CS 1550 Project 3 Write Up

Dr. Mosse - Spring 2019
Mathew Varughese
mav120@pitt.edu

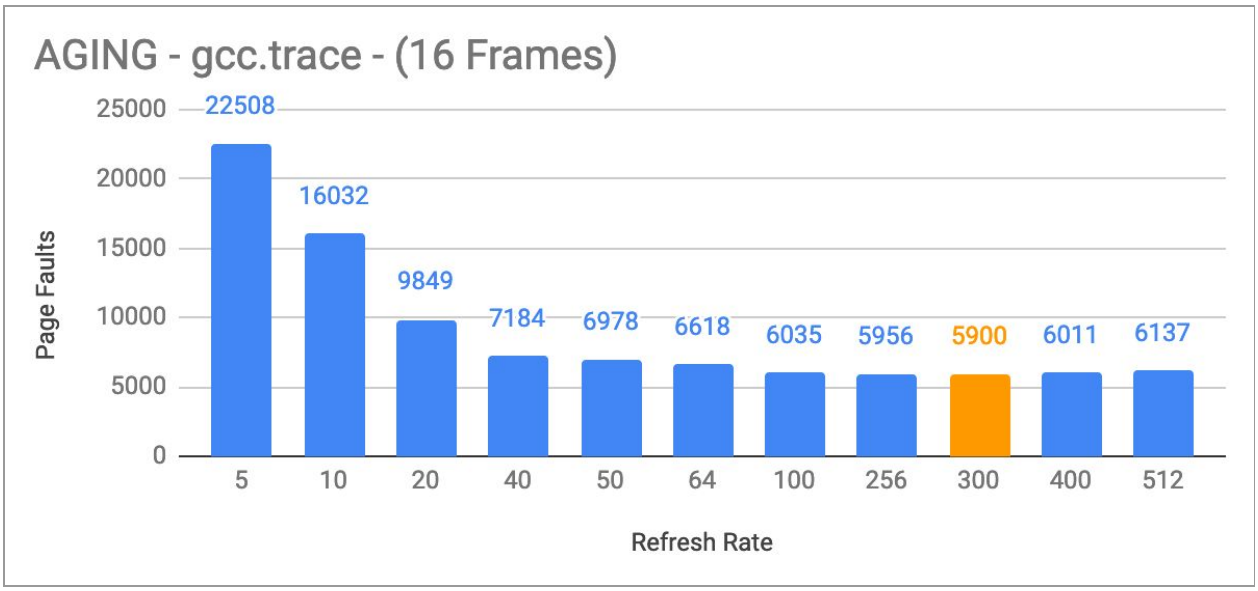
Aging Algorithm

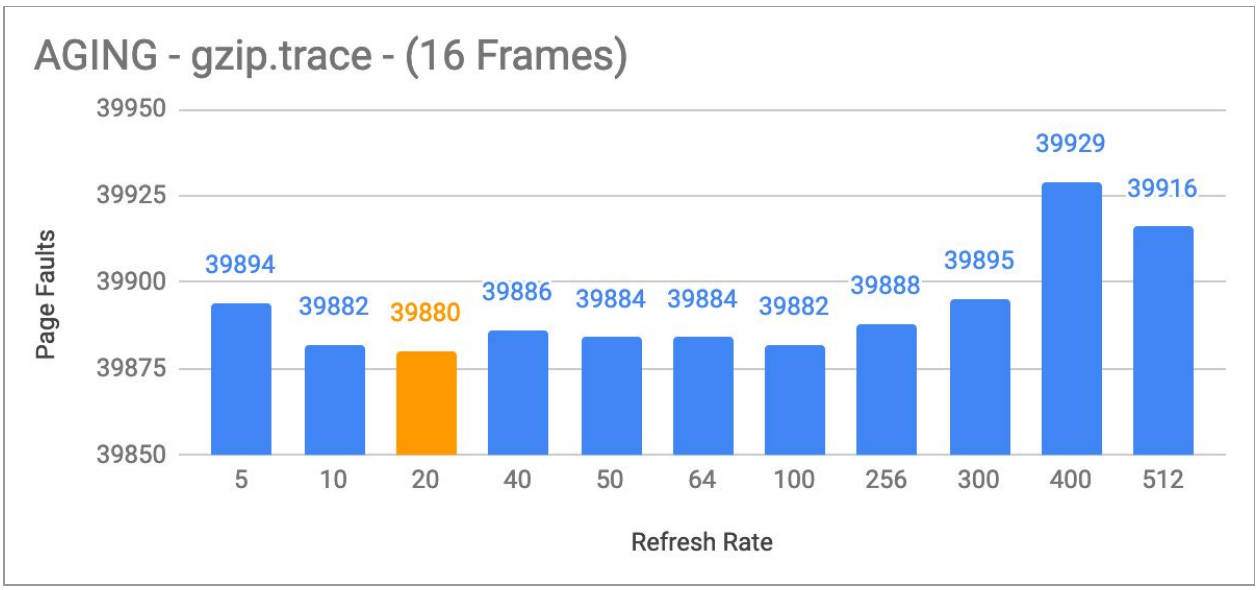
The aging algorithm takes in a “refresh” parameter. This refresh parameter is used to determine how many clock cycles should happen between each refresh. When a refresh occurs, all of the pages in the frame table have their aging counter shifted right by one. Effectively, this refresh ages the pages that are not often used. However, determining an appropriate refresh parameter is tricky. It turns out to be quite dependent on the specific file and the number of frames used. To determine a parameter, I first used 16 frames and tested the algorithm on different refresh rates.

