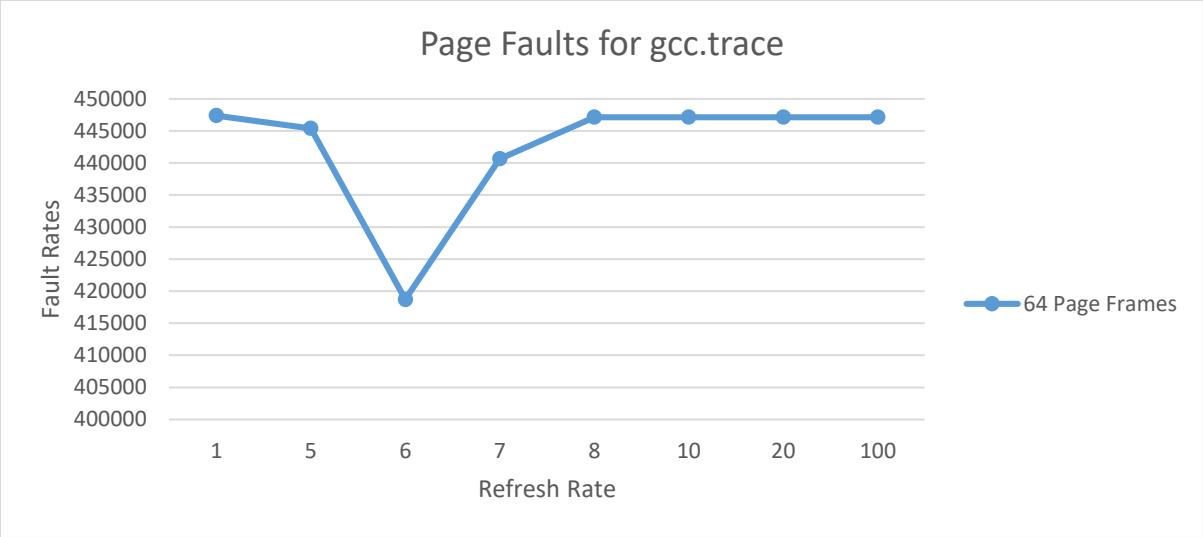
Jake Halloran

11/13/16

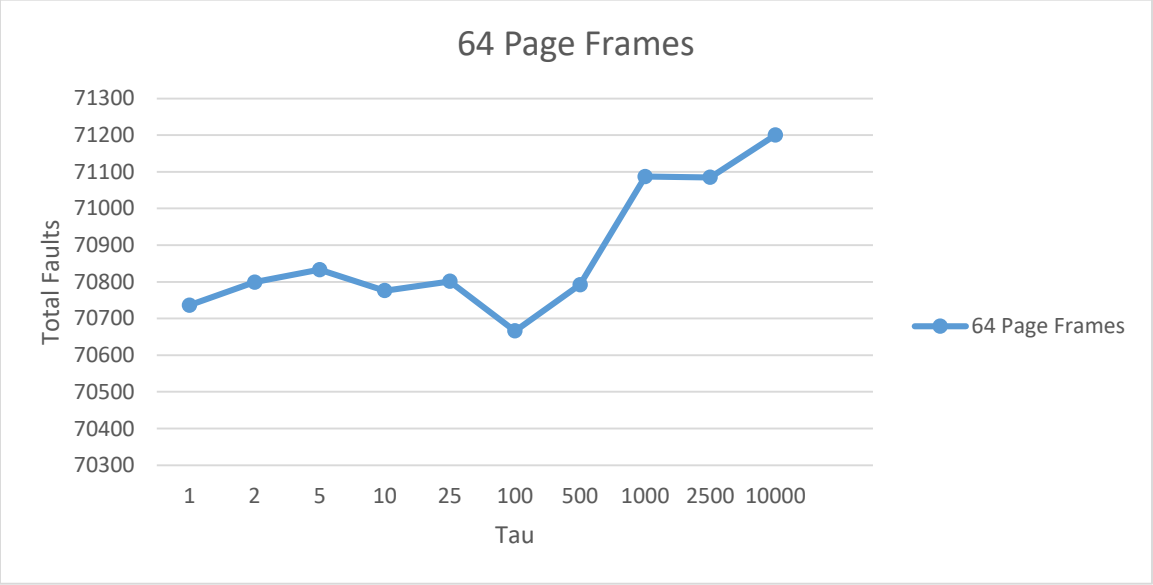
CS 1550 Project 3

Aging Algorithm.) All tests in chart done on gcc.trace.



My testing on gcc.trace seems to show that the optimal refresh rate for the aging algorithm is 6 cycles. A low number of cycles makes sense as it needs to be low or else some frames would be evicted before any refresh would be processed making the whole refresh process pointless. Additionally, the refresh rate cannot be too low as then not enough data is entered for a difference to be made by refreshing (for example refreshing every cycle would just be wasting time).

