Histogram based Analysis of Page Replacement Techniques

Ruchin Gupta
IT Deptt. , Ajay Kumar Garg
Engg. College, Ghaziabad

ABSTRACT

Modern operating systems use virtual memory concept because of its advantages but they use different page replacement techniques .An efficient page replacement technique is required so as to produce minimum number of page faults. Some of the page replacement techniques are FIFO, LRU, OPTIMAL etc.Optimal has been proven to be best producing minimum number of page faults .LRU approximates optimal. Considerable research has been done to evaluate theses policies and to develop new ones based on recency, frequency, token, and locality model parameters etc.This paper uses a histogram based approach to compare FIFO, LRU, LRU2, OPTIMAL policies. Simulation results show that histograms for all policies equalize as the number of frames increases. Also histogram for optimal policy equalizes more rapidly then other policy’s histograms. Also pages of large frequency of occurrences contribute much to the total number of page faults in both LRU and optimal page replacement algorithms.

General Terms

Operating system, virtual memory, simulation of page replacement techniques.

Keywords

Operating system, virtual memory, page fault, page replacement techniques, histogram, and matlab.

1. INTRODUCTION

Operating system acts as an interface between user and hardware resources of a computer system [1]. Two purpose of an OS are to provide a convenient interface to the user and efficient utilization of computer resources. Operating system is also known as a resource manager. It performs process management, I/O management, memory management, file management etc. Since memory in computer is very important for program execution, so it is very necessary to manage memory especially main memory efficiently. Now a days, almost all operating systems are multiprogramming.

The multiprogramming puts more than one program in memory to be executed simultaneously through CPU scheduling .Now for the programs bigger than main memory, earlier operating systems were using the concept of overlay drivers but it was not efficient, then after virtual memory concept came in to existence. Virtual memory is a technique which provides an illusion to a user having a very large amount of main memory available [2].It allows for program execution even the program is partially there in memory. In most of the operating systems virtual memory in implemented by demand paging. There are several advantages of using this concept like less I/O etc.

The principles behind virtual memory are as follows:

1. Load as much as possible of the process into actual/physical memory;

2. Keep a copy of the complete process (its memory image) in a disk file; this is called the swap file.

3. The virtual memory manager (in the kernel) organizes the process's (virtual) memory into chunks called *pages*; pages are normally of 512, 1024, 2048 or 4096 bytes. Some pages are in main memory, others are not, but all are in the swap file.

4. If in 1, only part of the process could be loaded, the process may run fine for a while. In this situation, the memory manager has nothing to do. All needs to be done is that the hardware that supports virtual memory management does appropriate address translation.

However, at some stage, the process will try to access a memory address that is not in physical memory. This is called a page fault. When a page fault occurs, the memory manager must read in from the swap file the page that is needed. This is not a happy situation for a process; reading a disk file takes a minimum of 10-millisec, while reading memory takes only 10-nanosec.However, where to put it. Which of the currently in memory page to replace if there is no space in memory? That is the subject of page replacement policy. There are various page replacement policies like FIFO, OPTIMAL, LRU etc.

It is well-known, however, that there are many situations where LRU behaves far from optimal [3].LRU based on recency and frequency has been implemented in cache replacement policy hence been proved to be better than simple LRU [4].

2. BASIC DEFINITIONS

2.1 First in, First out (FIFO)

With first-in-first-out, when a page is needed, the page that has been in memory for the longest period of time is chosen. The rationale is that a page that has only recently been swapped in will have a higher probability of being used again soon. However a frequently used page still gets swapped out when it gets to be old enough, even though it will have to be brought in again immediately.

2.2 Least Recently Used (LRU)

Least recently used page replacement policy is based on the assumption that the page reference pattern in the recent past is a mirror of the pattern in the near future. Pages that have been accessed recently are likely to continue to be accessed and ought to be kept in physical memory. An allocated memory page of a program will become a replacement candidate if the page has not been accessed for a certain period of time under two conditions: (1) the program does not need to access the page; and (2) the program is conducting page faults (a sleeping process) so that it is not able to access the page although it might have done so without the page faults. However, LRU page re-placement implementations do not discriminate between two types of LRU pages and treat them equally [5]. So it means that LRU can be made closer to optimal policy by making improvement in to that.

2.3 Optimal method

This policy selects a page for replacement which will be used after the longest period of time .Since this requires future knowledge of the reference string, this can not be implemented in the system. Hence this policy is used for comparative study only.

