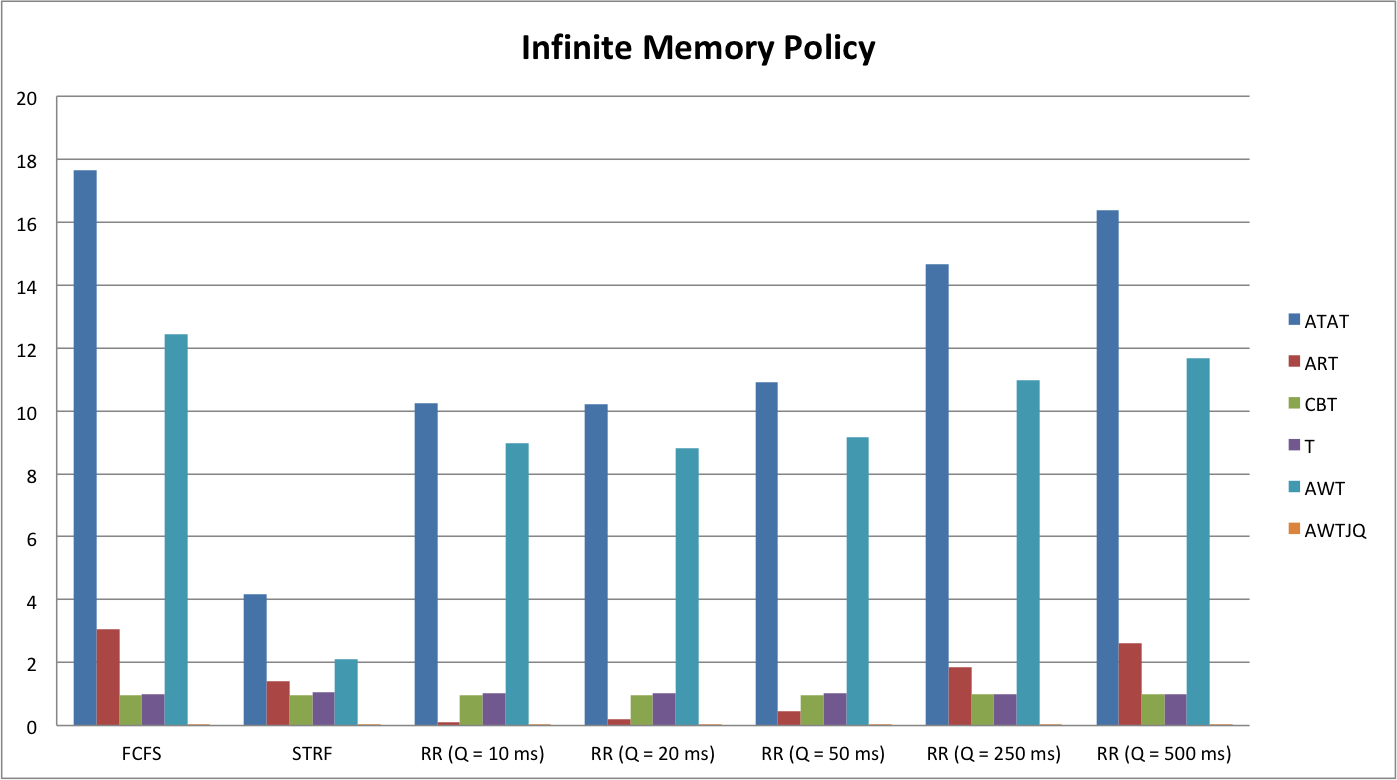
Walter Matthews

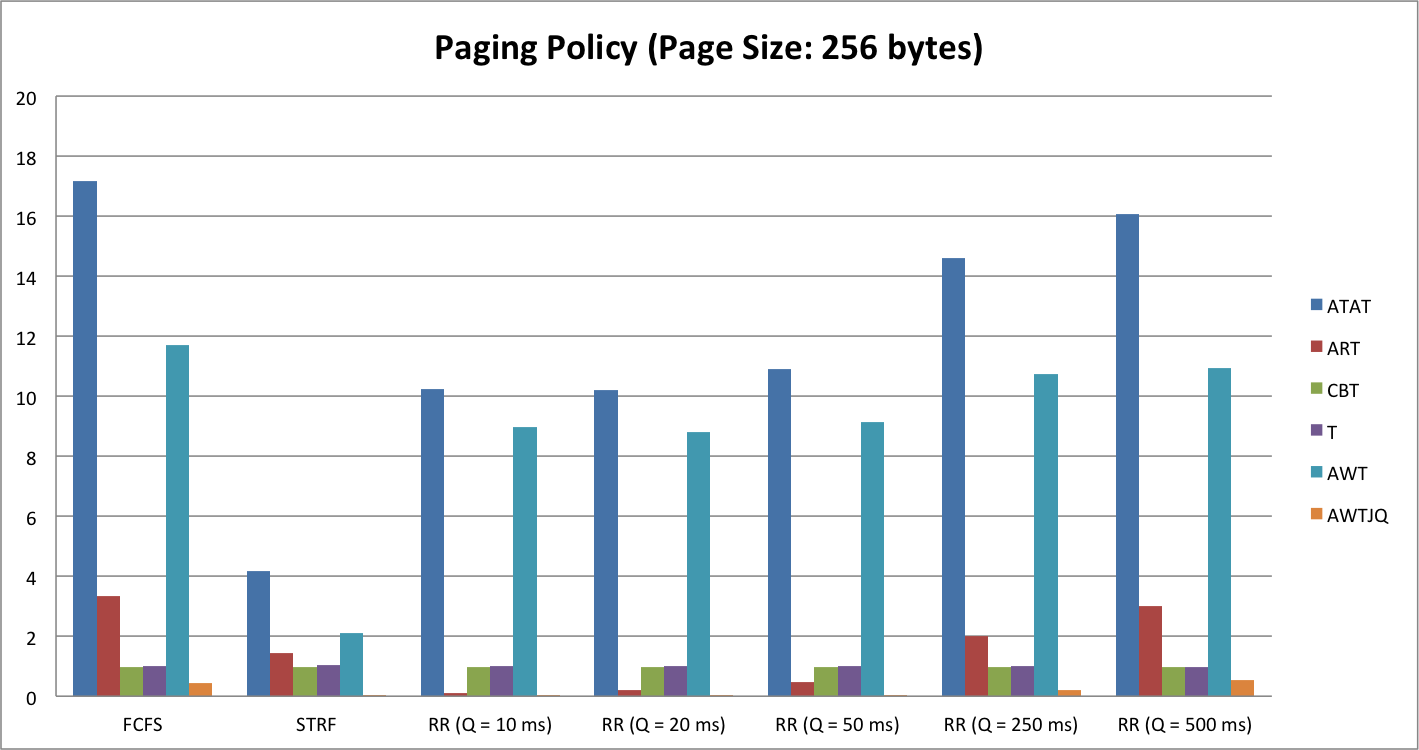
Adam Brown

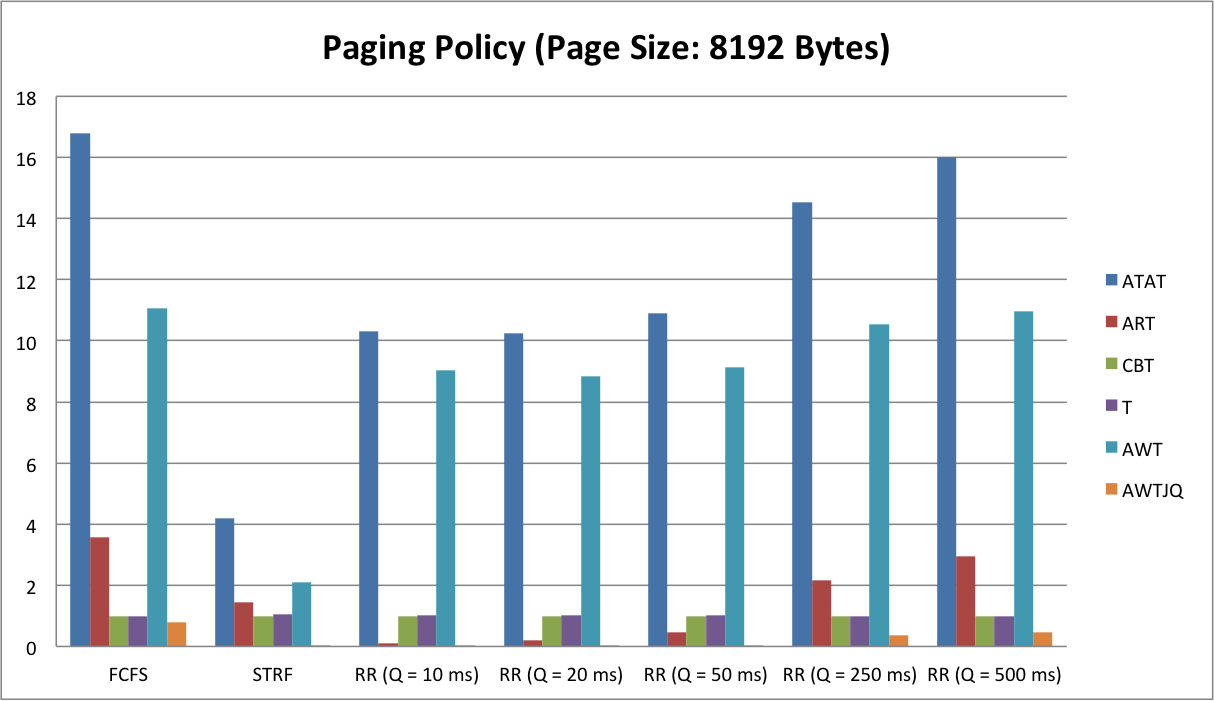
**Lab 2 Analysis**

Our code successfully compiled and executed for all Memory Policies. We designed and implemented in C. We simulate processes in order to test different Memory