ISSN No. 0976-5697

Volume 3, No. 3, May-June 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

LFNRU: A New Hybridized Page Replacement Algorithm for Efficient Memory Management and its Performance Analysis

H. S. Behera
Department of Computer Science & Engineering,
Veer Surendra Sai University of Technology (VSSUT)
Burla, Sambalpur, Orissa, India, 768018
hsbehera_india@gmail.com

B.Namadipta Patro
Department of Computer Science & Engineering,
Veer Surendra Sai University of Technology (VSSUT)
Burla, Sambalpur, Orissa, India, 768018
namadipta.patro@gmail.com

Ishita Bhuyan
Department of Computer Science & Engineering,
Veer Surendra Sai University of Technology (VSSUT)
Burla, Sambalpur, Orissa, India, 768018
ishitabhuyan@gmail.com

Abstract: This paper introduces a new dimension to the existing page replacement algorithms. Combining the LEAST FREQUENTLY USED (LFU) and LEAST RECENTLY USED (LRU) paging algorithms we hereby present a hybridized algorithm that effectively reduces the number of page faults incurred in the former two algorithms. Basically the idea is to compare the counter values and replace the page with a lower counter count and in cases of similar counts the page that has the lesser recent reference is evicted. As we demonstrate with simulation experiments, the "LFNRU-Least Frequently Not Recently Used" algorithm surpasses the performance and efficiency of LFU, LRU, LRU-K AND RLRU page replacement algorithms. Furthermore, the LFNRU algorithm adapts in real time to changing patterns of access.

Keywords: virtual memory, buffer, frame size, LFU, LRU

I. INTRODUCTION

For computer operating systems using paging for virtual memory management, paging algorithms decide which memory pages are to be paged out when a page of memory needs to be allocated. Paging happens due to the occurrence of page fault and a condition in which free pages are unable to satisfy the allocation, either because there are none, or because the number of free pages is lowers than some benchmark.

To address these issues elaborately and coherently we have organized the page into five different sections which separately attends to a distinct issue. We start up with the problem definition that familiarizes us with sub-divisions of memory. The next section covers all the paging algorithms invented till date. Next we have a section named related works that throws light on all the significant research done in this field so far. Then comes the main body of the project that encompasses our entire work. Here we present our algorithm and introduce LFNRU- least frequently and not recently used algorithm. We proceed further by placing forth tabulation and graphs that illustrate the performance of the algorithms. We have taken meticulously taken different cases and strings of varied lengths to further prove our point. We finally wind up by putting forth a comparative analysis discussing the aspects of each and every paging algorithm.

A. Problem Definition:

There are two levels of memory. Firstly, the fast access device. Secondly, the larger, slower, backing store. These levels are divided into page frames. If a program references a page in the second level, a page fault is encountered. So, this page is to be brought into the first level and a page from the first level is to be transferred to the second level. This process is known as page replacement. It is implemented by algorithms called page replacement algorithms.

The objective of every page replacement algorithm is to minimize the number of page faults encountered and reduce fragmentation.

B. Well Known Paging Algorithms:

a. Least Freqeuntly Used(Lfu):-

LFU paging algorithm produces the page in cache that has the least references in the past. However, the demerit in this case is that it is unable to adapt rapidly to fluctuating patterns in input. It has an improved performance over LRU. LFU defines a frequency of use associated with each page. The frequency implementation starts at the beginning of the page reference string and continuous to reckon the frequency over an ever increasing interval. Basically, reactions to locality transitions will be extremely slow. The variant of LFU accounts for frequency count of a page since it was last loaded rather than since the beginning of the page reference string.

b. Least Recently Used(Lru):-

LRU algorithm swaps the page in cache that had been last referenced. Here the disadvantage is that the algorithm does not distinguish between pages that are requested for very frequently and those that are not. LRU is unable to differentiate pages based on frequency. . The Self-correcting

LRU tries to improvise the replacement policy in the cache. The three major mistakes made by LRU are:

a) ***Bypass block:*** Many blocks are accessed only once, at the time of the miss and then they are not accessed again until they leave the cache. Such blocks are called Bypass block.
b) ***Deadlock:*** Blocks which are referenced more than once and then are not referenced are called Dead block.
c) ***Live block:*** When a block is accessed right after it was replaced is called live block.