2.4 LRU k method

LRU k [6] method has been successfully applied in database disk buffering and has been shown to be better than LRU. Also LRU 2 has been found to be best for k=2. As per this policy LRU is a special case of LRU k where k=1.

The LRU-K Algorithm specifies a page replacement policy when a memory frame is needed for a new page requested: the page p to be dropped (i.e., selected as a replacement victim) is the one whose Backward K-distance bt(p,K) is the maximum of all pages in memory. The only time the choice is ambiguous is when more than one page has bt(p,K) = ∞.In this case, a subsidiary policy may be used to select a replacement victim among the pages with infinite Backward K-distance; for example, classical LRU could be employed as a subsidiary policy.

2.4.1 Backward K-distance bt(p,K)

Given a reference string known up to time t, r1, r2. . . rt, the backward K-distance bt(p,K) is the distance backward to the Kth most recent reference to the page p:

bt(p,K) = x, if rt-x has the value p and there have been exactly K-1 other values i with t - x < i <=□ □ t ,
where ri = p,
= ∞, if p does not appear at least K times in r1, r2. . . rt

2.5 Memory Reference Pattern Plot

Memory reference plot show how page numbers are being used as the application executes. X axis shows the progress of time while Y axis shows which page numbers are used at different times during the execution of an application. It shows the memory access pattern of an application during execution of that particular application.

2.6 Histogram

Histogram has been very popular in digital imaging and other subjects. A histogram for any given data set shows the frequency of occurrences of every element in the given data set. It gives a clear idea about high frequency element and low and medium frequency elements for a given data set (Data set contains similar kind of data in the given data set).

3. SIMULATOR

The proper choice of a page replacement algorithm is actually quite a complex matter. To make the proper choice, we must know something about real applications. How do they really access memory? Do they generate many page accesses in order? To answer these questions, we must see what real applications do.

In this, paper evaluates how real applications respond to a variety of page replacement algorithms. Modifying a real operating system to use different page replacement algorithms is quite a technical mess, so it will make this by simulation. We write a program that simulates the behavior of a memory system using a variety of page replacement algorithms. We obtain memory traces from real applications so that we can evaluate algorithm properly.

Here the purpose is to build a simulator that reads a memory trace and simulates the action of a virtual memory system with a single level page table in single programming model. The simulator keeps track of what pages are loaded into memory. As it processes each memory event from the trace, it should check to see if the corresponding page is loaded. If not, it should choose a page to remove from memory. Assume that all pages and page frames are 4 KB etc.

It implements different page replacement algorithms such as LRU and OPTIMAL, FIFO, LRU2. The simulator is written in plain C in MS-DOS environment

It assumes that reference string is containing six thousands references stored in an array. Numbers of frames are varied and no of page faults are calculated. Reference strings of different applications are taken as input and numbers of page faults are calculated and graphs are plotted in MS-EXCEL between no of frames vs. no of page faults. Also MATLAB is used to plot memory reference pattern plot for different applications and to plot histograms for page fault behavior of different application.

3.1 Memory Traces

Each trace obtained from Internet is a real recording of a running program, taken from the SPEC2000 benchmarks. Real traces are enormously big having billions and billions of memory accesses. However, a relatively small trace will be more than enough. Each trace only consists of one million memory accesses taken from the beginning of each program. Traces are gcc.trace.gz , swim.trace.gz , bzip.trace.gz.

Each trace is a series of lines, each listing hexadecimal memory addresses followed by R or W to indicate a read or a write. For example, gcc.trace trace starts like this:

0041f7a0 R 13f5e2c0 R 05e78900 R 004758a0 R

4. SIMULATION RESULTS

In all the tables, numbers of page faults are shown for various numbers of frames for various page replacements techniques. It assumes the page size of 4KB and reference string of size six thousand.