Management Policies. We see the same metrics from the previous lab with an addition of ‘Average Wait Time in Job Queue’.

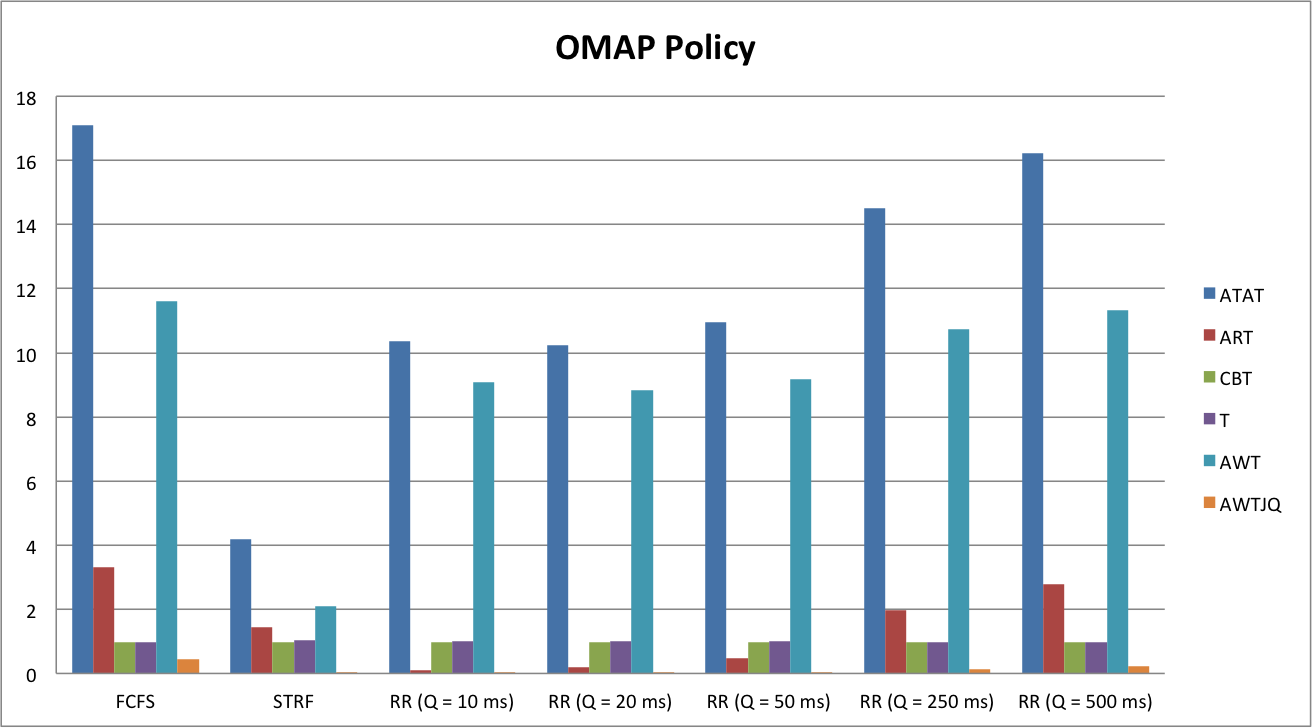


In this program, we act is if we have limitless memory to work with. Although this is impossible in the real world, we can use it for good comparison of metrics. A large discrepancy exists compared to the other memory policies. We see this discrepancy with the metric, Average Wait Time in Job Queue (AWTJQ). With an average of 0.00102, we can see a system with infinite memory would have no need to have jobs waiting in the Job Queue as they can continuously be admitted into the system.

