James Hahn

CS1550 Project 3

Analyzing Page Faults and Disk Writes of Page Replacement Algorithms

After simulating four page replacement algorithms (Optimal, NRU, Clock, and Random), generating comparisons between them is clear when graphing number of page faults and disk writes as the number of frames in physical memory are increased. One thing to note is two files were used (bzip.trace and gcc.trace), so the performance of the algorithms may differ slightly depending on a given file. If a memory address trace never requires writes, the random algorithm may be the most practical and efficient algorithm because it decides the page to evict in constant time. However, if many memory accesses are frequently performed on a few pages and a significant number of pages are dirty, out of the three practical algorithms (Optimal cannot be implemented because it requires knowledge of the future), the Clock algorithm will be the most efficient because it gives priority to pages that are constantly referenced between page evictions, whereas NRU may evict a page that was utilized in the previous ten memory accesses because its referenced bit was reset a couple of iterations ago.

Before summarizing the performance of each algorithm in this report, the refresh rate of NRU must be discussed. Because 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. To make this decision, Figures 1 and 2 were generated. It turns out that N=50 produces the most optimal results. Although it is does not deliver the least number of page faults (N=75) and disk writes (N=20), it is very slightly behind both of those values, which leads to it generating the best compromise between the two. Thus, for the resets, a refresh rate of fifty memory accesses will be utilized for the NRU algorithm.

From further analysis and simulations, as seen from the figures below, Optimal is by far the best algorithm. Regardless of the file, it outcompetes every other algorithm in both disk writes and page faults regardless of the number of frames in physical memory. Despite this, Optimal cannot be regarded as a practically implementable algorithm because it needs information about how soon each page will be accessed in the future. As a result, with a slight edge, Clock proves to be the best implementable algorithm. First, look at the gcc.trace file (Figures 3 and 4). As the number of frames increases from eight to sixty-four, it outperforms both the Random and NRU algorithms dramatically in the number of page faults incurred. As for disk writes, despite barely losing to NRU for sixteen and thirty-two frames, it makes up for that by either matching or besting it in eight and sixty-four frames. Note, the Random algorithm performs terribly with the number of required disk writes. Next, the bzip.trace file further cements the Clock algorithm's superiority compared to NRU (Figures 5 and 6). When examining the number of page faults, NRU produces a considerably worse number of page faults compared to the other three algorithms across all number of frames; Clock and Random are basically tied in their results, so no real comparisons can be made between the two in this file. As for disk writes, NRU's case does not look as bad as it initially outperforms the Clock and Random algorithms by about 8,000 disk writes; this is not a huge difference because bzip.trace performs one million memory accesses. However, its benefits stop there; for sixteen and more frames, Random and Clock heavily outperform NRU in terms of disk writes, essentially nullifying its competitive edge.

As a result, after analyzing two different trace files and four algorithms, the superior page replacement algorithm to utilize is Clock. Not only is it practically implementable, unlike Optimal, but it outperforms NRU and Random in most cases in terms of disk writes and page

faults. In addition, to rule out Random even further, its algorithm is not stable. Since it is random, it may perform significantly worse than what was shown in the figures despite taking the average page faults and disk writes of five simulations. To conclude, the Clock algorithm possesses the necessary characteristics to be practically utilized in an operation system with either sixteen, thirty-two, or sixty-four frames to squeeze out its optimal performance compared to the Random and NRU algorithms.