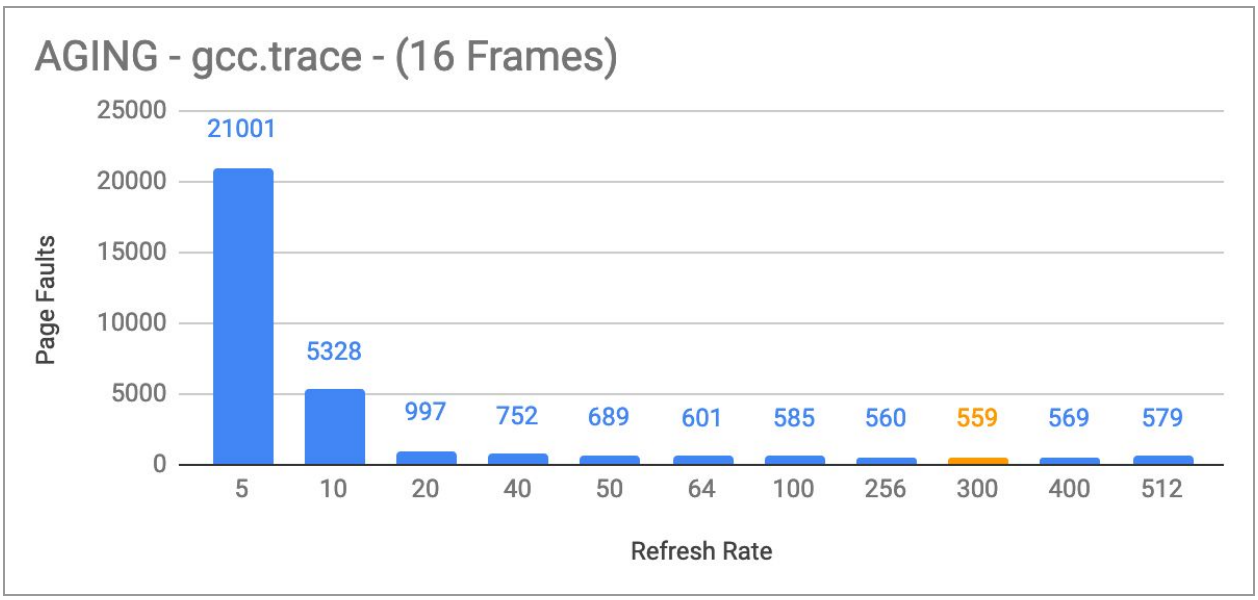
This data is shown in this table below:

gcc.trace		gzip.trace		swim.trace	
Refresh Rate	**Page Faults**	**Refresh Rate**	**Page Faults**	**Refresh Rate**	**Page Faults**
5	22508	5	39894	5	21001
10	16032	10	39882	10	5328
20	9849	20	39880	20	997
40	7184	40	39886	40	752
50	6978	50	39884	50	689
64	6618	64	39884	64	601
100	6035	100	39882	100	585
256	5956	256	39888	256	560
300	5900	300	39895	300	559
400	6011	400	39929	400	569
512	6137	512	39916	512	579

Graphing this data helped give a little insight into which parameter works the best. These can be seen on the next page. However, it does not give a clear picture which refresh parameter to choose. For `gcc` and `swim`, a parameter of 300 seemed to work. For `gzip`, the refresh rate parameter is 20. This is because it seems dependent on the number of frames used. On the page after this there is a chart that compares refresh rate and different frames.


AGING - gcc.trace - (16 Frames)
Page Faults
25000
20000
15000
10000
5000
0
22508
16032
9849
7184
6978
6618
6035
5956
5900
6011
6137
5
10
20
40
50
64
100
256
300
400
512
Refresh Rate


AGING - gzip.trace - (16 Frames)
Page Faults
39950
39925
39900
39875
39850
39894
39882
39880
39886
39884
39884
39882
39888
39895
39929
39916
5
10
20
40
50
64
100
256
300
400
512
Refresh Rate


AGING - gcc.trace - (16 Frames)
Page Faults
25000
20000
15000
10000
5000
0
21001
5328
997
752
689
601
585
560
559
569
579
5
10
20
40
50
64
100
256
300
400
512
Refresh Rate