LRU replaces a page that has not been referenced for maximum time. Most of the paging algorithms use the past as the prediction of the future to choose the page to be swapped. To improvise LRU an approach has been made to adaptive page replacement algorithm like LRU-WAR and LRU-Warlocks. The LRU-WAR is based on the maximum working set size between consecutive page faults. But it falters in a global memory management system. In LRU-Warlocks one part of memory is reserved for most referenced pages and other part keeps the original LRU-WAR management.

c. LRU-K:-

This algorithm looks at the kth latest request in the cache.

d. RLRU:-

Retrospective least frequently used algorithm which chooses from pages that are unmarked. A page is marked in two cases:
-RLRU has a fault
-RLRU has a hit and the page is different from pages of previous request.

e. Optimal:-

When a page is required to be swapped in, the operating system swaps the page whose next use will occur after the longest time. This algorithm has not found practical implementation in the general purpose operating system since it is insurmountable to compute reliably the time it will take before a page is going to be used, exempting cases when all software that will run on a system is either known beforehand and is necessary to the static analysis of its memory reference patterns, or only a class of applications allowing run-time analysis. Inspire of this limitation, algorithms exist that can offer nearly optimal performance. The operating system keeps track of all pages referenced by the program, and uses that data to decide which pages to swap in and out on upcoming runs. This algorithm can offer near-optimal performance, but not on the first run of a program, and only if the program's memory reference pattern is relatively consistent each time it runs.

f. Not Recently Used:-

The not recently used (NRU) page replacement algorithm is that paging algorithm which favors keeping pages in cache that have been recently used. This algorithm works on the principle that when a page is used, a referenced bit is set for that page, notching it as referenced. Accordingly, when a page is modified, a modified bit is set. The setting of the bits is usually done by the hardware. When a page needs to be replaced, the operating system divides the pages into four classes:

a) Class 0: Not referenced, not modified
b) Class 1: Not referenced, modified
c) Class 2: Referenced, not modified
d) Class 3: referenced, modified

g. FIFO:-

This algorithm uses the first-cum-first-serve base. It can be implemented using either a clock or a FIFO queue to replace the oldest page.

h. Second Chance:-

Second chance algorithm is an improved form of FIFO page replacement algorithm. It notches a reference bit with each page. The page whose reference bit is not set is selected for replacement. A page whose reference bit is set is cleared and then inserted at the tail of the queue.

i. Clock:-

It marks the oldest page with a hand. On the occurrence of a page fault, the reference bits of the page pointed by the hand are checked. If it 0, the page is selected. Else it is cleared. Subsequently, the clock hand is increased and the process is iterated until a page is replaced.

j. Random:-

Randomly, a page is chosen and replaced. This does away with the overhead cost of tracking page references. Generally it furnishes better results than FIFO, and in cases of looping memory references it is better than LRU, although in practice LRU performs better . OS/390 uses global LRU approximation and falls back to random replacement when LRU performance degenerates and the Intel i860 processor used a random replacement policy.

k. Not Frequently Used:-

The NFU paging algorithm uses a counter and every page has its own counter which is initially set to 0. At every clock interval, all pages that have been referenced within that clock interval will have their counter raised by 1. The counters keep track of how frequently a page is being used. Hence, the page with the lowest counter can be swapped out. The biggest problem with Not Frequently Used is that it keeps track of the frequency of use rather than focusing on the time span of use. For example, in case of a multi-pass compiler, pages with the highest frequency in the first pass, but are not need in the second pass will be prioritized over pages which have comparatively lower frequency count in the second pass. This results in poor performance. There are certain other cases where NFU performs in the same manner like in an OS boot-up. The NFU paging algorithm produces fewer page faults than the LRU paging algorithm when the page table contains null pointer values.

l. Most Frequently Used:-

It is based on the concept that the page with the smallest count is yet to be used.

m. Aging:-

It accounts for the use time span . The reference counter on a page is shifted right before adding the reference bit to the left of that binary number. This is done without basing on time.

n. Working Set:-

This algorithm focuses on the assumption of locality. It is a set of pages expected to be used by that process during certain time interval.

