PA4: Threading and Synchronization
Christopher McGregor - 726009537

1 Design

This PA focuses on using threads to increase efficiency in running of a program with large amounts of data requests. I will give a high-level overview of each created function.

1.1 Patient threads

Patient threads request 'n' data points per patient. This request is not directly to the server, instead we generate the request and push it to a request bounded buffer.

1.2 File threads

File threads request a piece of the file from the server. This is done by first getting the entire file size and generating as many requests as needed to eventually return the whole file. Like patient threads, we do not request directly from the server, but instead generate a file request and push it onto a bounded buffer.

1.3 Worker threads

Worker threads are the brains of the operation. The thread pops the top request from the bounded buffer and handles it accordingly. If the request is for data (what our patient threads request), we write the request to the server and push the response onto a response bounded buffer (since the histogram threads handle the responses). If the request is for files, we write the request to the server, read the response and then write the response to our copy file making sure to write in the correct location as threads are not always run linearly. Otherwise, if the request is for 'quit' we write the request on the specified channel and delete the channel to save against memory leaks.

1.4 Histogram threads

Histogram threads handle data responses and update their respective histogram. Since each person gets their own histogram, we send a special histogram packet from the worker thread which contains the person number and response. We use this to update the histogram accordingly. If the workers are done requesting, they send a packet with the person number to -1 which we handle as a quit message that breaks the thread.

1.5 Alarm handler

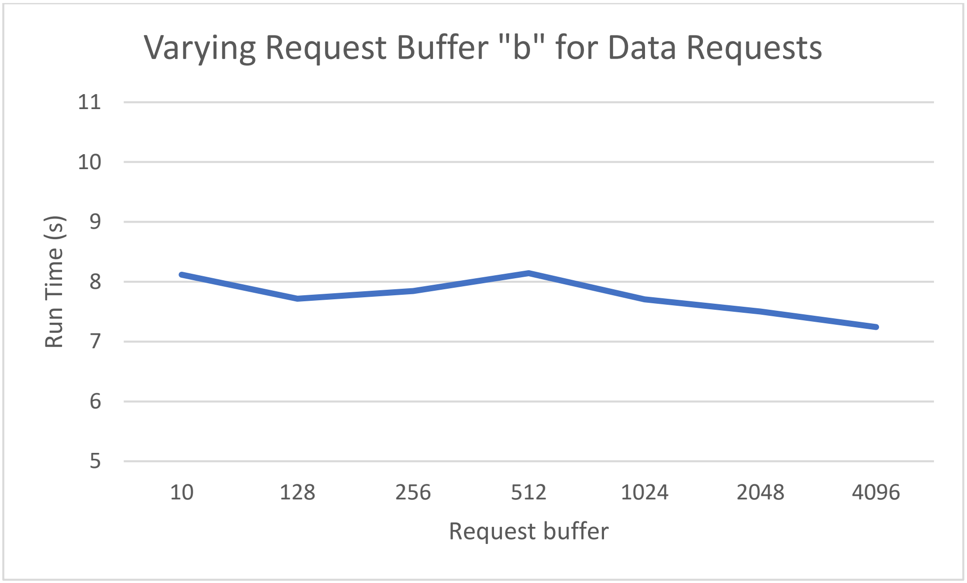
This is for the bonus portion which prints the histograms update every 2 seconds to the console. This is achieved by clearing the screen, writing the current histogram data, and resetting the alarm.

