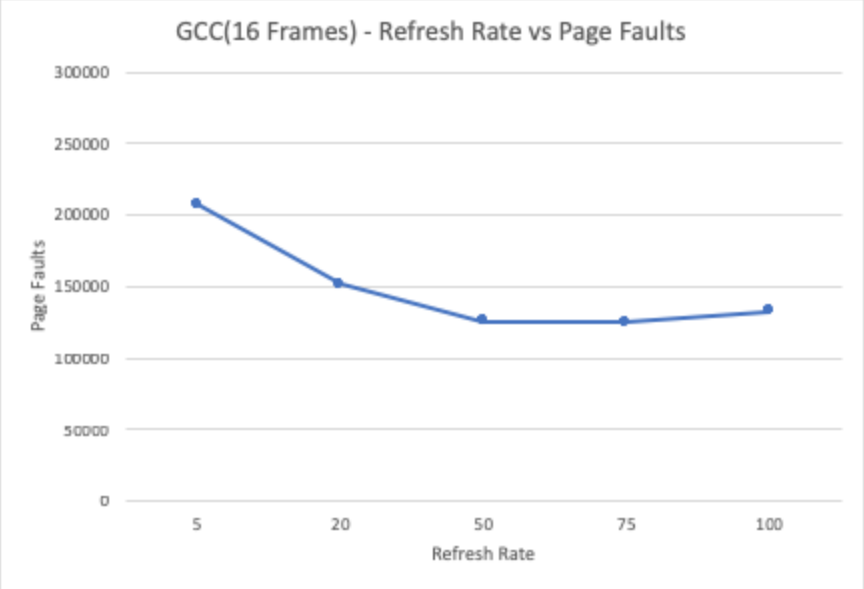
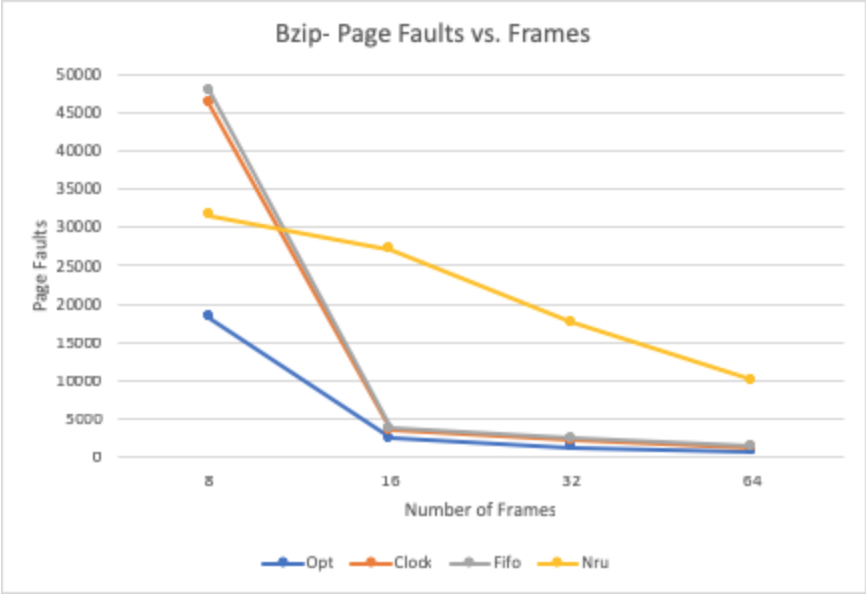
Joseph Reidell
CS 1550
Project 3 Write Up