II. RELETAED WORK

The paper by Elizabeth J. O'Neil and Patrick E. O'Neil and Gerhard Weikum presents a revised approach to database disk buffering namely the LRU-K method. The LRU-K keeps track of the times of the last *K* references to popular database pages. The information statistically estimates the inter-arrival times of references on a page by page basis. The understanding was further enhanced by following the manuscript of R.I.Phelps and L.C.Thomas.[1][2]

"An optimal performance in self-organizing paging algorithms" by R.I.Phelps and L.C.Thomas illustrates that an optimal self organizing paging algorithm can efficiently reduce the number of page faults and thereby minimize the cost by taking into consideration the probable posterior reference. [3]

"LRU-K page replacement algorithm" by Prof. Shahram presents several motivations and alternatives to LRU-K. It also takes into account its design and implementation. [4]

Silberschatz, Galvin and Gagne, "Operating System Concepts" has been referred to at each and every step of the course of the project for clarification of basic fundamentals. [5]

III. OUR CONTRIBUTION

Having keenly analyzed the existing page replacement algorithms, considering the loop falls in each of the above mentioned algorithms, we design an algorithm that is a conjunction of LRU and LFU page replacement algorithms.

A. Proposed Algorithm:

We take an array where the least occurrence will be stored and the page that satisfies the following properties will be replaced:-
-page with the least counter value
-page that stays for the longest period of time

The pseudo code of the proposed algorithm along with the corresponding flowchart displaying accurately the flow of control for the entire program is shown.

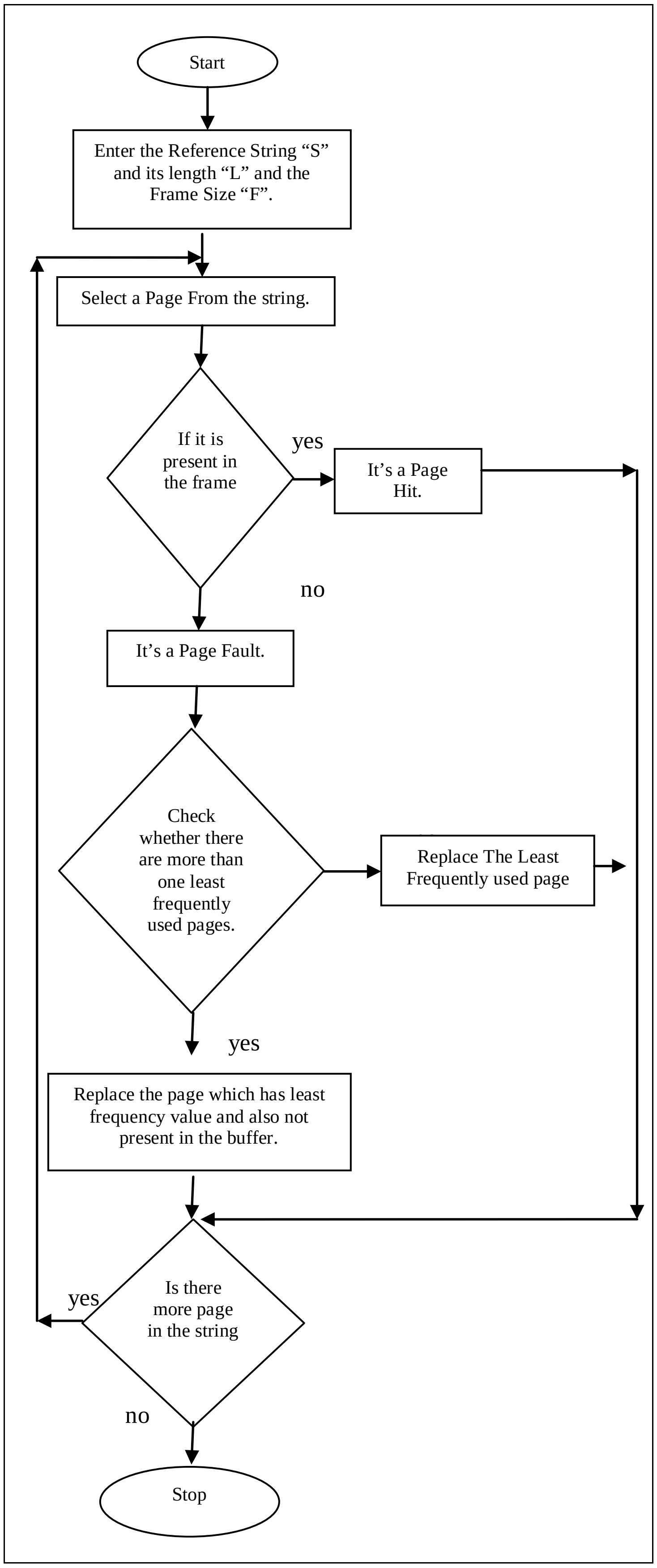
B. Pseudo Code Of The Proposedalgorithm:

```
Step: 1> Enter the Reference String "S" and its length
"L" and the Frame Size "F".Initialise the buffer.
Step: 2>repeat step: 3, 4 for L times.
Step: 3>select a page from the string.
Step: 4>
    If (page is already in the Frame) then
    {
            It's a Hit.
        If buffer is full go to step 5.
        Else insert in to buffer.
    }
  Else
  {
  /*page is not present in the Frame*/
  /*check the least frequently used page*/

            If (there is more than one least frequently
                    used page) {
          Replace the page which is not in the buffer.
        If buffer is full go to step 5.
        Else insert in to buffer.
            }
          Else if (there are more than one least
frequently used pages and both are in buffer)
            {
      Replace the page which stays in buffer for a
      long      time.
        If buffer is full go to step 5.
        Else insert in to buffer.
            }
            Else
            {
            Replace the least frequently used page.
        If buffer is full go to step 5.
        Else insert in to buffer.
            }
  }

Step: 5>replace the page which comes to the buffer
first.
                Return;
Step: 6>exit.
```