Table 1 (Using swim application)

No of frames	No of Page faults using FIFO	No of Page faults using LRU	No of Page faults using LRU2	No of Page faults using OPTIMAL	FIFO/ OPTIMAL	LRU/ OPTIMAL	LRU2 /OPTIMAL
5	2540	2155	2069	1574	1.613723	1.369123	1.314485
10	1949	1750	1186	939	2.075612	1.863685	1.263046
15	1558	1247	859	572	2.723776	2.18007	1.501748
20	1102	820	505	361	3.052632	2.271468	1.398892
25	754	472	395	247	3.052632	1.910931	1.59919
30	587	320	295	190	3.089474	1.684211	1.552632
35	459	253	256	156	2.942308	1.621795	1.641026
40	372	212	230	138	2.695652	1.536232	1.666667
45	318	187	208	126	2.52381	1.484127	1.650794
50	261	166	187	117	2.230769	1.418803	1.598291
55	236	149	174	112	2.107143	1.330357	1.553571
60	205	141	161	107	1.915888	1.317757	1.504673
65	184	133	153	105	1.752381	1.266667	1.457143
70	172	124	139	105	1.638095	1.180952	1.32381
75	155	118	137	105	1.47619	1.12381	1.304762
80	150	116	131	105	1.428571	1.104762	1.247619
85	140	113	123	105	1.333333	1.07619	1.171429
90	134	111	114	105	1.27619	1.057143	1.085714
95	132	107	112	105	1.257143	1.019048	1.066667
100	122	106	107	105	1.161905	1.009524	1.019048
105	105	105	105	105	1	1	1
110	105	105	105	105	1	1	1
115	105	105	105	105	1	1	1

Table 2 (Using gcc application)

No of frames	No of Page faults using FIFO (Series1)	No of Page faults using LRU (Series2)	No of Page faults using LRU2 (Series3)	No of Page faults using OPTIMAL (Series4)	FIFO/ OPTIMAL	LRU/ OPTIMAL	LRU2 /OPTIMAL
5	1709	1419	1414	1067	1.601687	1.329897	1.325211
10	1226	1034	1084	734	1.6703	1.408719	1.476839
15	997	876	977	612	1.629085	1.431373	1.596405
20	878	770	882	532	1.650376	1.447368	1.657895
25	796	714	789	468	1.700855	1.525641	1.685897
30	753	671	693	423	1.780142	1.586288	1.638298
35	722	627	642	384	1.880208	1.632813	1.671875
40	661	580	593	359	1.841226	1.615599	1.651811
45	616	530	574	336	1.833333	1.577381	1.708333
50	577	495	525	318	1.814465	1.556604	1.650943
55	560	468	501	307	1.824104	1.52443	1.631922
60	538	443	483	297	1.811448	1.491582	1.626263
65	504	426	458	292	1.726027	1.458904	1.568493
70	483	406	442	287	1.682927	1.414634	1.54007
75	455	387	425	282	1.613475	1.37234	1.507092
80	440	372	419	277	1.588448	1.34296	1.512635
85	424	359	400	272	1.558824	1.319853	1.470588
90	399	345	393	267	1.494382	1.292135	1.47191
95	394	338	380	262	1.503817	1.290076	1.450382
100	371	311	372	257	1.44358	1.210117	1.447471
105	361	326	362	252	1.43254	1.293651	1.436508
110	354	306	358	247	1.433198	1.238866	1.449393
115	350	298	351	246	1.422764	1.211382	1.426829
120	346	293	343	246	1.406504	1.191057	1.394309
125	344	289	337	246	1.398374	1.174797	1.369919
130	343	287	327	246	1.394309	1.166667	1.329268
135	339	286	324	246	1.378049	1.162602	1.317073
140	336	283	319	246	1.365854	1.150407	1.296748

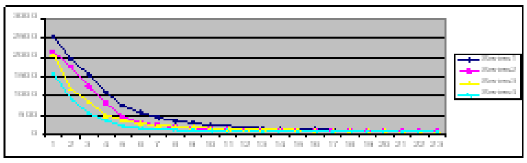
Figure 1 (for swim application)

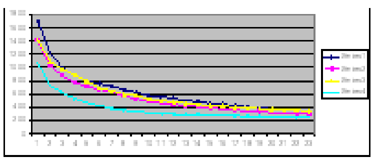
Figure 2 (for gcc application)

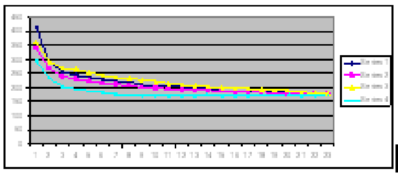
Figure 3 (for bzip application)

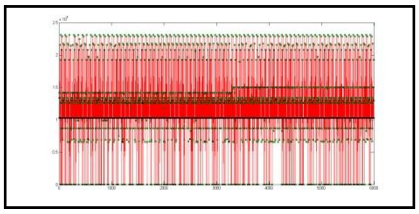
Figure 4 (memory reference pattern plot using swim app.)