NRU Analysis

Figure 1: Deciding optimal refresh rates based on page faults for NRU

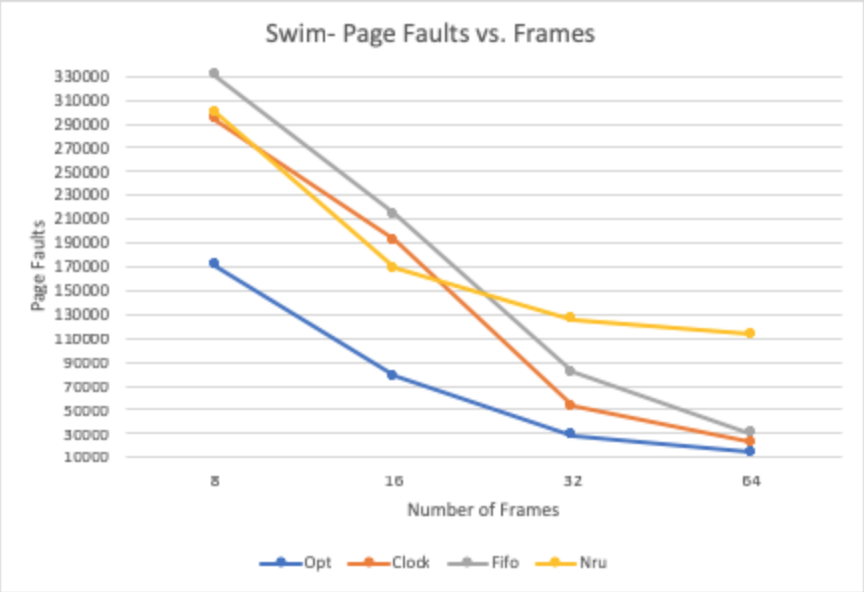


For the NRU(Not Recently Used) algorithm, higher and lower refresh rates can cause more or less page faults. I based my research on which refresh rate was better on five different rates. I started with 5, which ended up being the highest refresh rate. The middle of the road ended up being at 50. At 75 and 100, it starts to raise back up a bit. As we know, NRU resets the referenced bit of all frames in RAM after N instructions. To accurately compare the optimal NRU implementation to other algorithms, N must be set to the value that will provide the best results. The refresh rate of 50 ends up producing the most optimal results. It doesn't deliver the least amount of page faults but, it does to being the best compromise between other rates. Because of that, I used a refresh rate of 50 to be utilized for the NRU algorithm.



Bzip.trace

OPT	**Clock**
8: Page Faults: 18251	**8: Page Faults: 46164**
Writes to Disk: 7580	**Writes to Disk: 17568**
16: Page Faults: 2427	**16: Page Faults: 3468**
Writes to Disk: 847	**Writes to Disk: 1128**
32: Page Faults: 1330	**32: Page Faults: 2206**
Writes to Disk: 460	**Writes to Disk: 733**
64: Page Faults: 821	**64: Page Faults: 1314**
Writes to Disk: 283	**Writes to Disk: 441**
Fifo	**NRU**
8: Page Faults: 47828	**8: Page Faults: 31503**
Writes to Disk: 18797	**Writes to Disk: 7547**
16: Page Faults: 3820	**16: Page Faults: 27091**
Writes to Disk: 1335	**Writes to Disk: 6349**
32: Page Faults: 2500	**32: Page Faults: 17636**
Writes to Disk: 850	**Writes to Disk: 2227**
64: Page Faults: 1470	**64: Page Faults: 10019**
Writes to Disk: 513	**Writes to Disk: 1663**