Refresh Rate vs Page Faults per Frame				
gcc.trace	**Page Faults**			
Refresh Rate	**8 Frames**	**16 Frames**	**32 Frames**	**64 Frames**
16	22400	13471	12510	6474
32	22413	7454	5588	3266
64	22771	6618	3194	1910
128	23436	5976	1732	1213
256	24499	5956	1048	795
400	25268	6011	837	607
512	25801	6137	783	566
650	27101	6173	761	553
gzip.trace	**Page Faults**			
Refresh Rate	**8 Frames**	**16 Frames**	**32 Frames**	**64 Frames**
16	39903	39879	39875	39875
32	39902	39882	39874	39874
64	39916	39884	39874	39874
128	39998	39883	39874	39872
256	40089	39888	39874	39874
400	40049	39929	39875	39874
512	40203	39916	39874	39874
650	40462	40141	39890	39874
swim.trace	**Page Faults**			
Refresh Rate	**8 Frames**	**16 Frames**	**32 Frames**	**64 Frames**
16	9297	2115	1286	361
32	8127	825	618	319
64	6595	601	444	301
128	6377	575	376	290
256	5885	560	277	232
400	6468	569	231	198
512	6684	579	213	185
650	6861	585	208	174

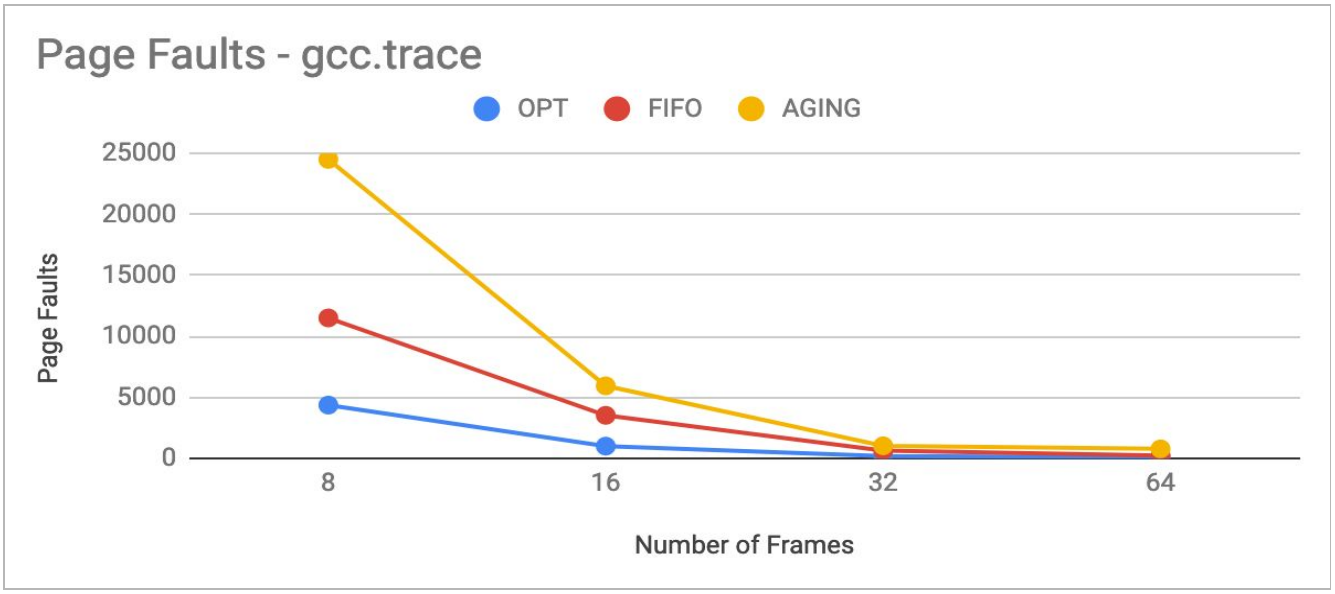
The graphs for this data in shown on the next page. From here, it seems that a refresh rate of about 256 seems to be sufficient for the purposes of this project. Although intuitively it seems like a small refresh rate would be better, more time allows more pages that will be accessed to turn their reference bit on. It would be better to have more traces to test to be more certain of an answer.