C. Flow Chart Of The Proposed Algorithm:


Start
Enter the Reference String "S" and its length "L" and the Frame Size "F".
Select a Page From the string.
If it is present in the frame
yes
It's a Page Hit.
no
It's a Page Fault.
Check whether there are more than one least frequently used pages.
Replace The Least Frequently used page
yes
Replace the page which has least frequency value and also not present in the buffer.
Is there more page in the string
yes
no
Stop

D. An Experimental Analysis Of Proposed Algorithm:

1 1 2 3 1 7 6 7 6 7 2 1 3 2 7 6

Least Recently Used:-

1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
-	-	2	2	2	7	7	7	7	7	7	7	3	3	3	6
-	-	-	3	3	3	6	6	6	6	6	1	1	1	7	7

Least Frequent Not Recently Used:-

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-	-	2	2	2	7	7	7	7	7	7	7	7	7	7	7
-	-	-	3	3	3	6	6	6	6	2	2	3	2	2	6

Least Frequently Used:-

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-	-	2	2	2	7	6	7	6	7	2	2	2	2	7	6
-	-	-	3	3	3	3	3	3	3	3	3	3	3	3	3

Extensively, computing for several paging algorithms taking strings and frames of different sizes, we get the following output.

Table: 1 Using Frame Size 3

Input String	*LFU*		*LRU*		*LFNRU*	
	Page fault	*Page fault rate*	*Page fault*	*Page fault rate*	*Page fault*	*Page fault rate*
1 3 1 2 3 4 5 1 2 3 4 5 1 2 1 1	11	68.75	13	81.25	9	56.25
1 1 2 3 1 7 6 7 6 7 2 1 3 2 7 6	11	68.75	10	62.5	9	56.25
1 2 3 4 5 6 7 8 7 8 5 6 7 8 5 6 1 2 3 4 5 6 7 8 1 2 5 6 5 6	27	90	28	93.33	22	73.333
1 2 3 4 5 6 7 8 1 7 8 3 5 6 1 2 7 1 8 5 6 8 7 1 2 3 4 5 6 3	25	83.33	26	86.66	24	80
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 3 0 32 1 2 0 1 0 2	19	63.33	16	53.33	13	43.333

The above tabular representation provides the following results. We realize that LFNRU gives the best results.
Taking the string 1312345123451211 in a frame size of 3 LFNRU outperformed by LRU and LFU.
Taking the string 1 1 2 3 1 7 6 7 6 7 2 1 3 2 7 6 in a frame size of 3LFNRU outperforms LFU and LRU.

Taking the strings1 2 3 4 5 6 7 8 7 8 5 6 7 8 5 6 1 2 3 4 5 6 7 8 1 2 5 6 5 6 in a frame size of 3 LFNRU outperforms LFU and LRU.

Taking the string 1 2 3 4 5 6 7 8 1 7 8 3 5 6 1 2 7 1 8 5 6 8 7 1 2 3 4 5 6 3in a frame size of 3 LFNRU outperforms LFU and LRU.

Taking the string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 3 0 32 1 2 0 1 0 2 LFNRU outperforms LFU and LRU.

