THE UNIVERSITY of ADELAIDE

Group Report

Practical 2

COMP SCI 7064 Operating System

Group Name: Prac2 Group

Yili Wang a1714739

Changhao Li a1877290

Tianhao Luo a1896896

1. Introduction

The efficiency of memory management serves as a pillar in the performance of computer systems which is heavily determined by the efficacy of page replacement algorithms that manage virtual and physical memory. These algorithms manipulate the selection of pages and swap pages between virtual and physical memory when the memory is at capacity. In this project, our team aims to build a simulator to analyse and compare performance of four different page replacement algorithms: Clock, FIFO (First In First Out), LRU (Least Recently Used), and Random(RAND). Our object is to examine the efficiencies of these algorithms using different race datasets, which allows us to simulate real world application scenarios to differentiate each algorithm's performance under varying conditions.

2. Algorithm Analysis

2.1 Experiment Methodology

Our main object is to analyse and compare the performance of four page replacement algorithms which are clock, FIFO, LRU, and random by testing different trace datasets. Memory size will be the independent variable, which is defined by the number of memory page blocks available during the simulation. Page access hit rate and the number of disk writes will be the dependent variables. In the analysis, we will use temporal locality as one of the parameters which refers to the tendency of programs to access the same memory locations in a certain period of time. The page algorithms will be accessed based on their effectiveness of predicting the data needed. The parameter could assist in analysing the performance and efficiency of page replacement algorithms; the concepts can help understanding how well an algorithm can predict the data that will be used based on the recent access history of data. Algorithms that leverage both concepts can optimise the performance and maximise the efficiency.