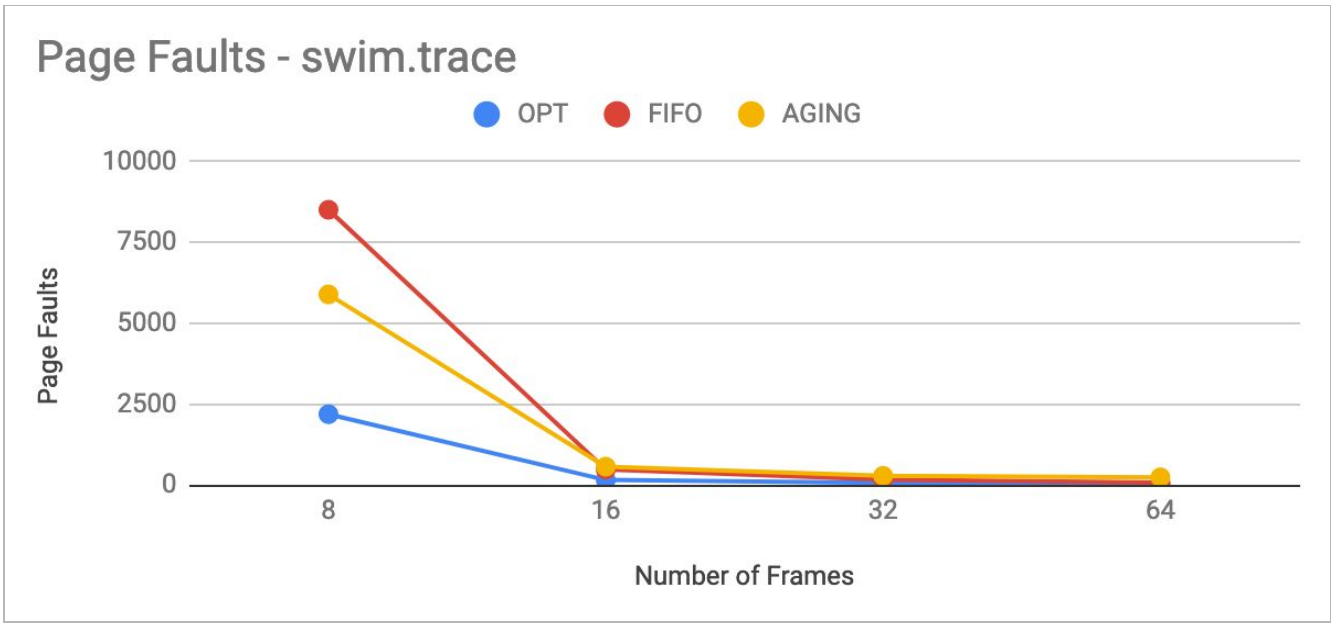
Comparing Page Replacement Algorithms

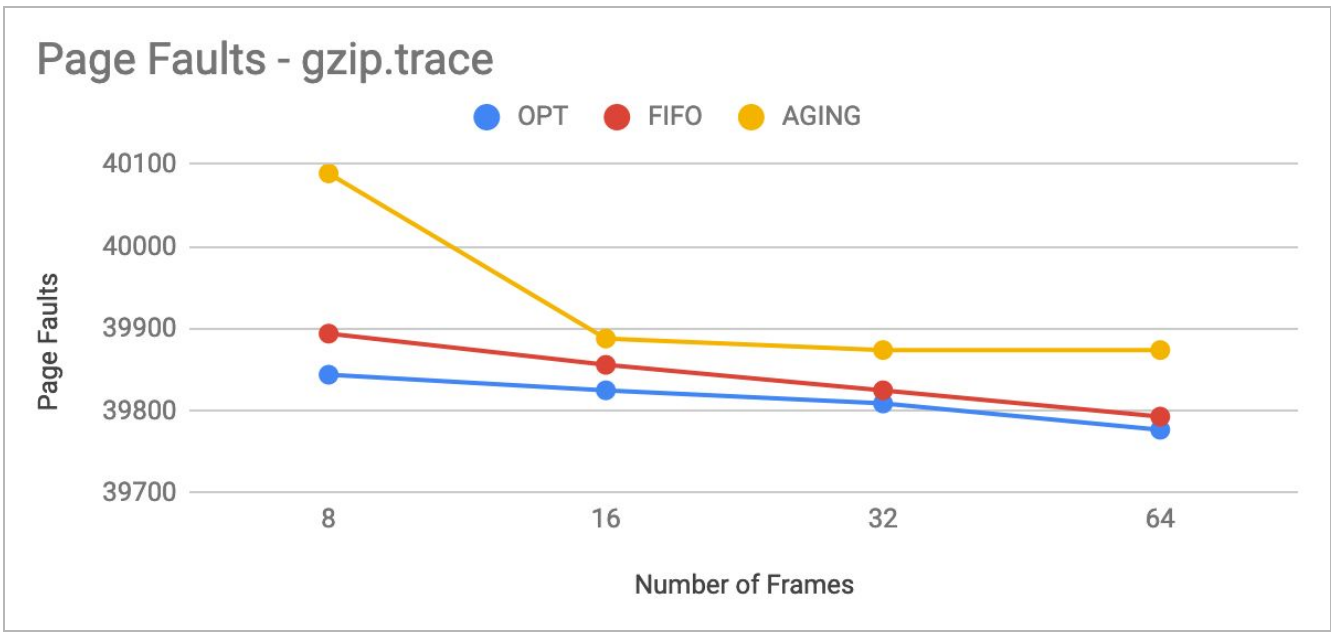
The table below shows the page faults and disk writes of performing the simulation on different frames and using different algorithms.