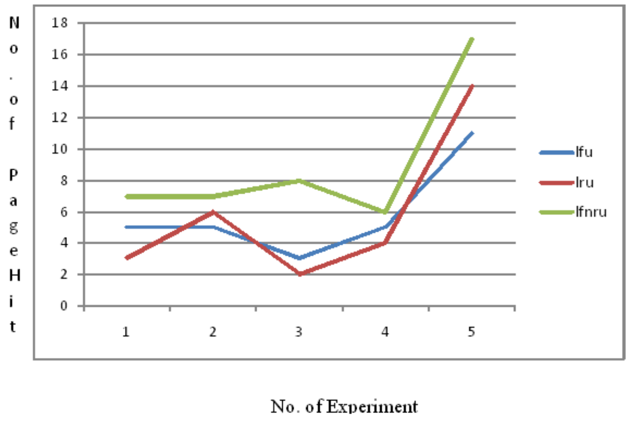


Figure: 1 Graphical representation of page hit using frame size 3.

Accordingly, for clarity we express the performance of LFU, LRU and LFNRU using linear graph. The graph explains the efficiency comparison of the three paging algorithms. With the exception of few cases where LFNRU is outperformed by LRU, in the remaining cases we notice that if not the best, LFNRU and LFU perform equivalently well. However we realize that in 5 cases LFNRU outperforms LFU and LRU.

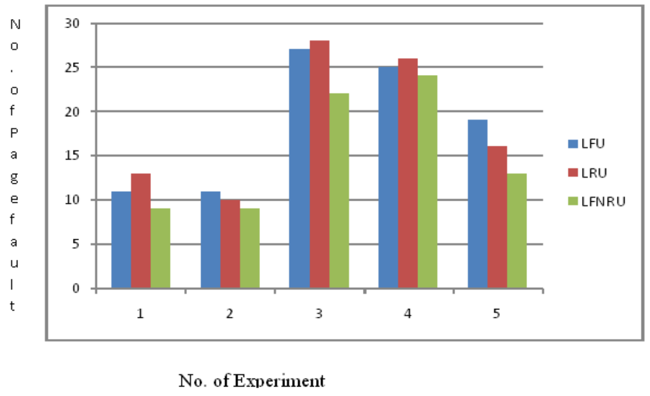


Figure: 2 Graphical representation of page fault t using frame size 3.

The adjacent bar graph shows the comparative behavior of the standard paging algorithms of LFU and LRU with LFNRU. It is clearly evident that taking a frame size of 3, in certain cases LFU, LRU and LFNRU perform with similar efficiency. In certain other cases we notice that even though LRU outperforms LFNRU, it performs as well as LFU. We also notice that in several cases LFNRU gives the best results.

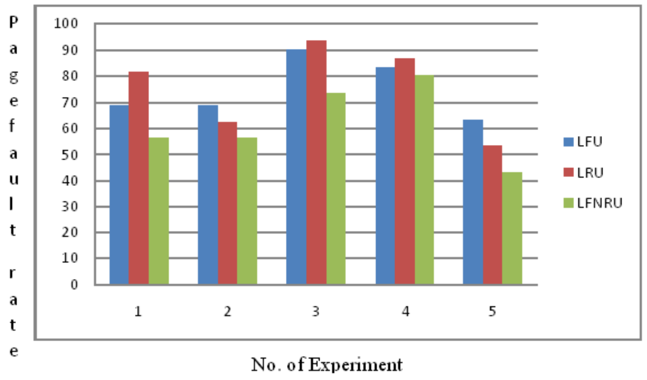


Figure: 3 Graphical representation of page fault rate using frame size 3.

The above graph shows the page fault rate encountered in each case. Page fault rate is defined as the rate of page faults. It can be said to be the number of miss per number of pages.

=>PAGE FAULT RATE = (NUMBER OF MISS) / (NUMBER OF PAGES)

=>PAGE FAULT RATIO= [(NUMBER OF PAGE FAULT)/(NUMBER OF PAGES)]*100

So as is evident, lower the page fault rate better the algorithm. In the above graph we can clearly see that though in the case of the first three strings the performances remain equal, in the subsequent cases LFNRU outperforms the performances of LFU and LRU.

