An Assessment of Hybrid LRU (H-LRU) with Existing Page Replacement Algorithms

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

Sanjay Kumar
Assistant Professor
Uttarakhand Technical university
Dehradun, India

Nidhi Rawat
M.Tech CSE
Uttarakhand Technical university
Dehradun, India

ABSTRACT
The goal of any page replacement algorithm is to reduce fault rate by selecting best victim page to remove. This paper presents a framework through which, we tried to compare our previous work i.e. H-LRU [1] with other existing algorithms and prove that it has a better performance in average than former methods. Our major attempt in this paper is to extend our previous work HLRU and to compare characteristics of some former methods with HLRU. The general idea behind the comparison is to evaluate the hit ratio of pages for different algorithms and to prove that it has the best performance among all .For this purpose we consider traditional algorithm like FIFO, LRU and some recent approaches like PRO-LRU, H-LRU.

Keywords
Page Replacement, LRU, FCFS, PRO-LRU, H-LRU,

1. INTRODUCTION
Managing the memory in a computer system entails two or more levels of memory to be organized. Among these levels, one with shortest access time is selected as the first level, also called as primary memory. A major method for organizing the memory space and allocating the limited space between the applications to be accomplished was the memory segmentation. But, because of the phenomenon of external fragmentation capitulated to evidently wasting the memory space, this method was swapped with the paging method. Also, the latent memory wasting was the reason for the paging method to be inefficient. So a method based on a grouping of these two models was introduced to settle their problems. In a paging basis, the obtainable memory space is divided into blocks called as page frames. Each application asking the memory is subdivided into some pages and will be given a number of page frames to enclose some of these pages. This number is resolved by the operating system and can be different for applications. Until the memory encloses at least one empty page frame, assigning the frames for the applications and loading the pages into them is simple, relatively. But, the main problem in organizing the frames will be evolved when all available frames have been assigned to the applications and there are no empty frames. The solution for this, will be ejecting one or more pages from the memory and free some frames to be assigned for new pages. It is important to decide which page(s) must be ejected. Evicting a page can influence the system performance that may be required in close future because it inflicts a reloading time overhead.

Page replacement is one of the most valuable methods of keeping page faults. This process basically happens when there is no gratis page in the physical memory. The superiority of **page replacement algorithms** have a significant affect on the working of the process of replacing pages. Like all other aspects of computer science, the concept of page replacement algorithms also underwent evolution and development. It was particularlly in the span of 1960s and 1970s that this concept receive the maximum attention.

A **page replacement algorithm** can be local or global. When the process calls upon a page fault, the local algorithm chooses a replacement page that belongs to the same processor, a set of processes sharing a memory partition. Whereas a global algorithm can select a replacement page from anywhere in the system memory. Pre-cleaning is an important part of page replacement algorithms and have to be achieved in case the selected page is dirty (or, written upon).

2. REPLACEMENT ALGORITHMS
There are many different types of replacement algorithms. Some of which are described below:

2.1 First-in-first-out (FIFO)
The simplest page-replacement algorithm is a FIFO algorithm (first in first out). The first-in, first-out (FIFO) [5] page replacement algorithm is a less-overhead algorithm that entails little book-keeping on the part of the operating system. The idea is obvious from the name - the operating system keeps track of each page in memory in a queue, with the latest arrival at the back, and the earliest arrival in front. The operating system sustains a list of all pages presently in memory, with that page which is at the head of the list the oldest one and the page at the tail the most topical arrival. When a page needs to be swapped, the page at the front of the queue (the oldest page) is considered. While FIFO is cheap and instinctive, it results poorly in practical application.

This algorithm suffers from some drawbacks. As the first, if a page is used frequently in multiple time periods, it will be acknowledged as the last or oldest page, ultimately and may be picked to be moved out from the memory, while there is a believable probability for vital need to it. In such these cases, the selection will be incompetent; since the swapped page must be reloaded into memory almost immediately [8].