Appendix

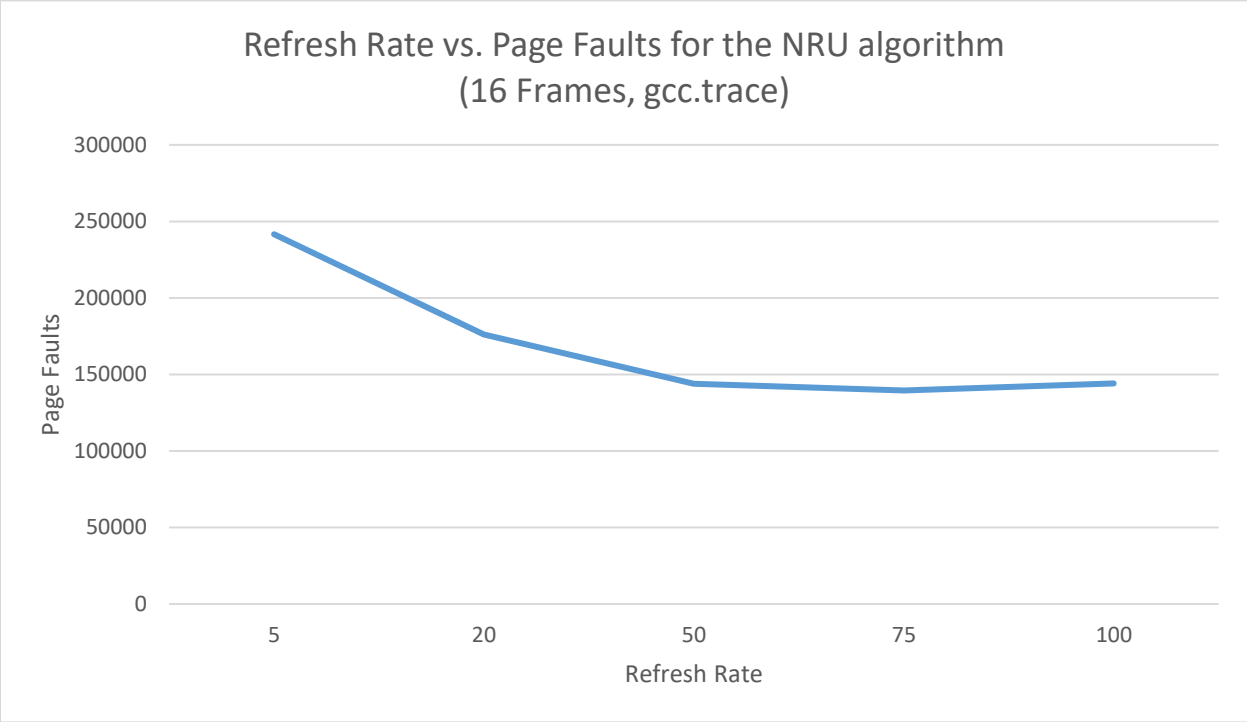

Figure 1. *Deciding the optimal refresh rate based on Page Faults for NRU*

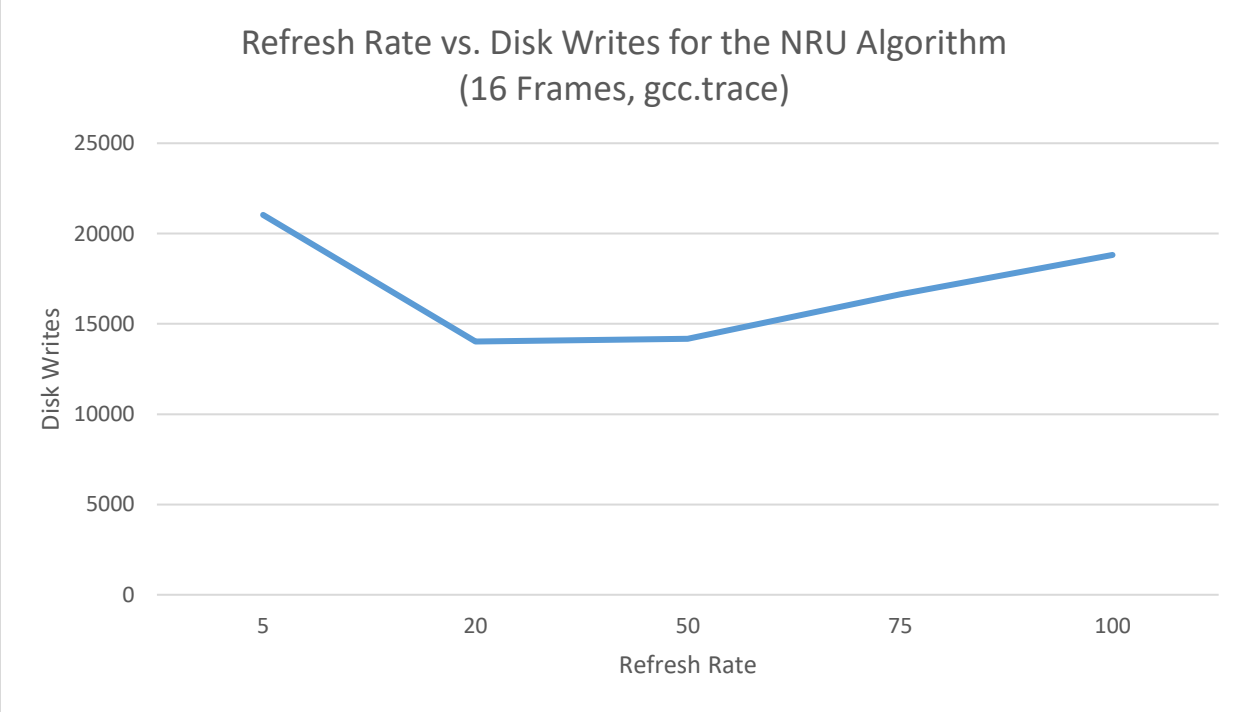