2.2 Results and Analysis

2.2.1 bzip Trace

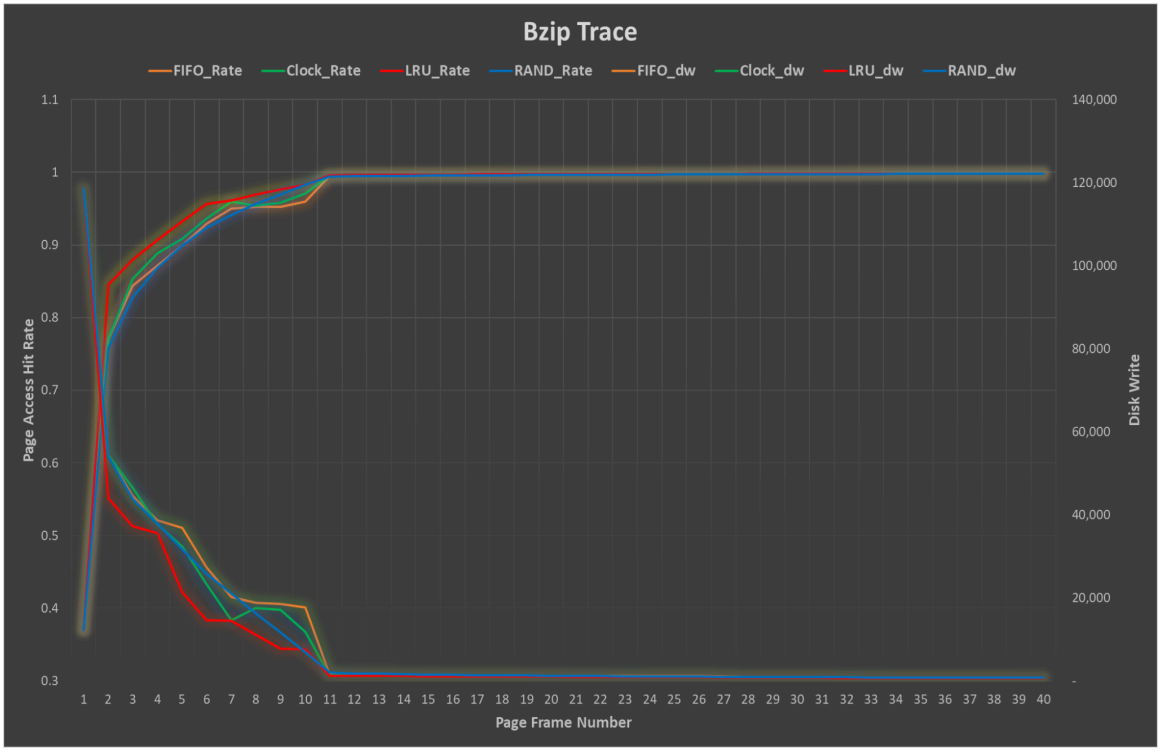

Figure 1. Hit Rate and Disk Write Number

The chart above shows the performance of clock, FIFO, LRU, and random page replacement algorithms running the Bzip trace over a period of 40 frames (total 600 frames was tested), the right vertical axis represents disk write and the left vertical axis represents page access hit rate, the horizontal axis is the page frame number which only the first 40 are showed on the graph after page access hit rate is nearly close to 100%.

All four algorithms have a similar hit rate at the first frame, which is normal since they all started with the same condition. After the second frame, all four algorithms had a substantial increase in hit rate. The LRU algorithm had a slightly higher hit rate during the first 10 frames which showed its ability to access more recently used pages. The Clock algorithms outperformed the FIFO algorithm since it can be considered as a modified form of the FIFO page replacement algorithm, it works just like FIFO but instead of paging out that page immediately, the algorithms checks to see if its referenced bit is set or check if it is 1 or 0. FIFO and RAND stayed close to each other for all frames. In the Bzip trace, LRU was the most efficient algorithm since it benefited from temporal locality; the pages that are recently accessed are more likely to be accessed again soon, which leads to higher hit rate. As the second most efficient algorithm, the clock algorithm also took advantage of temporal locality in which it kept track of frequently accessed pages and retained the pages that have been recently accessed in the memory.

The approximate memory sizes needed for each algorithm to reach 99.00% hit rate are shown below:

Algorithm	Page Frame Number	Memory Size
RAND	11	44 KB
FIFO	11	44 KB
CLOCK	11	44 KB
LRU	11	44 KB

Table 1. Memory Estimation

Overall, it took all four algorithms a short time to reach a hit rate of 99.9% in the Bzip Trace. The relationship of page access hit rate and disk write are negatively correlated since at a higher hit rate, the system is more frequently finding the needed data in the memory, which requires less operation of fetching data from the disk, or writing and updating new data.