Another disadvantage for this algorithm narrates to this fact that improving the memory frames designated for a process can capitulate to a lower page fault rate.

2.2 Least Recently Used (LRU)

The LRU policy is based on the principle of locality which states that program and data references within a process tend to cluster. The LRU page replacement policy chooses that page for replacement which has not been used for the longest time. For a long time, LRU was deliberated to be the most optimum online policy.

LRU while being operative, is again, not without problems. The first drawback among them is the fact that it is very costly to implement it. In fact, the most costly method in linked with LRU, which facilitates in attaining what it is meant to. This can be a factor for not choosing this algorithm. The second problem with this approach is the difficulty in implementation. LRU policy does nearly as well as an optimal policy, but it is intricate to implement and imposes significant overhead.

The LRU is based on the observation that pages that have been used a lot in the last few instructions will probably be utilized a lot again in the next few. Contrarily, pages that have not been utilized for ages will probably remain unused for longest period of time. This idea suggests a realizable algorithm: when a page fault takes place, evict the page that has been unused for the longest period of time.

There are a few implementation methods for this algorithm that attempt to diminish the cost yet keep as much of the performance as possible. The most costly method is the linked list method, which utilizes a linked list enclosing all the pages in memory. At the front is the most recently used page and at the back of this list is the least recently used page, which is a very time-consuming process. LRU's weakness is that its performance tends to degenerate under many quite common reference patterns. On the other hand one important advantage of the LRU algorithm is that it is agreeable to full statistical analysis.

2.3 Pro- LRU

In this section, a LRU based algorithm is introduced, and is referred to as PRO_LRU [2]. The first and most important note about this algorithm is using an extra feature TNR (Total no. of references), which is to count total number of references and for selecting the outgoing pages, together with this feature, an extra feature is also used by LRU which is referred as STR (spent time since last reference).

Whenever any page fault takes place, the first parameter, TNR, will be investigated for all pages. If there is onlya single page with minimum TNR value, this page will be evicted from memory. Or else, if the minimum TNR value is divided between no. of pages, the second parameter came on existence. In the other word, a page with most STR values with minimum TNR value will be selected for eviction.

2.4 Hybrid LRU:

Hybrid LRU also referred as HLRU is also an extension of LRU. It also uses the extra feature TNR (**total number of references)** for each encountered page and as a modification it uses the concept of **modified reference**, i.e. when a page is modified, a modified reference, M=1 will always be set for that page. when a page fault occurs, it examines the first parameter, TNR for each page. If only one page is found, it immediately replace it. Else if there is more than one page is available with minimum TNR, it checks modified reference for those pages and replaces the page which is recently modified i.e. M=1.

Each time when a modified page is re-modified all modified references for each page will be set to 0 by default.

3. PERFORMANCE ANALYSIS

To evaluate page replacement algorithms experimentally, our previous algorithm i.e. H-LRU [1] has been implemented in C for 25 random strings of length 20 for 2, 3 , 4 and 5 frames correspondingly and then same strings has been evaluated for other policies like FIFO, LRU and finally PRO-LRU for 2 , 3, 4 and 5 frames.

The obtained page fault rate depends on the replacement algorithm, frame size and the locality of reference for cache requests. Using total count of page faults and hits for each algorithm, hit ratio has been calculated in average for 25 random strings and listed in table 1.