Figure 2. *Deciding the optimal refresh rate based on Disk Writes for NRU*

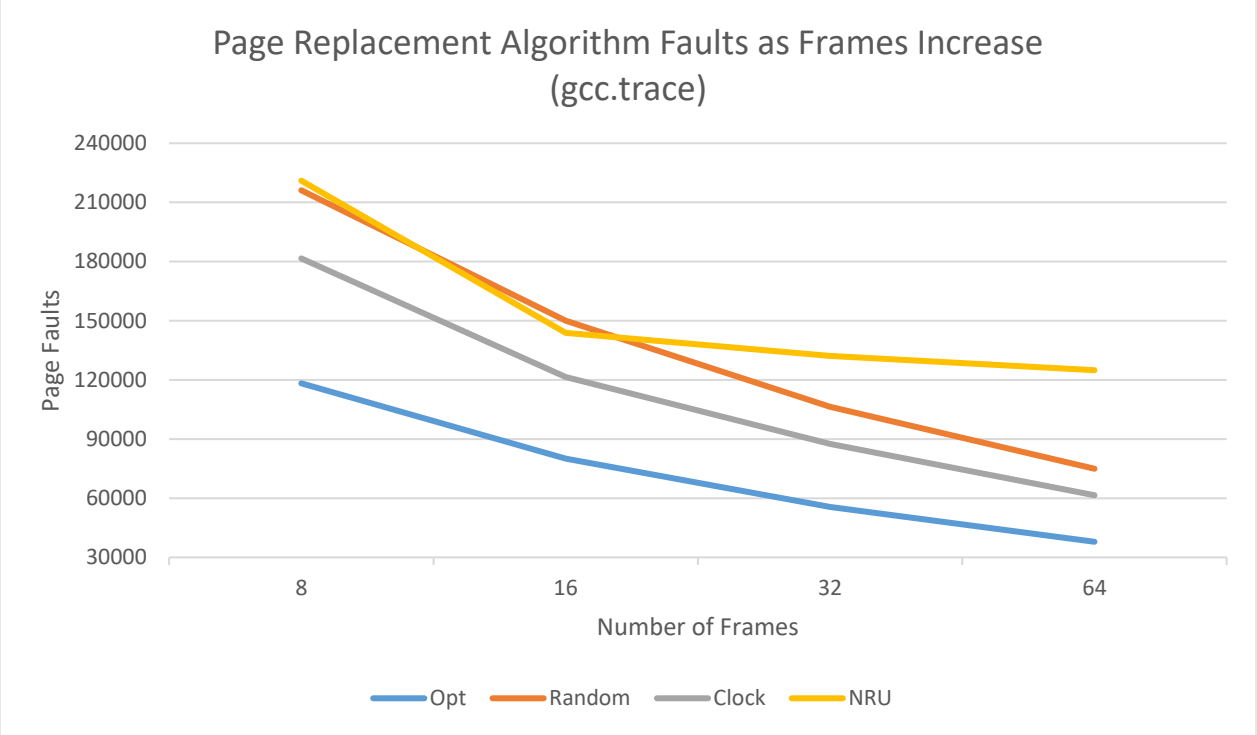

Figure 3. *Number of faults per algorithm with 8, 16, 32, and 64 frames for file gcc.trace*