2.2.2 gcc Trace

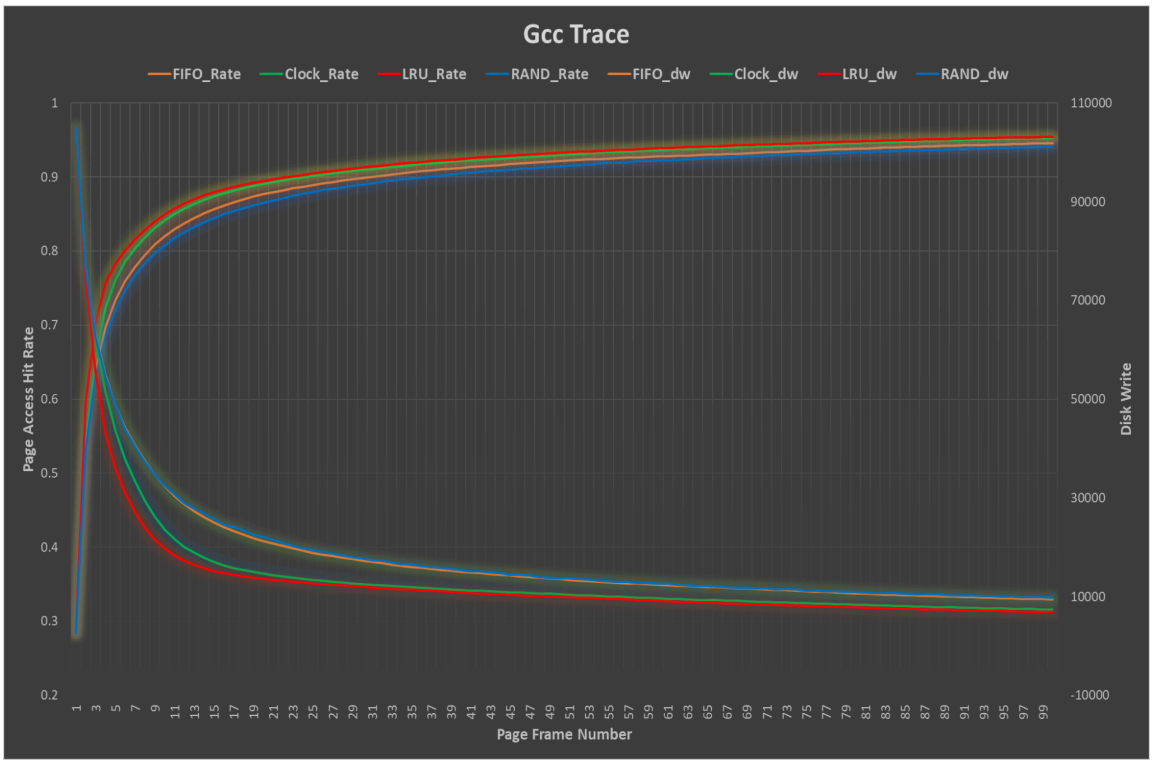

Figure 2. Hit Rate and Disk Write Number

The chart above shows the performance of clock, FIFO, LRU, and random page replacement algorithms running the Bzip trace over a period of 100 frames (total 1200 frames was tested), the right vertical axis represents disk write and the left vertical axis represents page access hit rate, the horizontal axis is the page frame number which only the first 100 are showed on the graph after page access hit rate is nearly close to 95%. The reason for not showing the rest of the 1100 frames is that the first 0-50 frames is where the most substantial changes happen regarding hit rate and disk write, it is easier to see the rate of change. The first 10 frames are

where the hit rates for all the algorithms rapidly increase. In the 10-20 frames, the slope of hit rate is flatter showing that algorithms are beginning to stabilise as more frequently accessed pages are learned by the algorithm and stored on the memory. The algorithms continue to improve the cache at a smooth increase rate, the hit rate reached 95% around 600th frames, and our goal is to reach a miss rate less than 1%, so we added another 600 frames, and the hit rate eventually reached 99%+.

In the gcc trace, the LRU algorithm which leverages temporal locality consistently has a higher hit rate across all frames by focusing on the most recently used items. Similar to the performance in the bzip trace, the clock algorithm is the second most efficient algorithm that showed its effective use of temporal locality.

For gcc trace, the approximate memory sizes needed for each algorithm to reach 99.00% hit rate are shown below:

Algorithm	Page Frame Number	Memory Size
RAND	775	3.03 MB
FIFO	747	2.92MB
CLOCK	568	2.22 MB
LRU	529	2.07MB