Table1. Comparison of hit ratio for various algorithm

Frame Size	FIFO (25 strings)	LRU (25 strings)	Pro-LRU (25 strings)	H-LRU (25 strings)
2 Frame	40.93	41.82	41.33	40.24
3 Frame	49.26	48.86	50.05	49.65
4 Frame	57.48	56.69	58.67	59.27
5 Frame	62.14	64.62	66.5	66.2

Hit ratio can be calculated using the formula as follows-

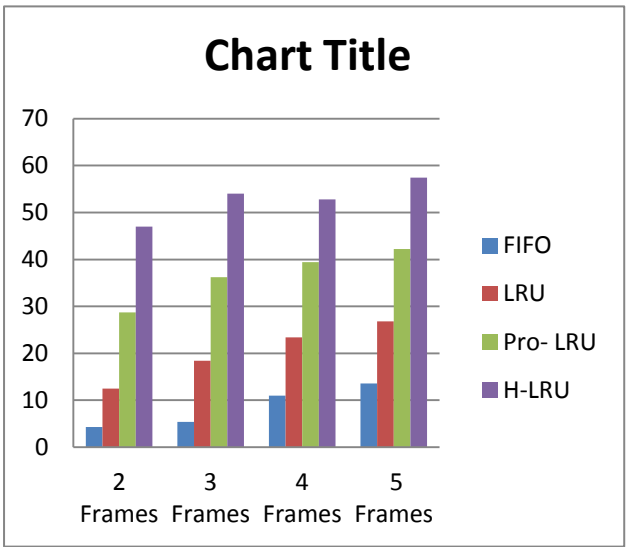
$$Hit\ Ratio = \frac{Total\ Number\ of\ Hit\ Counts}{Total\ Number\ of\ Reference\ Counts}$$

For example suppose the total no. of hits for an algorithm is 7 and the total no. of references are 20 then total Hit Ratio will be:

Hit Ratio = 7/20 = 0.35
(In percentage) = 35 %

The bar graph representing the comparison of various algorithms has been presented in chart1.

Chart1. Bar graph for hit ratio of various algorithms



4. RESULTS

Diagram 1.1 represents the screen shot of H-LRU for a string of length 20 for 2 frames, While implementing H-LRU in C.

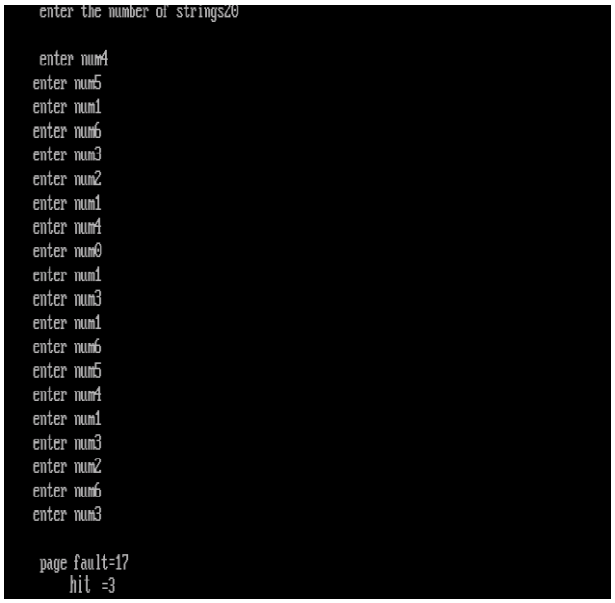


Figure1.1: Screenshot of h lru for 2 frames

5. CONCLUSION

In this paper we have discussed various famous page replacement policies like FIFO, LRU, PRO-LRU and H-LRU, then implemented and compared them to evaluate their efficiency and it is clear that HLRU has a best performance among all in average. It makes easier to choose a specific policy for a specific set of memory reference. It also explains the variant characteristics of different algorithms, which helps us to characterize their behavior and development of new page replacement techniques in future development.

6. REFERENCES

[1] Pooja khulbe, Shruti pant, "HYBRID LRU Page Replacement Algorithm" , *International Journal of Computer Applications (0975 – 8887) Volume 91 – No.16, April 2014*

[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] Kaveh Samiee, "WRP: Weighting Replacement Policy to Improve Cache Performance", *International Journal of Hybrid Information Technology,Vol.2,No.2, April, 2009*.

[4] 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.

[5] 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*