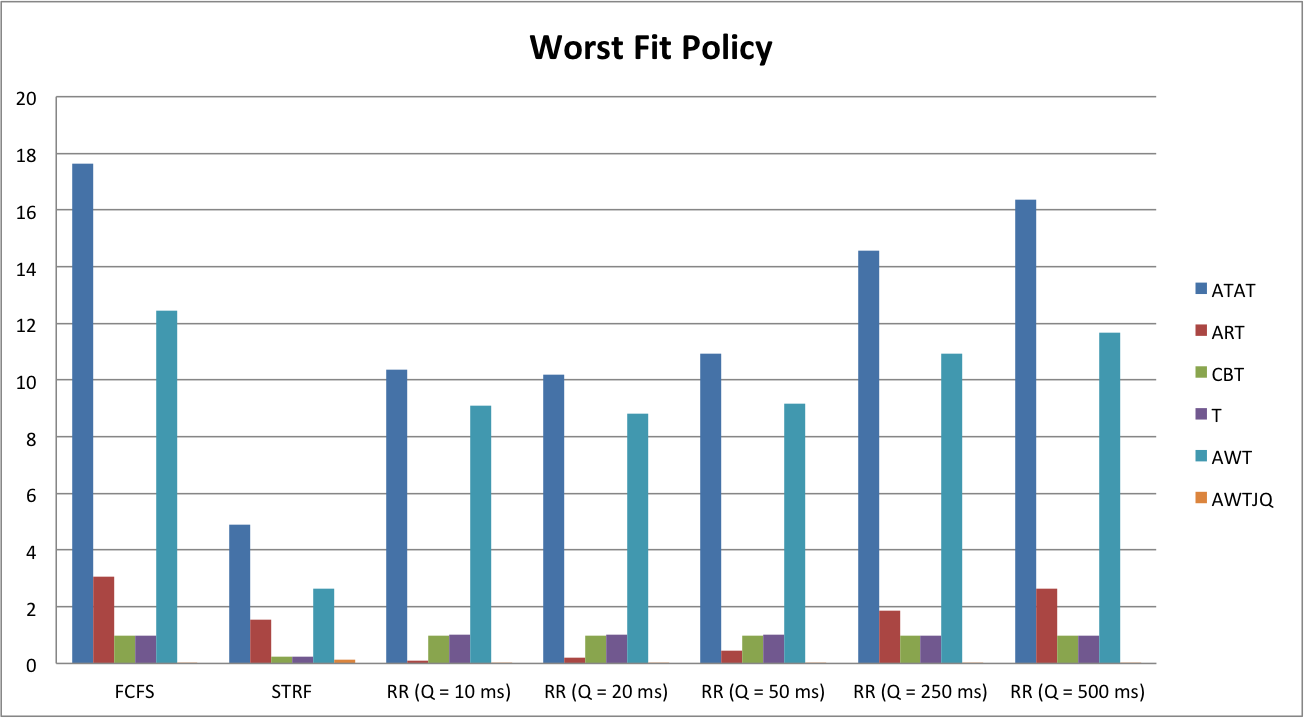


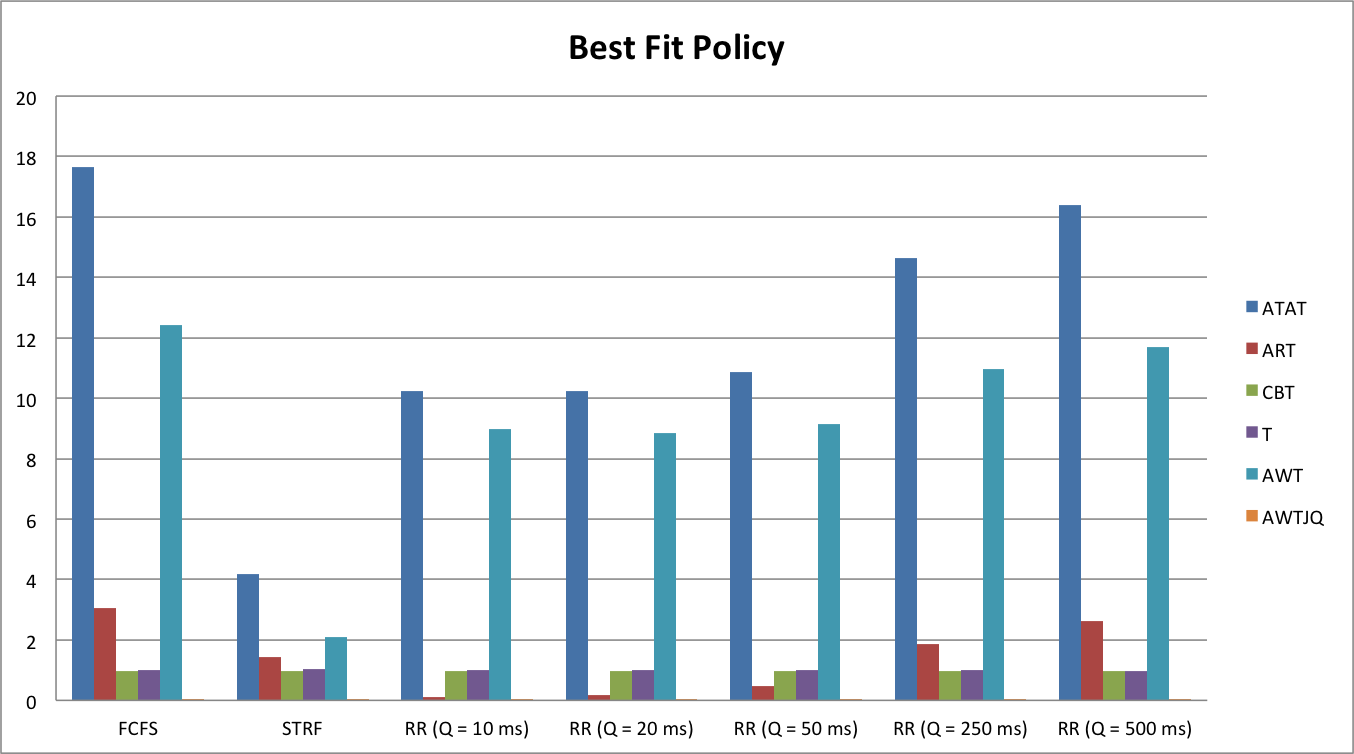


Here we see the results of 2 separate executions implementing the Paging Memory Policy. Paging allows a process to be noncontiguous and split its addresses into “pages”. Here we have 2 different page sizes: 256 bytes and 8192 bytes. Turnaround Time, Response Time, CPU Busy Time, Throughput, and Average Waiting Time were all very similar in both page sizes. We do see a discrepancy in the AWTJQ between the page sizes. The program with the smaller page size has almost .1 less units. Fewer pages can be admitted into the memory at once with a larger page size causing the Job Queue to clog. AWT is lower than all other memory policies with paging policy. Paging allows processes to get in and out of the ready queue in a timely manner.

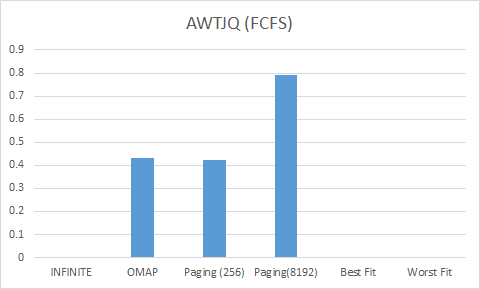


Here we see the implementation of the Optimal Memory Allocation Policy (OMAP). Here we introduced the fact we only have a limited amount of memory in a system and we can only accept processes that will fit when we have enough memory available. Here we see a better AWT compared to at least half of the other memory policies. OMAP did not have a very optimal value for the AWTJQ metric. We assume this because processes will be rejected from the memory frequently since it has to wait a lot for processes to release memory after being completely processed.





Here we implented



Here we can compare the AWTJQ for the different memory allocation policies. For the data we collected, we observed much higher values for the OMAP and simple paging relative to the others. Worst fit was the most efficient of the feasible policies at just slightly slower than infinite, followed very closely by best fit. I do not think that the data we collected here is perfectly sound as OMAP should perform better than the rest on average. Perhaps we experienced some outliers or there is an unknown issue in the code. We can see however that the increase of page size caused an increase in the waiting time. This was as anticipated because processes tend to waste more memory when assigned a frame too large.

To conclude, although the data we collected may have had variance due to outliers, we could determine that worst fit and best fit perform better that simple paging on average, likely due to compaction allowing the memory to be allocated more efficiently.