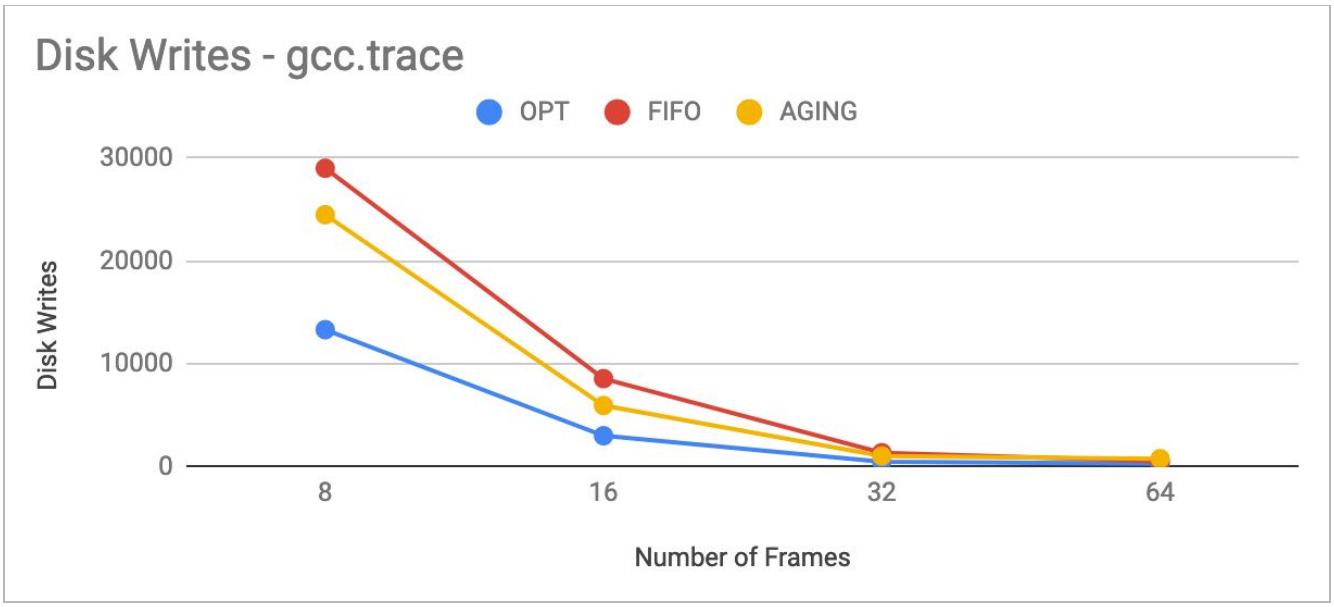
gcc.trace	**Page Faults**			**Disk Writes**		
Frames	**OPT**	**FIFO**	**AGING**	**OPT**	**FIFO**	**AGING**
8	4380	11519	24499	13328	29011	24499
16	1019	3542	5956	3020	8568	5956
32	215	660	1048	491	1375	1048
64	105	258	795	318	551	795
swim.trace	**Page Faults**			**Disk Writes**		
Frames	**OPT**	**FIFO**	**AGING**	**OPT**	**FIFO**	**AGING**
8	2182	8499	5885	4417	13893	5885
16	149	470	560	358	844	560
32	60	158	277	144	326	277
64	26	69	232	135	177	232
gzip.trace	**Page Faults**			**Disk Writes**		
Frames	**OPT**	**FIFO**	**AGING**	**OPT**	**FIFO**	**AGING**
8	39844	39894	40089	39874	44918	40089
16	39825	39856	39888	39856	42384	39888
32	39809	39825	39874	39856	41120	39874
64	39777	39793	39874	39856	40496	39874