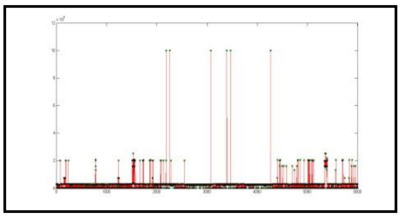
Figure 5(memory reference pattern plot using gcc app.)

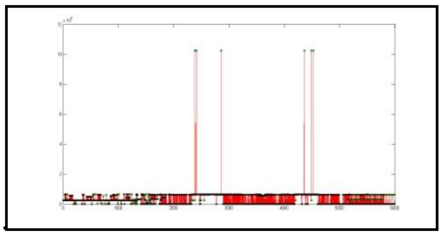
Figure 6 (memory reference pattern plot using bzip app.)

Table 3 (using bzip application)

No of frames	No of Page faults using FIFO (Series1)	No of Page faults using LRU (Series2)	No of Page faults using LRU2 (Series3)	No of Page faults using OPTIMAL (Series4)	FIFO/ OPTIMAL	LRU/ OPTIMAL	LRU2 /OPTIMAL
5	416	342	362	297	1.400673	1.151515	1.218855
10	290	270	295	233	1.244635	1.158798	1.266094
15	258	240	270	206	1.252427	1.165049	1.31068
20	244	231	265	196	1.244898	1.178571	1.352041
25	237	226	255	189	1.253968	1.195767	1.349206
30	229	216	244	184	1.244565	1.173913	1.326087
35	223	213	238	179	1.24581	1.189944	1.329609
40	221	209	234	174	1.270115	1.201149	1.344828
45	210	202	229	171	1.22807	1.181287	1.339181
50	208	199	223	171	1.216374	1.163743	1.304094
55	205	196	214	171	1.19883	1.146199	1.251462
60	203	192	210	171	1.187135	1.122807	1.22807
65	198	190	207	171	1.157895	1.111111	1.210526
70	197	190	204	171	1.152047	1.111111	1.192982
75	191	188	201	171	1.116959	1.099415	1.175439
80	188	187	200	171	1.099415	1.093567	1.169591
85	186	186	198	171	1.087719	1.087719	1.157895
90	185	184	194	171	1.081871	1.076023	1.134503
95	185	182	190	171	1.081871	1.064327	1.111111
100	185	181	190	171	1.081871	1.05848	1.111111
105	183	180	184	171	1.070175	1.052632	1.076023
110	182	179	183	171	1.064327	1.046784	1.070175
115	182	178	181	171	1.064327	1.040936	1.05848

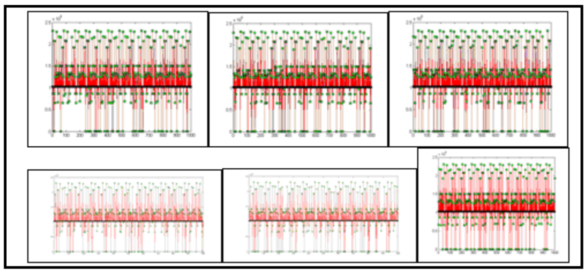
Figure 7 swim application memory reference pattern plot in detail

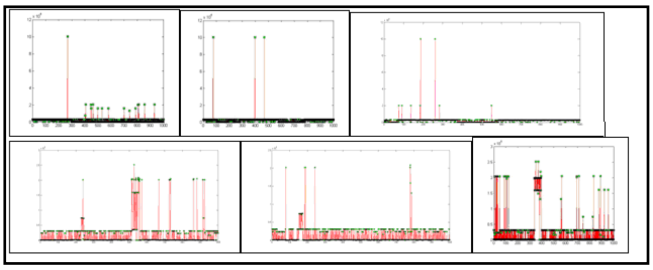
Figure8 gcc application memory reference pattern plot in detail

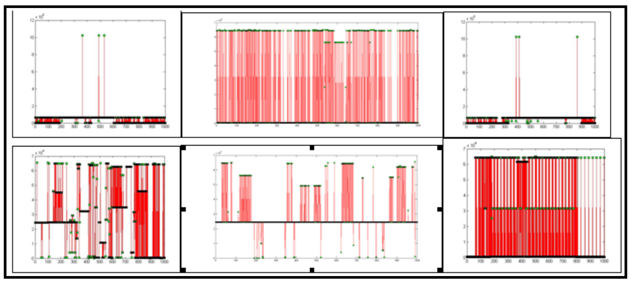
Figure 9 bzip application memory reference pattern plot in detail