Swim.trace

OPT

8: Page Faults: 171244

Writes to Disk: 46451

16: Page Faults: 78312

Writes to Disk: 18132

32: Page Faults: 28826

Writes to Disk: 6910

64: Page Faults: 14289

Writes to Disk: 4100

Clock

8: Page Faults: 293519

Writes to Disk: 54327

16: Page Faults: 191848

Writes to Disk: 48350

32: Page Faults: 53025

Writes to Disk: 11140

64: Page Faults: 22611

Writes to Disk: 5844

Fifo

8: Page Faults: 330893

Writes to Disk: 55488

16: Page Faults: 214295

Writes to Disk: 47434

32: Page Faults: 81638

Writes to Disk: 18172

64: Page Faults: 30422

Writes to Disk: 7585

NRU

8: Page Faults: 299994

Writes to Disk: 51432

16: Page Faults: 168633

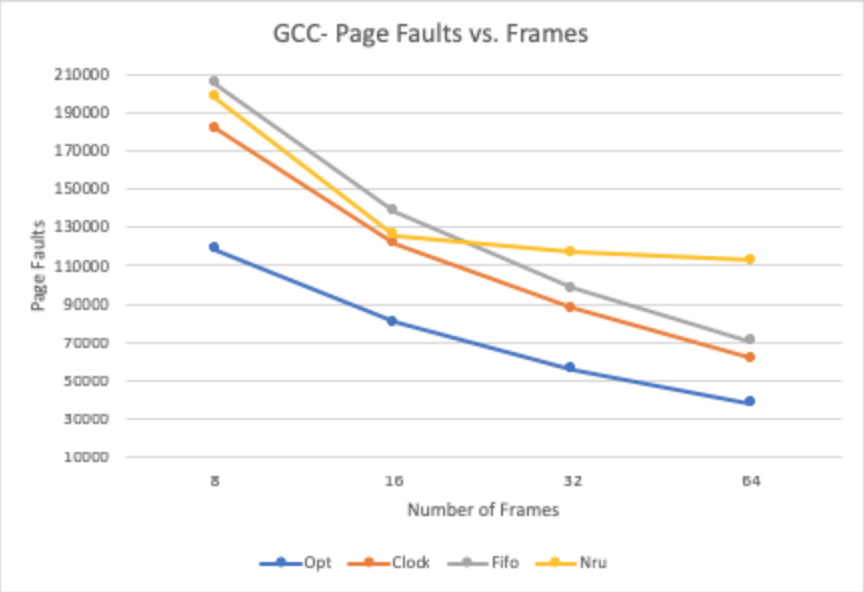
Writes to Disk: 29544

32: Page Faults: 125545

Writes to Disk: 13220

64: Page Faults: 113665

Writes to Disk: 12390



gcc.trace

OPT	**Clock**
8: Page Faults: 118480	**8: Page Faults: 181856**
Writes to Disk: 15031	**Writes to Disk: 29401**
16: Page Faults: 80307	**16: Page Faults: 121682**
Writes to Disk: 11316	**Writes to Disk: 16376**
32: Page Faults: 55802	**32: Page Faults: 87686**
Writes to Disk: 8274	**Writes to Disk: 12293**
64: Page Faults: 38050	**64: Page Faults: 61640**
Writes to Disk: 5730	**Writes to Disk: 9346**
Fifo	**NRU**
8: Page Faults: 205368	**8: Page Faults: 197881**
Writes to Disk: 37524	**Writes to Disk: 18938**
16: Page Faults: 138539	**16: Page Faults: 125610**
Writes to Disk: 24043	**Writes to Disk: 12441**
32: Page Faults: 98067	**32: Page Faults: 117062**
Writes to Disk: 16759	**Writes to Disk: 10122**
64: Page Faults: 70315	**64: Page Faults: 113027**
Writes to Disk: 12053	**Writes to Disk: 9676**

Based off of four algorithms (opt, clock, fifo, nru), and the comparisons I generated between them, it was quite clear based off of the amount of page faults and disk writes. This was based off the performance from three trace files (bzip.trace, swim.trace, gcc.trace). They all had different performances based on the files size of each. It wouldn't be bad to say that the OPT algorithm is the best way to go about it but, it is rather impossible right now to come up with an absolute perfect way to implement the algorithm. NRU could be very useful but, with refresh rate being a factor, it could become a little bit more complicated to use on a standard operating system. Fifo isn't a bad option either but, from what I have discovered that will be talked about later, it does have a slight possibly for page faults to increase a bit with more frames being used. It can make Fifo not that reliable to use for the standard operating system. In my opinion, I found clock to be the better algorithm for a standard operating system. The clock algorithm gives priority to pages that are constantly referenced between page evictions, where as NRU may evict a page that was utilized in the previous ten memory accesses because its referenced bit was a reset a couple of iterations ago. Clock algorithm tends to be the most efficient.

Based on the graphs above, opt is obviously the lowest but, again, it is nearly impossible to get a good implementation to make it good for standard operating systems. Fifo starts off really high with lower number of frames but, tends to even out over time with higher frames. Based off the graphs, fifo seems to have the worst performance out of the four algorithms. Comparing clock and fifo, for the file bzip.trace, they tend to work at the same rate. Comparing that to swim.trace and gcc.trace, it just isn't the same. Fifo has way more page faults than clock has on any of these files. NRU at eight frames starts off high but, starts to level out being even after sixteen frames and so on. NRU does tend to come up with the worst amount of page faults

but, clock at the beginning does the same. Clock however, significantly gets lower page faults the higher the frames tend to get. After 16 frames, it is difficult to say NRU can be any better than clock can be. It's also hard to compare clock to opt seeing as opt definitely works better but, it isn't easy to implement it. In the end, after all the comparisons, the clock algorithm possesses the necessary characteristics to be practically utilized in an operating system with either sixteen, thirty-two, or sixty-four frames to squeeze out its optimal performance compared to NRU and Fifo.

Talking about fifo, I found one instance that would cause Belady's anomaly. It happens in the bzip.trace file. With frames 91, it has a number of page faults at 1009. However, with frames 92, it has a number of page faults at 1103. That's a significant spike compared to how low the number of page faults was going. With 92 frames, the spike goes back down to 1077. After that it never does hit a page fault number of 1009 again. The lowest it goes once it hits 100 frames is 1029 page faults. That is still significant compared to how low it had already gotten too. That was the only instance of Belady's anomaly that I encountered during testing.