Table: 2 Using Frame Size 4

Input String	***LFU***		***LRU***		***LFNRU***	
	Page fault	***Page fault rate***	***Page fault***	***Page fault rate***	***Page fault***	***Page fault rate***
1 3 1 2 3 4 5 1 2 3 4 5 1 2 1 1	8	50	12	75	8	50
1 1 2 3 1 7 6 7 6 7 2 1 3 2 7 6	8	50	8	50	7	43.75
1 2 3 4 5 6 7 8 7 8 5 6 7 8 5 6 1 2 3 4 5 6 7 8 1 2 5 6 5 6	26	86.66	20	66.66	15	50
1 2 3 4 5 6 7 8 1 7 8 3 5 6 1 2 7 1 8 5 6 8 7 1 2 3 4 5 6 3	24	80	25	83.33	22	73.33

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 3 0 32 1 2 0 1 0 2	10	33.33	12	40	9	30

The above tabular representation provides the following results. We realize that LFNRU gives the best results.
Taking the string 1312345123451211 in a frame size of 4 LFNRU outperformed by LRU and LFU.
Taking the string 1 1 2 3 1 7 6 7 6 7 2 1 3 2 7 6 in a frame size of 4 LFNRU outperforms LFU and LRU.
Taking the strings1 2 3 4 5 6 7 8 7 8 5 6 7 8 5 6 1 2 3 4 5 6 7 8 1 2 5 6 5 6 in a frame size of 4 LFNRU outperforms LFU and LRU.
Taking the string 1 2 3 4 5 6 7 8 1 7 8 3 5 6 1 2 7 1 8 5 6 8 7 1 2 3 4 5 6 3in a frame size of 4 LFNRU outperforms LFU and LRU.
Taking the string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 3 0 32 1 2 0 1 0 2 LFNRU outperforms LFU and LRU.

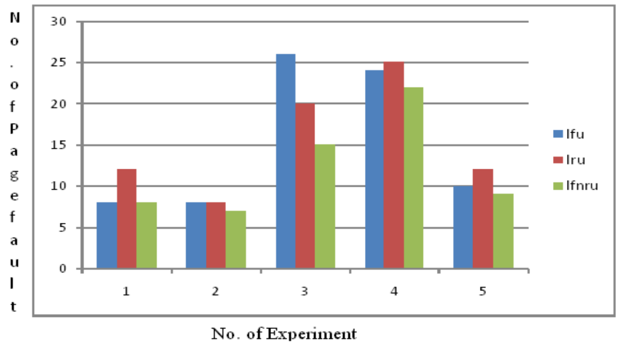


Figure: 4 Graphical representation of page fault using frame size 4.

The adjacent bar graph shows the comparative behavior of the standard paging algorithms of LFU and LRU with LFNRU. It is clearly evident that taking a frame size of 4, in certain cases LFU, LRU and LFNRU perform with similar efficiency. In certain other cases we notice that even though LRU outperforms LFNRU, it performs as well as LFU. We also notice that in several cases LFNRU gives the best results

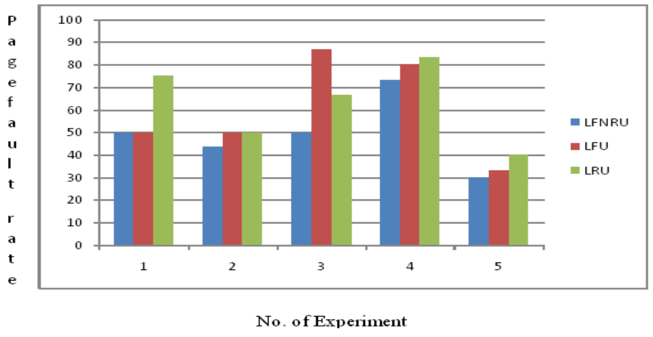


Figure: 5 Graphical representation of page fault rate using frame size 4.

The above graph shows the page fault rate encountered in each case. Page fault rate is defined as the rate of page faults. It can be said to be the number of miss per number of pages.

=>PAGE FAULT RATE = (NUMBER OF MISS) / (NUMBER OF PAGES)

=>PAGE FAULT RATIO= [(NUMBER OF PAGE FAULT)/(NUMBER OF PAGES)]*100

So as is evident, lower the page fault rate better the algorithm. In the above graph we can clearly see that though in the case of the first three strings the performances remain equal, in the subsequent cases LFNRU outperforms the performances of LFU and LRU.

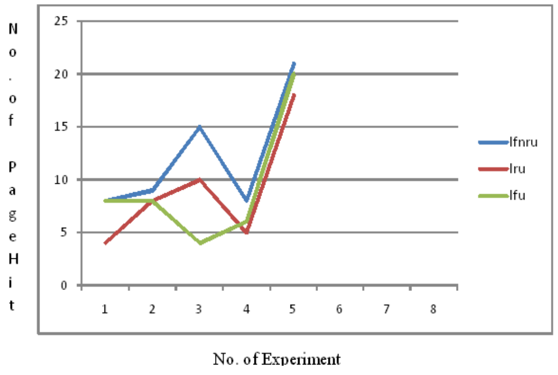


Figure: 6 Graphical representation of page hit using frame size 4.