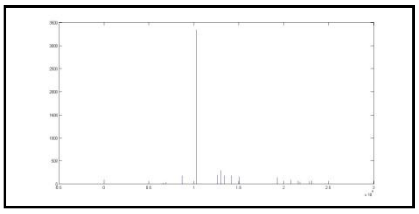
Figure 10 (swim app. Histogram)

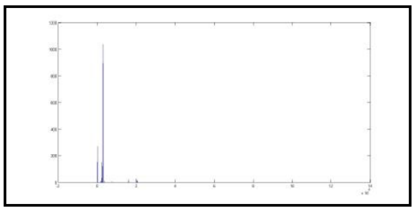
Figure 11 (gcc app. histogram)

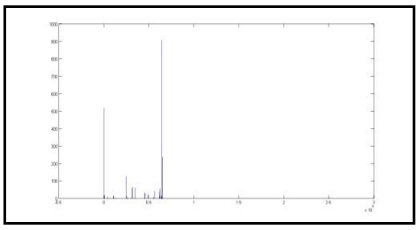
Figure 12 (bzip app. Histogram)

4.1 Histograms for various applications' page faults for different numbers of frames

4.1.1 swim application

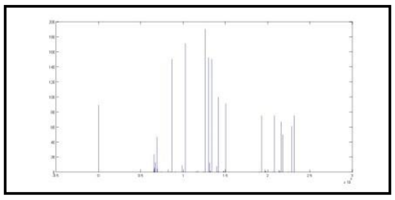
Figure 13 using lru, frames =5

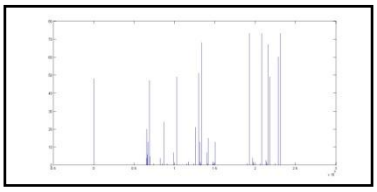
Figure 14 using lru, frames =15

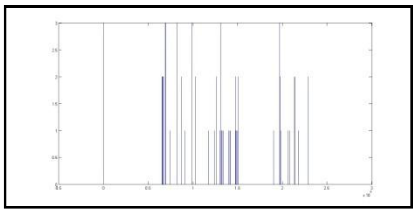
Figure 15 using lru, frames =50

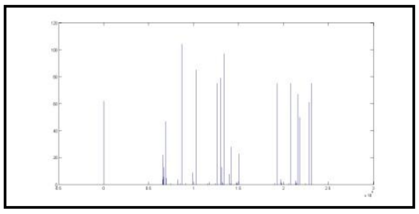
Figure 16 using optimal, frames =5

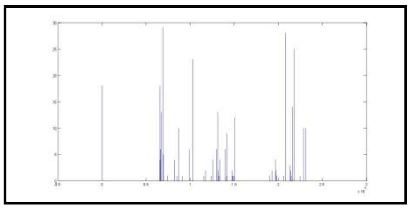
Figure 17 using optimal, frames =15

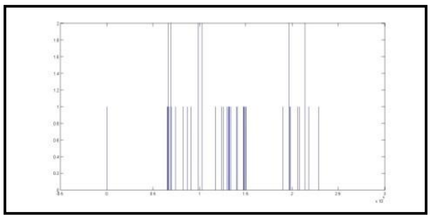
Figure 18 using optimal, frames =50

4.1.2 For gcc application

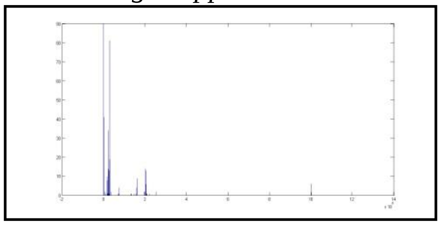
Figure 19 using lru, frames =5

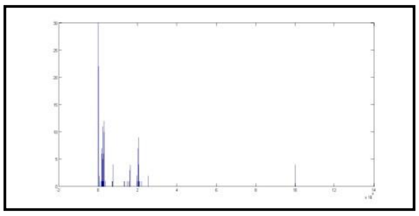
Figure 20 using lru, frames =15

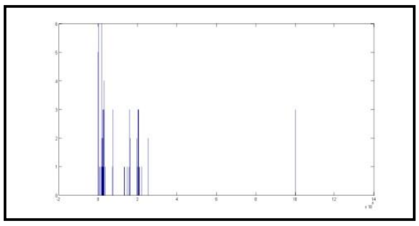
Figure 21 using lru, frames =50

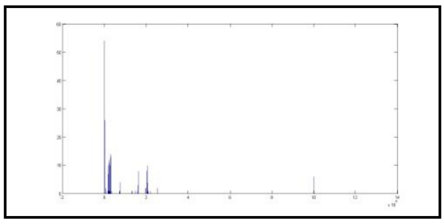
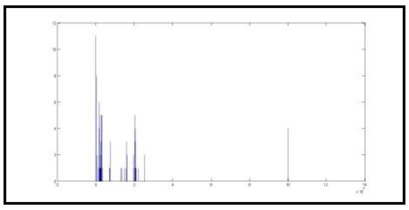
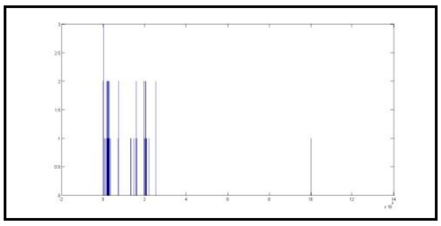

Figure 22 using optimal, frames =5 **Figure 23 using optimal, frames =15** **Figure 24 using optimal, frames = 50**