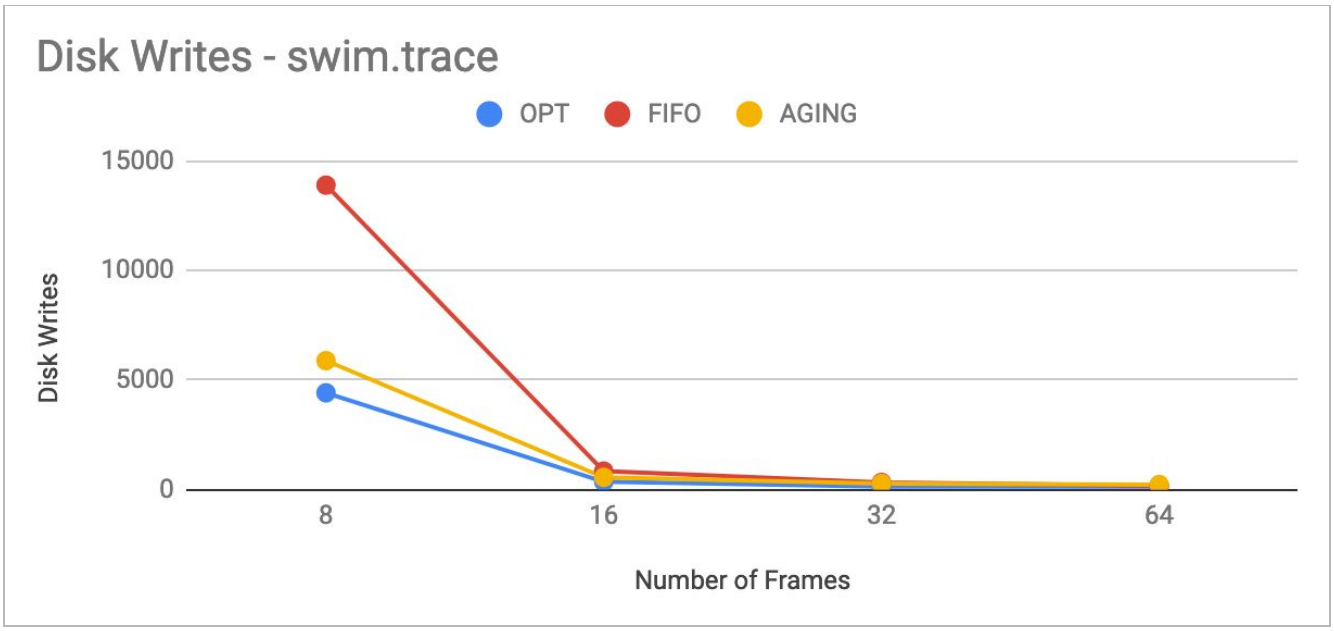
Charts on the next pages display a better picture of this data.

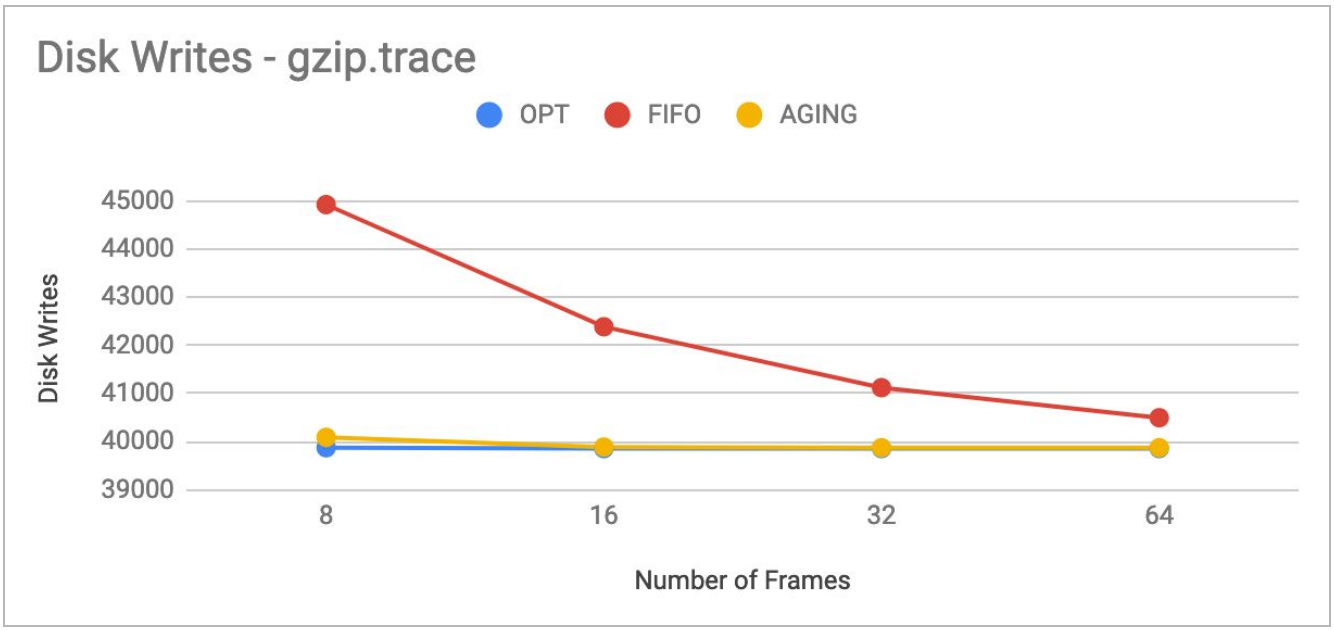

Page Faults - gcc.trace
OPT
FIFO
AGING
25000
20000
15000
10000
5000
0
Page Faults
8
16
32
64
Number of Frames


Page Faults - swim.trace
OPT
FIFO
AGING
10000
7500
5000
2500
0
Page Faults
8
16
32
64
Number of Frames


Page Faults - gzip.trace
OPT
FIFO
AGING
40100
40000
39900
39800
39700
Page Faults
8
16
32
64
Number of Frames


Disk Writes - gcc.trace
OPT
FIFO
AGING
30000
20000
10000
0
Disk Writes
8
16
32
64
Number of Frames


Disk Writes - swim.trace
OPT
FIFO
AGING
15000
10000
5000
0
Disk Writes
8
16
32
64
Number of Frames


Disk Writes - gzip.trace
OPT
FIFO
AGING
45000
44000
43000
42000
41000
40000
39000
Disk Writes
8
16
32
64
Number of Frames

The data shows that increasing the number of page frames decreases the number of page faults and disk writes. This is as expected. Other than cases of Belady's Anomaly, which will be explained in the next section, increasing the number of frames will increase the number of pages stored. This means page faults occur less because there is a much higher probability a page that needs to be found can be in the page table.