Table 2. Memory Estimation

2.2.3 sixpack Trace

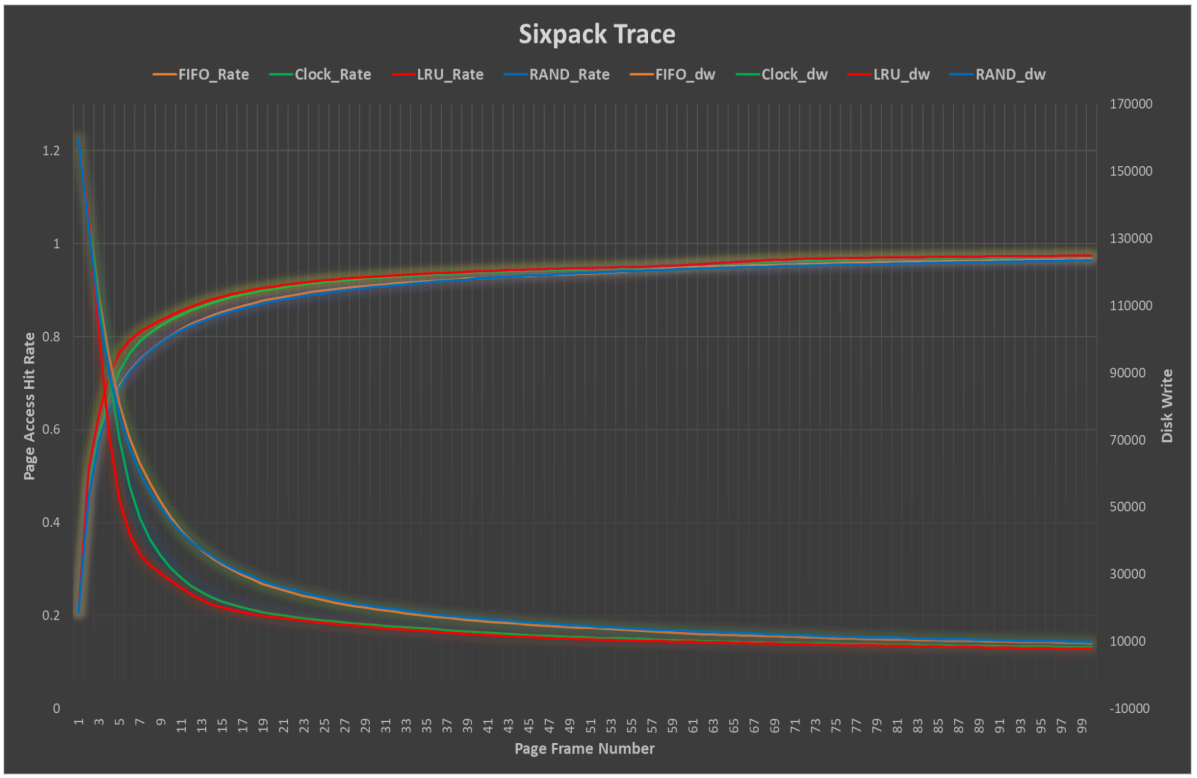

Figure 3. Hit Rate and Disk Write Number

In the chart, all four algorithms display similar hit rate curve as those of the gcc trace that Clock and LRU perform similarly and better than FIFO and RAND across all numbers of PFN. The disk write curves also have similar shapes, as the hit rate goes up, we certainly expect fewer page faults, and fewer dirty pages to be swapped out thus reducing the number of disk writes. However, the inflection points of LRU and Clock seem to occur at a lower hit rate compared with the gcc ones', an indication that the temporal locality is comparatively low in this simulation's workload when the page frame number is below 15, when the physical memory is low. As memory size increased, the hit rate of these two algorithms in gcc do go lower than that of sixpack eventually. Another interesting aspect is that FIFO and RAND share pretty much identical curve, which also suggest that the oldest page that gets removed in FIFO might get swapped back soon after so randomly swapping a page might achieve a better result. In order to achieve 99 plus percent hit rate, LRU needs 291 page frame number, which would translate to 1.16MB of memory, Clock at 304 PFN, translating to 1.18MB, FIFO at 400 PFN, needing 1.56 MB and RAND at 444, which would need 1.73MB. We can see that such a percentage of hit rate is achieved much earlier than the gcc case, so its workload actually has comparatively higher temporal locality, and we just need enough memory to store all the frequently visited pages.