Accordingly, for clarity we express the performance of LFU, LRU and LFNRU using linear graph. The graph explains the efficiency comparison of the three paging algorithms. With the exception of few cases where LFNRU is outperformed by LRU, in the remaining cases we notice that if not the best, LFNRU and LFU perform equivalently well. However we realize that in 5 cases LFNRU outperforms LFU and LRU.

Table: 3 Using Frame Size 5

Input String	***LFU***		***LRU***		***LFNRU***	
	Page fault	***Page fault rate***	***Page fault***	***Page fault rate***	***Page fault***	***Page fault rate***
1 3 1 2 3 4 5 1 2 3 4 5 1 2 1 1	5	31.25	5	31.25	4	31.25
1 1 2 3 1 7 6 7 6 7 2 1 3 2 7 6	5	31.25	5	31.25	4	31.25
1 2 3 4 5 6 7 8 7 8 5 6 7 8 5 6 1 2 3 4 5 6 7 8 1 2 5 6 5 6	20	66.66	20	66.66	14	46.67
1 2 3 4 5 6 7 8 1 7 8 3 5 6 1 2 7 1 8 5 6 8 7 1 2 3 4 5 6 3	21	70	22	73.33	19	63.33
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 3 0 32 1 2 0 1 0 2	7	23.33	7	23.33	6	20

The above tabular representation provides the following results. We realize that LFNRU gives the best results.

a. Taking the string 1312345123451211 in a frame size of 5 LFNRU outperformed by LRU and LFU.
b. Taking the string 1 1 2 3 1 7 6 7 6 7 2 1 3 2 7 6 in a frame size of 5 LFNRU outperforms LFU and LRU.
c. Taking the strings1 2 3 4 5 6 7 8 7 8 5 6 7 8 5 6 1 2 3 4 5 6 7 8 1 2 5 6 5 6 in a frame size of 5 LFNRU outperforms LFU and LRU.
d. Taking the string 1 2 3 4 5 6 7 8 1 7 8 3 5 6 1 2 7 1 8 5 6 8 7 1 2 3 4 5 6 3in a frame size of 5 LFNRU outperforms LFU and LRU.
e. Taking the string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 3 0 32 1 2 0 1 0 2 LFNRU outperforms LFU and LRU.

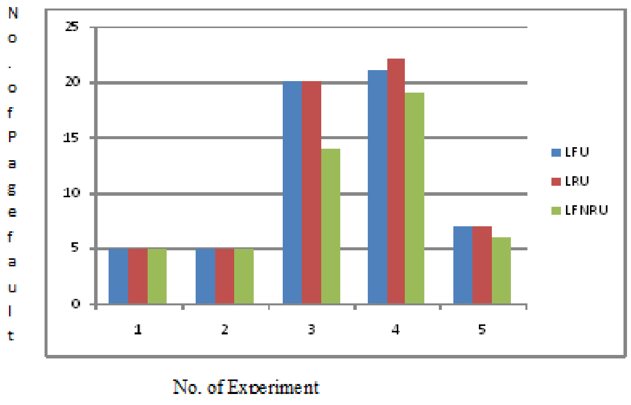


Figure: 7 Graphical representation of page fault using frame size 5.

The adjacent bar graph shows the comparative behavior of the standard paging algorithms of LFU and LRU with LFNRU. It is clearly evident that taking a frame size of 4, in certain cases LFU, LRU and LFNRU perform with similar efficiency. In certain other cases we notice that even though LRU outperforms LFNRU, it performs as well as LFU. We also notice that in several cases LFNRU gives the best results.

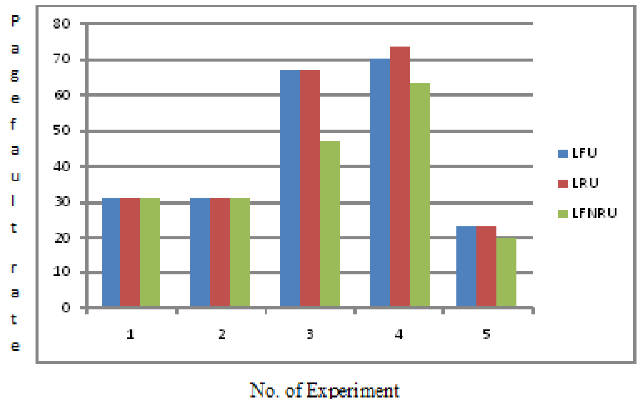


Figure: 8 Graphical representation of page fault rate using frame size 5.