These charts also depict the idea of the law of diminishing returns. The law of diminishing returns is an economic concept that shows that not every increase in input will give a linear increase in output. At a point, after adding more input, there is decreasing benefit to the rate of return. After analyzing the number of frames, is it evident that increasing them from 8 to 16 gives a substantially lower number of page faults. The difference between 32 and 64 is not so large however. This is because as the number of frames grows to infinity, the number of page faults decreases. However, this might be unnecessary. Not all programs need to use an infinite number of frames.

This is where the importance of picking an appropriate page replacement algorithm comes into play. A smart page replacement algorithm is able to decide which frames to evict with higher confidence. This in turn lowers the number of page faults. The optimal page replacement algorithm can only occur in a perfect world where the Operating System somehow knows exactly which memory calls will be made when. It chooses which page to evict based on which one will be used the farthest into the future. In the graphs, it is evident the optimal page replacement algorithm performs the best. It can be mathematically shown that it is better than the other two.

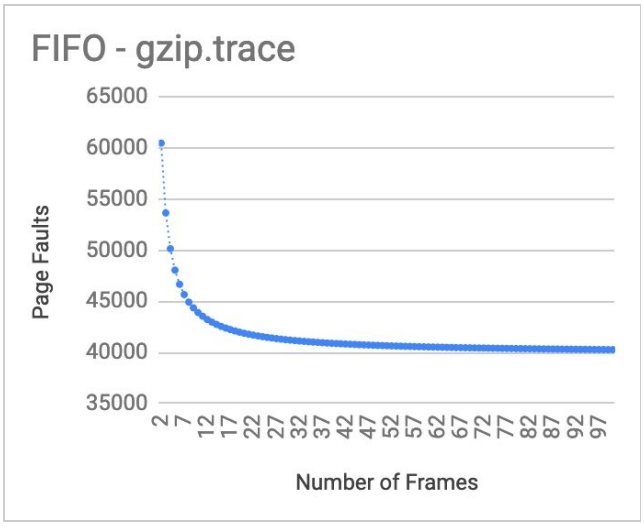
First-In-First-Out is a simple algorithm that does not perform that great. It decides to evict the first page that was inserted into the page table. In the chart, it surprisingly performed better than the aging algorithm in some cases. For example, in `gzip`, it was able to do significantly less page faults than the aging algorithm. Again, this is program dependent. Since `gzip` is a compression program, it likely does not use the same memory locations often. However, this is speculation. Based on the other programs, FIFO performs more disk operations and usually more page faults. Writing pages back to disk occur when a page is dirty. This is extremely slow and should be avoided at all costs.

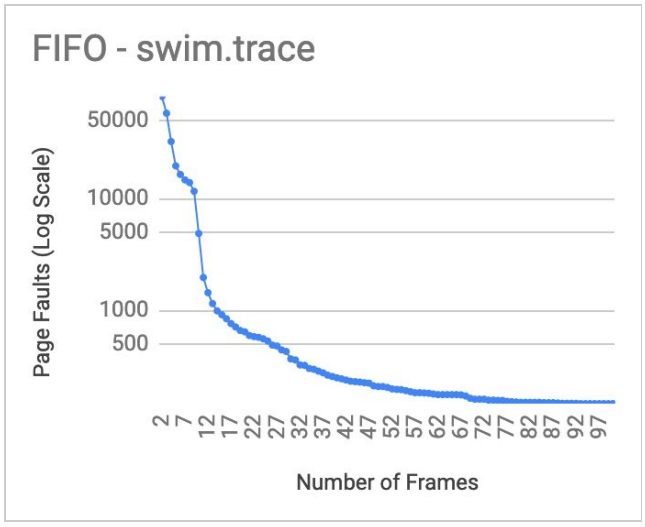
For this reason, the *aging algorithm* is likely the best page replacement algorithm to use in an actual Operating System. Of course, this is dependent based on the data and programs given. The aging algorithm could adjust the refresh rate based on machine learning or other algorithms to determine the best refresh rate for each process. 256 was used as a rough estimate as optimal here, but it is hard to tell for sure. Machine learning would be extremely useful here in using past data to predict future behavior in order to pick the most efficient values.