4.1.3 For bzip application

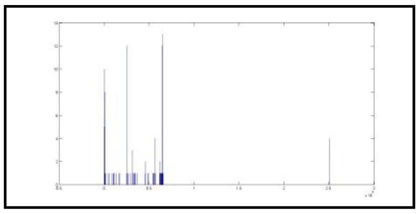
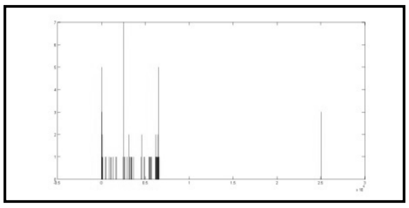
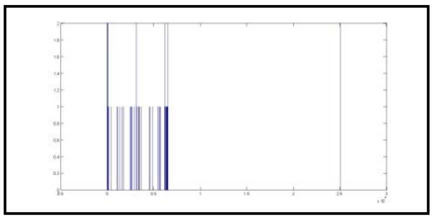

Figure 25 using lru, frames =5 **Figure 26 using lru, frames =15** **Figure 27 using lru, frames =50**

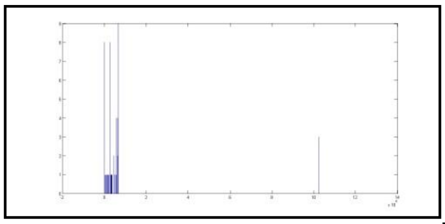
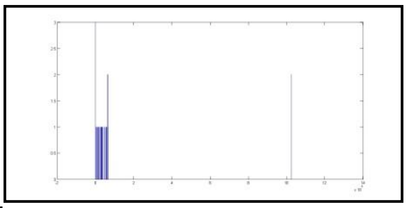
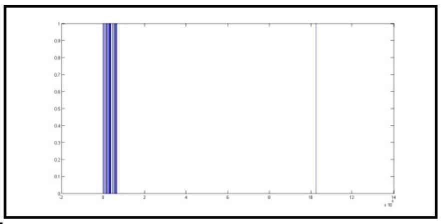

Figure 28 using optimal, frames =5 **Figure 29 using optimal, frames =15** **Figure 30 using optimal, frames =50**

5. OBSERVATIONS FROM SIMULATION RESULTS

Here we define a new term named as per page fault factor for the analysis. Per page fault factor for any policy defines the ratio of number of page faults for that policy and the number of page faults in optimal policy for the same application. If this factor is always decreasing for increasing number of frames then that policy behaves more closely with optimal policy. So in tables, factors FIFO/OPTIMAL, LRU/OPTIMAL, LRU2/OPTIMAL define per page fault factors for FIFO, LRU, LRU2 policies. Also for any policy, if this factor decreases more rapidly then that policy is better. So this factor gives deviation from optimal policy in terms of page faults.