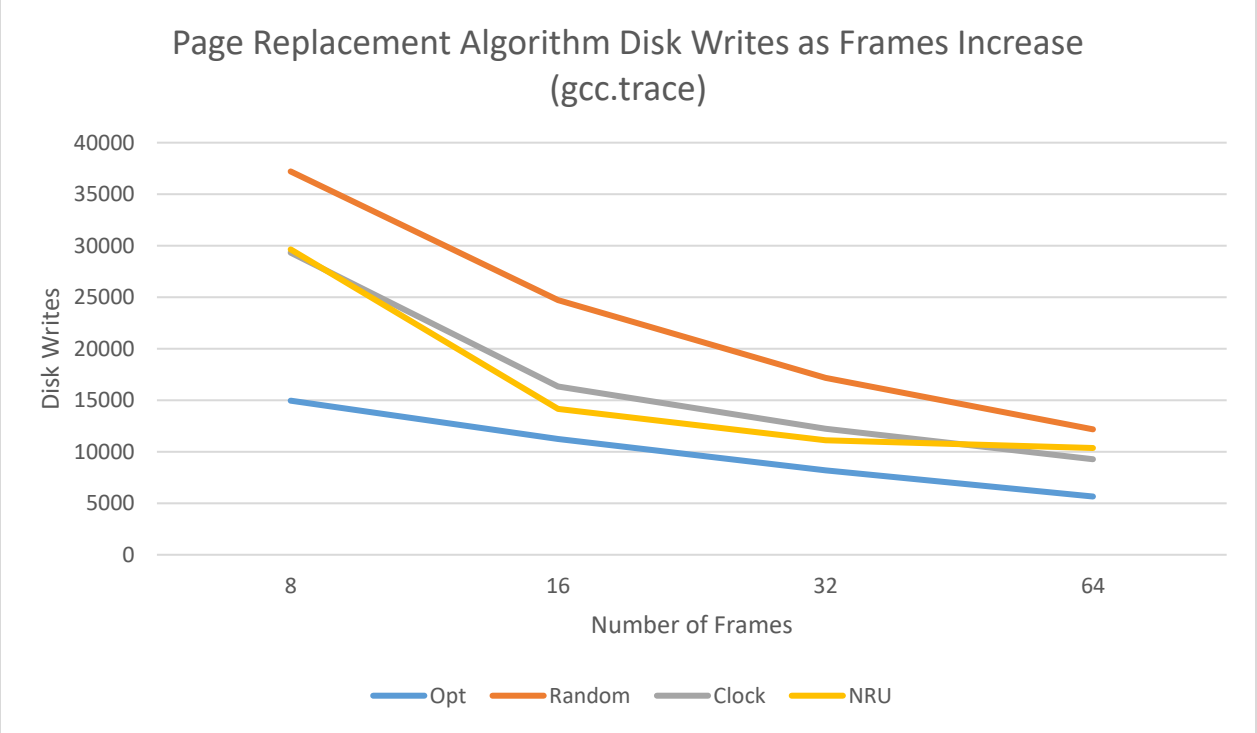

Figure 4. *Number of disk writes per algorithm with 8, 16, 32, and 64 frames for file gcc.trace*