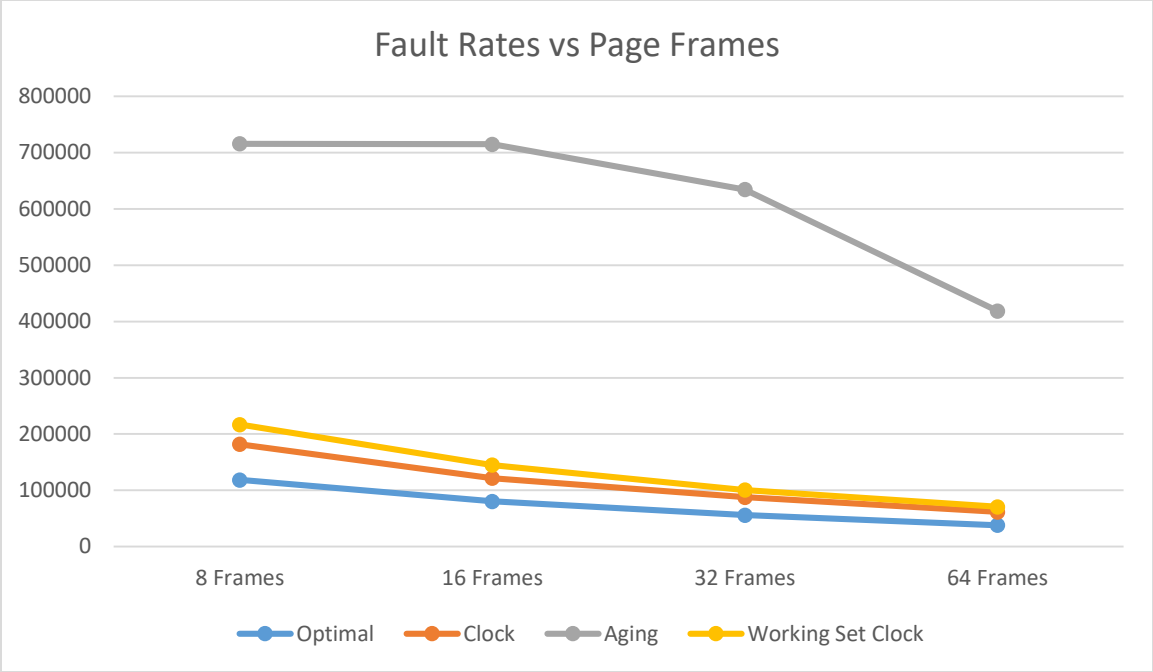
Working Set Clock Algorithm) All tests done on gcc.trace



For choosing a tau I tried many values before settling in for around 100 as the best option as it both is late enough to allow some values to be older than tau while at the same time not being so late that everything is older than tau (thus killing the point). Additionally, I looked at tau values based at least partly on the number of disk writes on which a tau of around 100 worked as it was a median tradeoff between lower disk writes and faults. After determining tau, I went back to fiddle with the refresh rate to experiment on whether or not the optimal refresh rate was different and found that while, for low frame numbers the optimal rate had lowered, for high frame numbers, the optimal refresh was about the same and so I simply left it as is.

Results) All tests in chart run on gcc.trace

Algorithm	Page Frames	Page Faults	Memory Writes
Optimal	8	118480	15031
Optimal	16	80307	11316
Optimal	32	55802	8274
Optimal	64	38050	5730
Clock	8	181856	29401
Clock	16	121682	16376
Clock	32	87686	12293
Clock	64	61640	9346
Aging	8	715727	104895
Aging	16	715007	104794
Aging	32	634404	95235
Aging	64	418711	37175
Working Set Clock	8	216956	39589
Working Set Clock	16	144819	25506
Working Set Clock	32	100579	17535
Working Set Clock	64	70666	12762



My results seem to imply that of the 3 algorithms other than optimal studied that clock is the best to use in a real life system, at least as far as can be determined by running against the memory requests found in these traces. Clock has the results most similar to optimal for both raw number of faults and memory writes required for functionality, which means that by the two most important metrics it wins. As far as the other options studied here, working set clock also puts up respectable performance especially for the low frame number runs, however the clock algorithm runs quicker in real time and is slightly better by the studied metrics giving it the edge once again. Finally, the last studied option was the aging algorithm which simply did not work as well as clock or working set and definitively not as well as optimal. For some cases the aging algorithm had nearly 7 times the faults of optimal which is simply unacceptable for real world performance as far as conclusions can be drawn from this limited data. Additionally, although the data is not displayed here, the conclusions made above are also supported by running the program against bzip.trace, however, for bzip.trace working set clock and clock were much closer together to the point of being nearly interchangeable. In fact, for 8 page frames,

clock was worse than working set clock due to the increased trend of repetitive data. This increase for repetitive data further means that although I would still choose clock due to its lower fault rate on random data, the working set clock implementation is certainly a valid choice as programs often use data repeatedly. Even for bzip.trace however, aging was the slowest of the algorithms. In conclusion, I would use clock for a real life operating system due to its performance on random data more accurately reflecting optimal fault rates. Additionally, I would choose clock because working set was more competitive with optimal fault rates on runs with low numbers of memory frames which does not at all reflect the modern situation where even my laptop has 16GB of RAM.