Also from figur4, which shows the memory reference pattern plot for swim application, pattern appears to be quite stable. Also from table2 for gcc application & table3 for bzip application ,conclusions are same for FIFO and LRU2 described in the above paragraph except the fact that per page fault factor is always more then LRU for gcc and bzip application. From figure 8&9, it is clear that LRU2 performs better & close to optimal when pattern is more stable. Patterns in 8 &9 are not as stable as pattern for swim application in figure4.

Figure 10, 11, 12 shows histograms for the reference strings of 3 applications swim, gcc, and bzip applications respectively. From figure 10,11, 12 it is clear that few pages are used very heavily while few pages are used very rarely .It is clear from figure 10 page numbers near 100000 have very high frequency of occurrences .Also these page numbers are being used quite stably over a uniform pattern which is confirmed from figure 4, 7.

Also from figure 13, 14, 15, it is clear that as no of frames increase for LRU, page faults occur on same pages but with less frequency .Also as no of frames increases, LRU equalizes no of page faults for different pages.

From figure 25, 26, 27, it is clear that for 5 frames LRU generates largest no of page faults for pages of highest frequency of occurrence. Also for LRU, as no of frames increases, the pages on which higher no of page faults occur change. The same is true for optimal policy in figure 28, 29, 30.

In figure 13 , largest no of page faults are not generated for highest frequency of occurrence page numbers .Also from figure 13,14,15 it is concluded that as the no of frames increases , the pages on which higher no of page faults occur change .The same is true for optimal policy in figure 16,17,18.

From figure 19, 20, 21,22,23,24 it is clear that both LRU and Optimal generates largest no of page faults for pages not having highest frequency of occurrence. Also in both, as no of frames increases the pages on which largest no of page faults occur remain almost same.

From figure 16, 17, 18 (Histogram for page faults for optimal), It is clear that optimal also equalizes the no of page faults as no of frames increases but in a better way than LRU.

6. CONCLUSION

This paper studies lru and optimal with respect to frequency of page faults on different pages .It compares lru and optimal using histogram approach. Selected applications swim, gcc, bzip show different memory access pattern .Swim application show quite table pattern of memory access. Also different applications have different number of pages. Bzip application exhibits more correlated access than swim and gcc application.

It concludes that optimal and LRU both may and may not generate largest no of page faults on page numbers of highest frequency, depending upon memory reference access pattern.

Results show that as number of frames increases, both LRU, Optimal generates page faults on almost same page numbers with less frequency of page faults. For both LRU and Optimal the no, of page faults on pages of higher frequency change with no of frames.

It shows that page number of high frequency of occurrence contribute much to the total number of page faults for LRU policy. Also higher frequency page number's contribution to total number of page fault varies with the number of frame present in main memory. The same applies to optimal policy. It can be noted that pages of higher frequency of occurrence contribute a good amount to the total no. of number of page faults for both LRU and Optimal policies. Also in some cases, contribution of highly frequent pages to total no of page fault is less in optimal then LRU policy. It can be concluded that histograms for both LRU and Optimal policies equalize as the number of frames increases. Also histogram for optimal policy equalizes more rapidly then LRU policy's histograms.

This study can be quite useful for future designing of page replacement techniques.

7. ACKNOWLEDGEMENTS

I am thankful to my wife Mrs. Vandana gupta and my guide Prof. Navin rajpal for their support and cooperation.

8. REFERENCES

[1] Abraham Silberschatz, Peter Baer, 1999, Operating System Concepts (5th Ed.).New York: John Wiley & Sons, Inc.

[2] Peter J. Denning, Working Sets Past and Present, 1980 IEEE.

[3] Ben Juurlink, Approximating the Optimal Replacement Algorithm, *CF'04,* April 14–16, 2004, ACM 1581137419/ 04/0004.

[4] Sedigheh Khajoueinejad, Mojtaba Sabeghi, Azam Sadeghzadeh, A Fuzzy Cache Replacement Policy and its Experimental Performance Assessment, 2006 IEEE.

[5] Song Jianga,, Xiaodong Zhangb,, Token-ordered LRU: an effective page replacement policy and its implementation in Linux systems, 2004 Elsevier.

[6] Elizabeth J. O'Neil1, Patrick E. O'Neil1, Gerhard Weikum, the LRU-K Page Replacement Algorithm For Database Disk Buffering, SIGMOD Washington, DC, USA 1993 ACM.