2 Timing Data

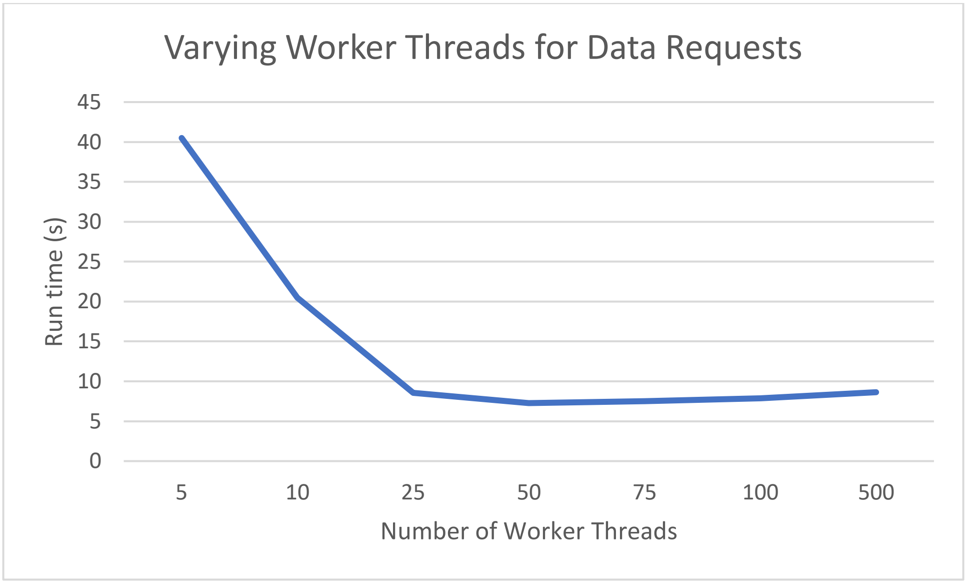
We plot run times against varying parameters for both data request and file requests.

2.1 Data Requests

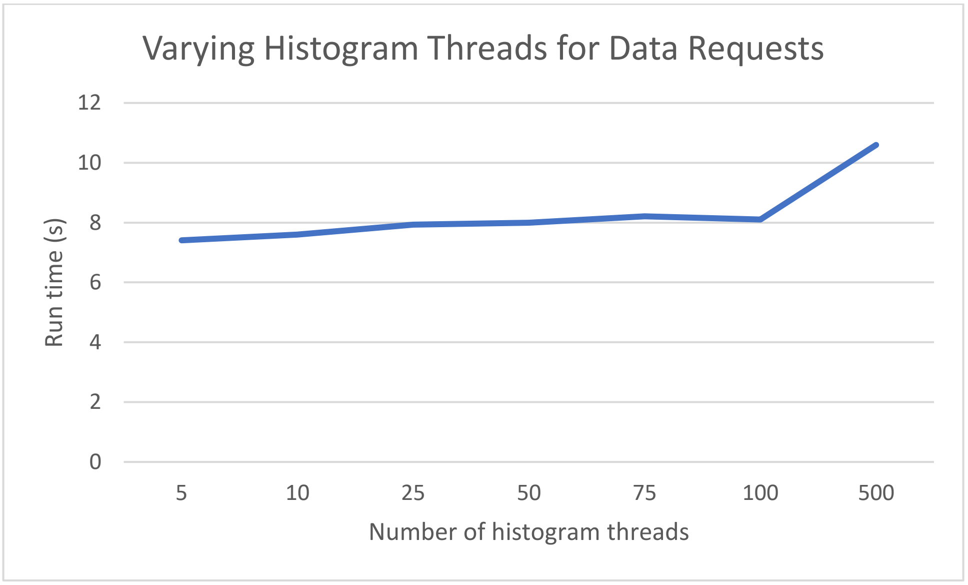
We test the performance of the system with varying number of worker threads, histogram threads and size of the request buffer with 15000 data points of 5 patient.



We note that with increase buffer size performance does not appear to change much. With a very small b there is some slow down, but b about 20 and greater seems to have no effect. This is likely because after a capacity of ~20 the producer / consumers are able to push and pop without hitting the buffer limit.



