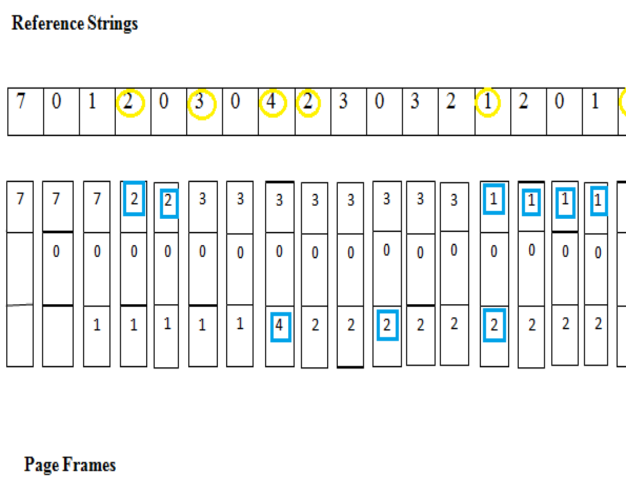

Figure.4: H-LRU representation for 3 frames

5. EXPERIMENTAL RESULTS

In order to assess the proposed algorithm (with respect to page faults and hit ratio) and comparing it with the other algorithms, they have been simulated using MATLAB. In this experiment, hit ratio is reasoned as the compare standard. For this purpose, 20 random sequences with length 20 were imposed to LRU and HYBRID LRU, as page arousing sequence. In following experiment four cases for memory frame number, include 2-frame , 3- frame, 4- frame, 5- frame were reasoned and outcomes have been reported in Table.1 Based on this table, the bar diagram for these results can be shown as chart 1.

Table 1. Comparision of hit ratio between lru and hlru

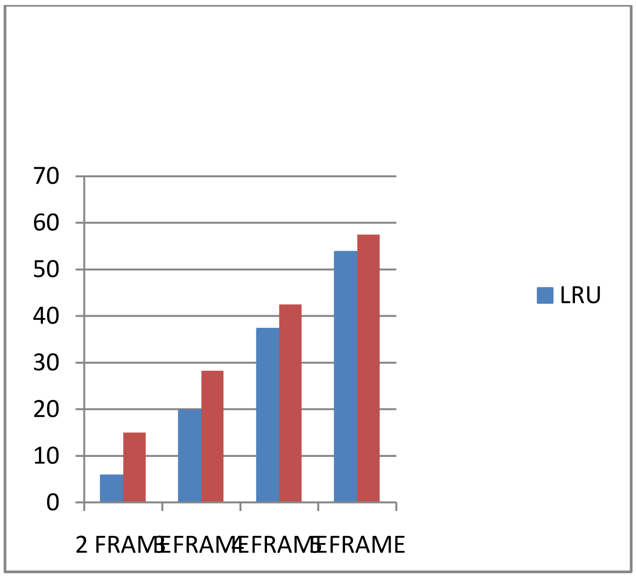

Chart 1: Bar graph representing comparision between lru and hlru

	Hit Ratio(%)			
	2 Frame	3 Frame	4 Frame	5 Frame
LRU	5.75	19.75	37.25	53.75
H-LRU	15	28.25	42.5	57.5

In the second experiment, again the Hit ratio has been reasoned as the compare standard. For this purpose, again 20 random sequences with length 20 were imposed to LRU and HYBRID LRU. For each frame size, the rate of invoking Hit ratio for HYBRID- LRU is equal to LRU Hit ratio, is reported. The same is repeated for HYBRID-LRU< LRU and HYBRID- LRU > LRU. In following experiment four cases for memory frame number include 2-frame, 3- frame, 4- frame, 5- frame were reasoned and outcomes have been reported in Table.2 Based on this table, the bar diagram for these results can be shown as chart 2.

Table 2. Comparision of success rate between lru and hlru

	in(%)			
	2 Frame	3 Frame	4 Frame	5 Frame
H-LRU=LRU (Equal)	25	10	20	35
H-LRU<LRU (Failure)	5	20	20	10
H-LRU>LRU (Success)	70	70	60	55

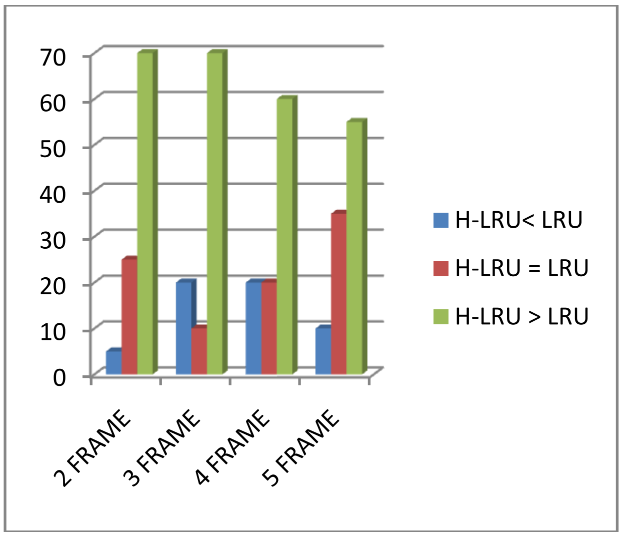

Chart 2: Bar graph representing comparisons between lru and hlru

6. CONCLUSION

Studying the page replacement algorithms has suggested that the LRU has had the best results among all practical algorithms, and the upcoming researches have focused on this algorithm, to meliorate the performance of system and to diminish the page fault rate. In this paper a page replacement algorithm based on LRU is represented. This algorithm used a complementary parameter beside the LRU parameter, to determining the page i.e. TNR (total no. of references) that must be replaced with a new one with the concept of modified reference.

7. REFERENCES

[1] O’Neil, J. E., O’Neil, E. P., Weikum, G., "An Optimality Proof of the LRU-K Page Replacement Algorithm", Journal of the ACM, Vol. 46, No. 1, pp. 92-112, January 1999.

[2] Ali Khosrozadeh, Sanaz Pashmforoush, Abolfazl Akbari, Maryam Bagheri, Neda Beikmahdavi., “Presenting a Novel Page Replacement Algorithm Based on LRU” , Journal of Basic and Applied Scientific Research , 2(10)10377-10383, 2012.

[3] Kim, K., Park, K., "Least Popularity – Per – Byte Replacement Algorithm for a Proxy Cache ", IEEE, pp. 780 – 787,2001.

[4] S.M. Shamsheer Daula, Dr. K.E Sreenivasa Murthy, G Amjad Khan., “A Throghput Analysis on page replacement algorithm”,International Journal of Engineering Research and Applications (IJERA), ISSN: 2248-9622, Vol. 2, Issue 2, pp.126-13, Mar-Apr 2012.

[5] Jaafar Alghazo, Adil Akaaboune, and Nazeih Botros. Sf-lru cache replacement algorithm. In MTDT, pages 19-24. IEEE Computer Society, 2004

[6] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong-Sang Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. Computers, 50(12):1352-1361, 2001.

[7] Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall, 1992