The above graph shows the page fault rate encountered in each case. Page fault rate is defined as the rate of page faults. It can be said to be the number of miss per number of pages.

=>PAGE FAULT RATE = (NUMBER OF MISS) / (NUMBER OF PAGES)

=>PAGE FAULT RATIO= [(NUMBER OF PAGE FAULT)/(NUMBER OF PAGES)]*100

So as is evident, lower the page fault rate better the algorithm. In the above graph we can clearly see that though in the case of the first three strings the performances remain equal, in the subsequent cases LFNRU outperforms the performances of LFU and LRU.

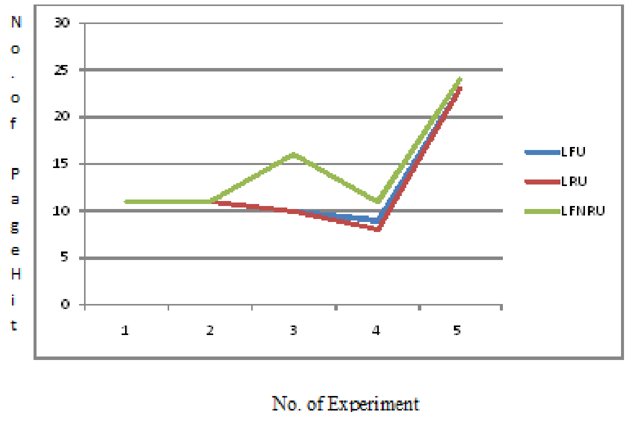


Figure: 9 Graphical representation of page hit using frame size 4.

Accordingly, for clarity we express the performance of LFU, LRU and LFNRU using linear graph. The graph explains the efficiency comparison of the three paging algorithms. With the exception of few cases where LFNRU is outperformed by LRU, in the remaining cases we notice that if not the best, LFNRU and LFU perform equivalently well. However we realize that in 5 cases LFNRU outperforms LFU and LRU.

Table: 4 Performance Analysis Table

NAME	*PAGE REPLACEMENT ALGORITHM*
1.LFU	Crude and hence non-adaptive
2.LRU	Rules out frequency as a parameter and suffers from bypass, livestock and deadlock issues.
3.LRU-k	Excessively parameterizes on time aspect.
4.RLRU	It shows resistance to sequential scans.
7.FIFO	Suffers from starvation problem.
8.LFNRU	Shows promising results compared to standard algorithms.

IV. CONCLUSION

We have compared the performance and efficiency of LFU, LRU and LFNRU page replacement algorithms. We have experimentally verified their performance and noted their performances graphically. We have seen that taking different cases, with different strings with varied frame sizes we have encountered few cases. Firstly, in all cases with frame size 3 and 4 LFNRU outperforms LRU and LFU perform the same.

Secondly, increase in buffer size renders better performance. Illustrating the organisation of the paper, we begin with the introduction to all the paging algorithms developed till date. Then we begin the computation of the efficiency of LFU, LRU and LFNRU.

We have also realized in the due course of preparing this manuscript that this algorithm can be utilized in improving the performance of search engines and browsers. Further extensions of this project can be to implement the algorithm in real life practical examples.

V. REFERENCES

[1] Elizabeth J. O'Neil and Patrick E. O'Neil and Gerhard Weikum, "AN Optimality Proof of LRU-K Page Replacement Algorithm", August 1998,pp.1-6, in press.

[2] R.I.Phelps and L.C.Thomas "AN optimal performance in self-organizing paging algorithms", pp.1–9 in press.

[3] Prof. Shahram Ghandeharizahde CSCI 485 lecture notes on "LRU-K page replacement algorithm" presentation in press.

[4] Silberschatz, Galvin and Gagne,"Operating System Concepts",8th edition, vol 4, 2010 pp.339-340.

[5] Wayne A. Wong and Jean –Loup Baer, "MODIFIED LRU Policies for Improving second level Cache Behavior",pp.-1-4,pp-9-12 in press.