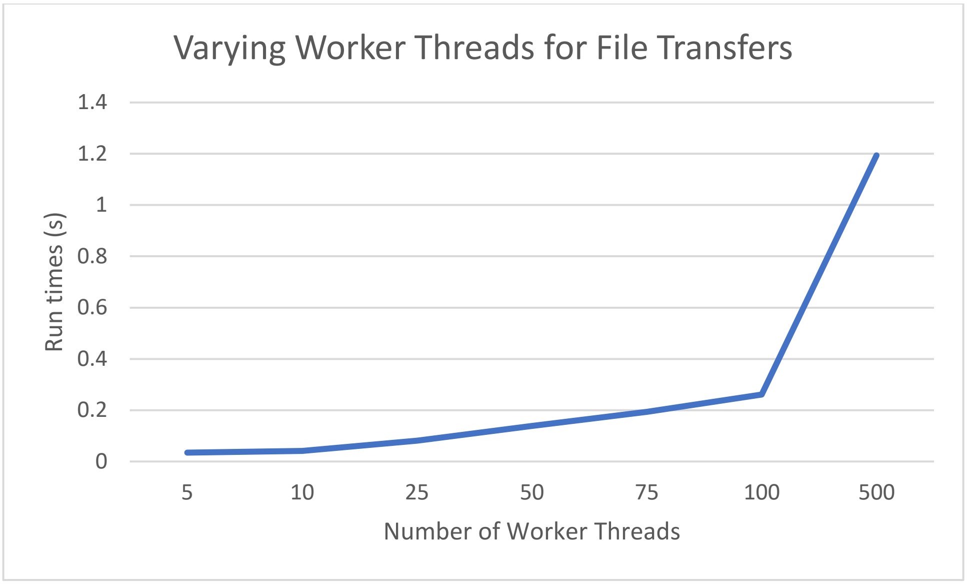
Here we notice that increasing workers does improve performance to a point. This is because more workers can fulfill more requests from the bounded buffer which speeds up run time. We notice a shift in improvement around w=50. This is likely due to having too many workers in relation to producers which means workers are sitting idle and further increase to their quantity increase overhead without improving performance.



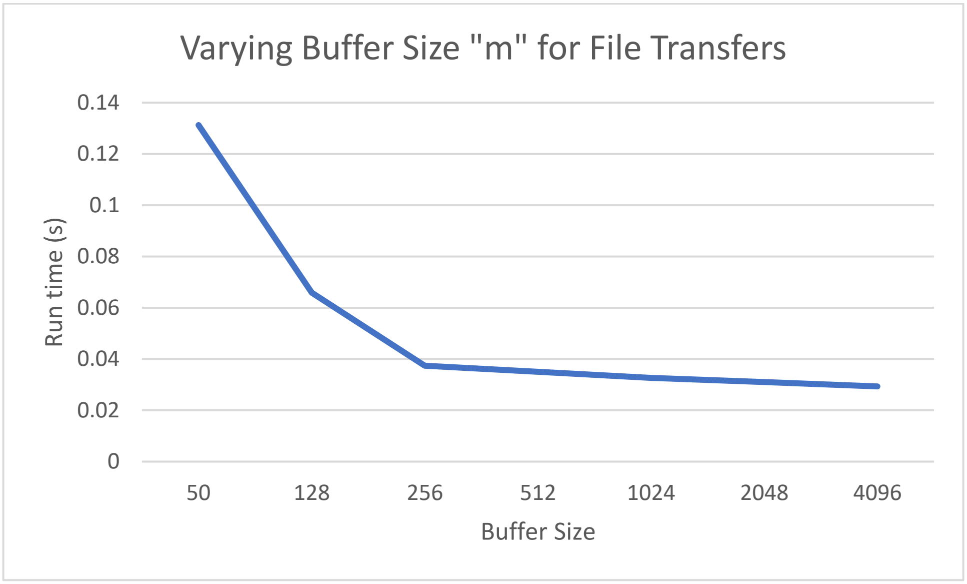
Here we notice very little change in performance when increasing histogram threads. This is likely because with my implementation the workers still do the heavy lifting as far as requesting data from the server. What is fed to the histogram threads is the response which takes little to no time to process and update the histograms. After ~3 threads performance seems to decrease slightly even as this requires much more resources for no efficiency trade off. It is worth nothing that should the histogram threads be the ones to request from the server, we would see a trend line much like the worker threads instead.

2.2 File Requests

We test the performance of the system with varying numbers of worker threads and the size of the request buffer with a uniform file size of 100000 bites.



We notice that increasing worker threads decreases performance. This is because more workers do not introduce load balancing since the file threads are the bottleneck while introducing much more overhead.



Here we notice an increase in performance when the buffer size increase. This trend should continue until our buffer size is equal to and greater than our file size. At this point, there will be no further increase to performance. This is because we need fewer total requests since the increased buffer size allows more returned data. Using an overly simple example, a file with size 100 and buffer 10 needs 10 requests. However, if we have the same files size 100 and now a buffer of size 100, we only need 1 request and anything over this will still require one request.