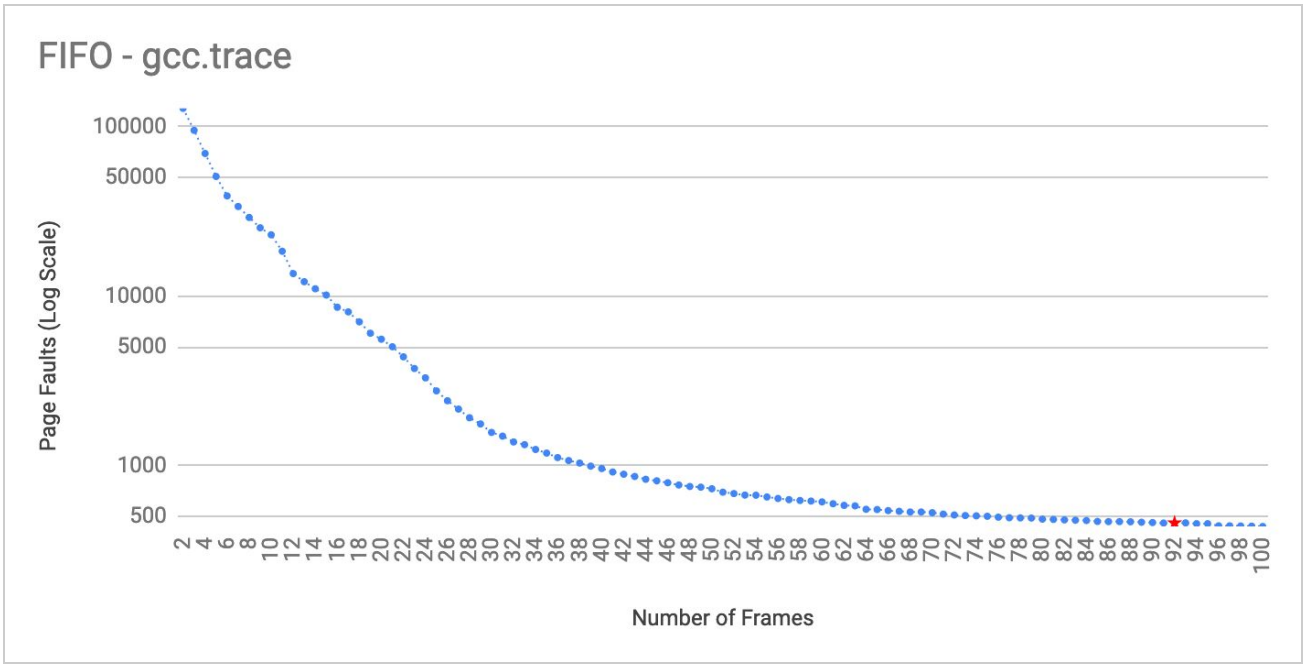
Therefore, I believe the aging algorithm is most appropriate to use in an actual operating system. It performs disk writes at a rate much closer to the optimal algorithm than FIFO which make it a strong candidate. Even though it causes more page faults than the optimal algorithm, this could be remedied through the use of machine learning algorithms to adjust the refresh parameter.

Belady's Anomaly

Belady's Anomaly is a phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern. This is very counterintuitive. However, it occurs in the `gcc` program. To find this, here are three graphs the show the number of page faults that occur as the number frames grow from 2 to 10. Belady's Anomaly occurs when the number of frames is equal to 92 on `gcc.trace`. The table data is shown on the next page.







Frames (gcc.trace)	Page Faults	Difference
2	127688	0
3	94726	32962
4	69158	25568
5	50623	18535
6	38848	11775
7	33664	5184
8	29011	4653
9	25202	3809
10	22916	2286
11	18325	4591
12	13529	4796
13	12118	1411
14	11014	1104
15	10111	903
16	8568	1543
17	8045	523
18	7026	1019
19	6018	1008
20	5550	468
21	5011	539
22	4370	641
23	3728	642
24	3283	445
25	2752	531
26	2407	345
27	2148	259
28	1910	238
29	1755	155
30	1565	190
31	1488	77
32	1375	113
33	1325	50
34	1241	84
35	1182	59
36	1112	70
37	1068	44
38	1032	36
39	990	42
40	958	32
41	914	44
42	887	27
43	860	27
44	828	32
45	811	17
46	790	21
47	766	24
48	751	15
49	744	7
50	728	16
51	695	33
52	682	13
53	668	14
54	667	1
55	651	16
56	638	13
57	628	10
58	621	7
59	616	5
60	609	7
61	594	15
62	581	13
63	577	4
64	551	26
65	549	2
66	541	8
67	536	5
68	531	5
69	531	0
70	526	5
71	516	10
72	510	6
73	506	4
74	504	2
75	501	3
76	495	6
77	491	4
78	490	1
79	489	1
80	482	7
81	480	2
82	477	3
83	475	2
84	473	2
85	467	6
86	466	1
87	466	0
88	465	1
89	462	3
90	460	2
91	457	3
92	**458**	**-1**
93	458	0
94	453	5
95	453	0
96	440	13
97	440	0
98	439	1
99	438	1
100	436	2