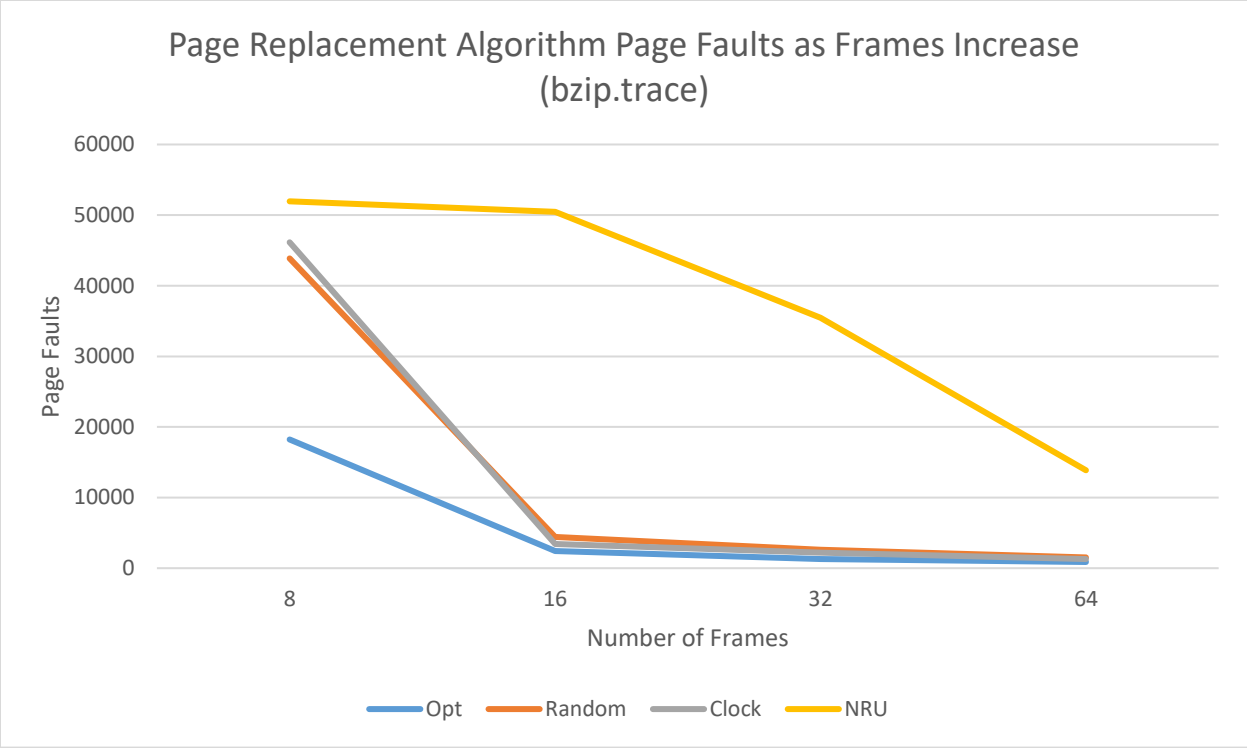

Figure 5. *Number of faults per algorithm with 8, 16, 32, and 64 frames for file bzip.trace*

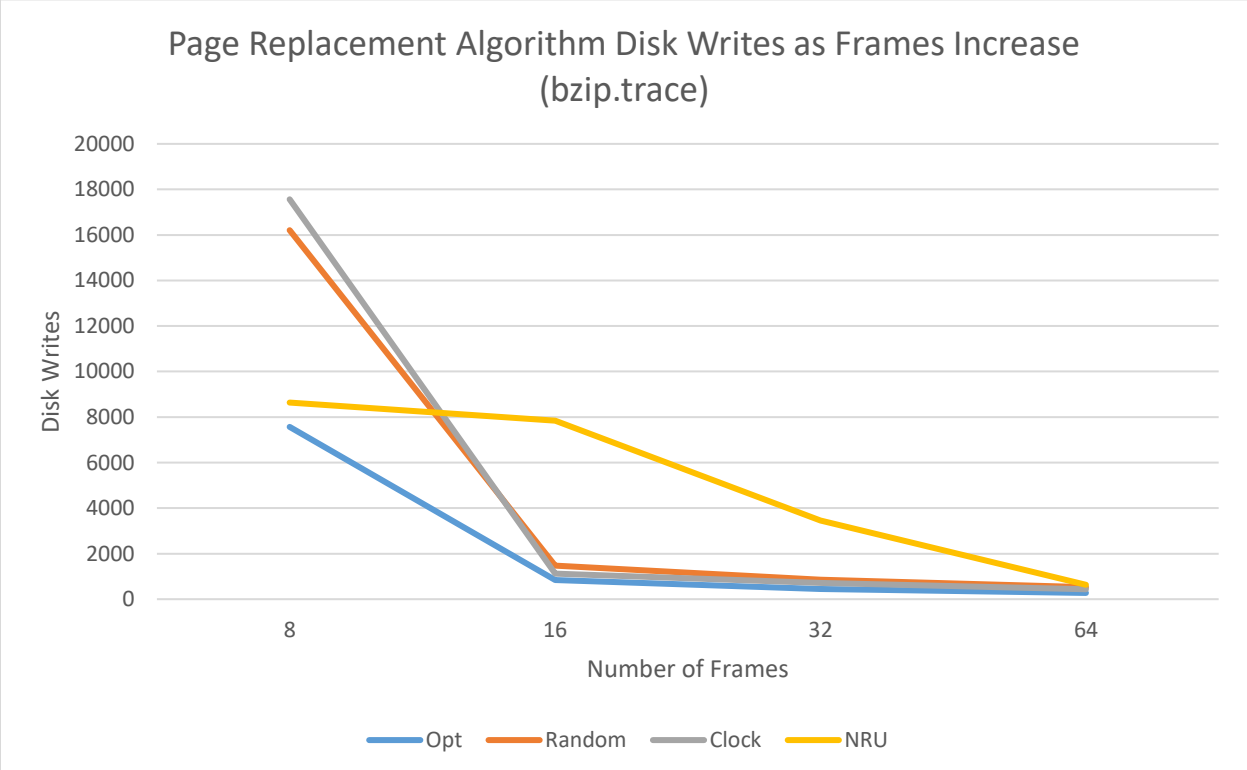

Figure 6. *Number of disk writes per algorithm with 8, 16, 32, and 64 frames for file bzip.trace*