2.2.4 swim Trace

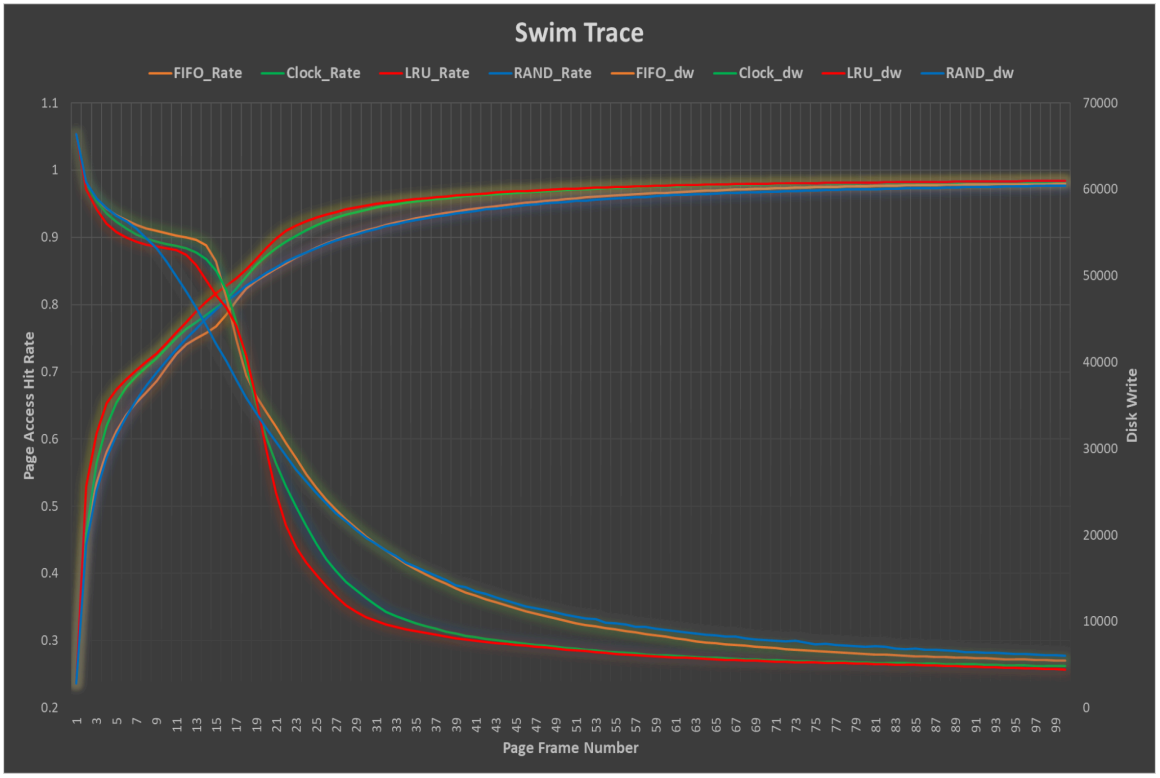

Figure 4. Hit Rate and Disk Write Number

It is very interesting that RAND actually performs relatively well when physical memory is low. Around PFN 15, it outperforms FIFO and even matches the performance of Clock. In the case of the low memory size in this simulation's workload, where randomly selecting a page

and getting lucky might achieve a better result than evicting the least used page. We can also observe that a plateau seems to occur from page frame number 3 to 17 for LRU, Clock, and FIFO, indicating that adding more memory does not reduce that much disk write, that pages are still consistently swapped out of the memory even though the memory size is increased. As the page frame number continues to increase, Clock and LRU still outperform FIFO and RAND. The 99 percent hit rate is achieved by LRU at 163 PFN, Clock at 176 page frame number, FIFO at 216, and RAND at 24. Since it is a write heavy workload and disk write drastically increases execution time, I think it would be beneficial if we try to increase memory size to prevent dirty pages from being replaced or swapped. Increasing memory size to 1.36MB at PFN around 350 for LRU and Clock would be better since it reduces disk write to half from around 3000 to around 1500 for both algorithms, as well as increases hit rate to 99.5 percent and to 2.14MB for the other two algorithms for similar effect.

3. Conclusion

The algorithms such as LRU and Clock that exploit the temporal locality of any workloads generally perform better than those that do not, across all the test cases and most ranges of PFN. However, we also need to know that in certain cases such as when the workload has low temporal locality and memory size is limited, IRU and Clock might not be able to outperform FIFO and RAND by much while spending more time searching pages to evict.

References

Arpaci-Dusseau, RH 2018, *OPERATING SYSTEMS : three easy pieces.*, Createspace, S.L.