Adrian Gamez-Rodriguez

CSCE 313-515

UIN:126009409

11/17/19

Report

The purpose of this machine problem is to rethink the performance of the client and data server programs. Instead of having multiple worker threads each with its own channel, we should have **one** worker thread to take care of **N** request channels to the data server. We still have 3 request threads and 3 statistics threads. This method differs from MP3 in that we still have high-performance multiprogramming however we have fewer threads and concurrency is achieved through event handling programming by using the "select()" call to monitor which request channels are ready to be read from. We used the select() call because the blocking call of "send_request()" was broken down into a non-blocking call "cwrite()" and the blocking call "cread()". The "cread()" is blocking because we have to wait for a response from the data server.

The results are shown below.

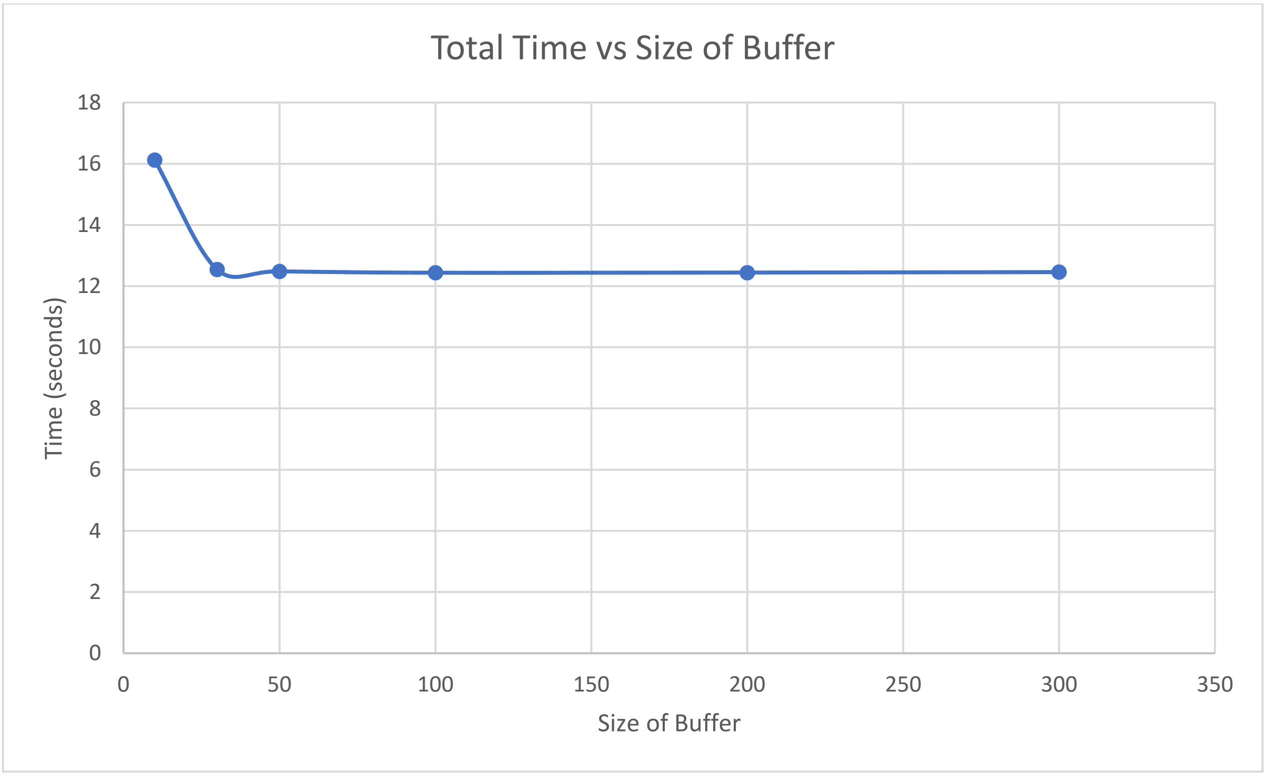
NOTE: To run program, type "**make all**" and then type the command "**./client**"

NOTE: There are default arguments of 10 request per patient, size of buffer is 50, and 10 request channels. To change arguments, simply just include the arguments with flags on the command line. (i.e. "**./client -n -b -w** " where n is the number if data requests, b is the size the buffer, and w is the number of request channels.

Time vs Size of Buffer

Constant: Number of Request threads: 120

Constant: Number of Requests per patient: 15000

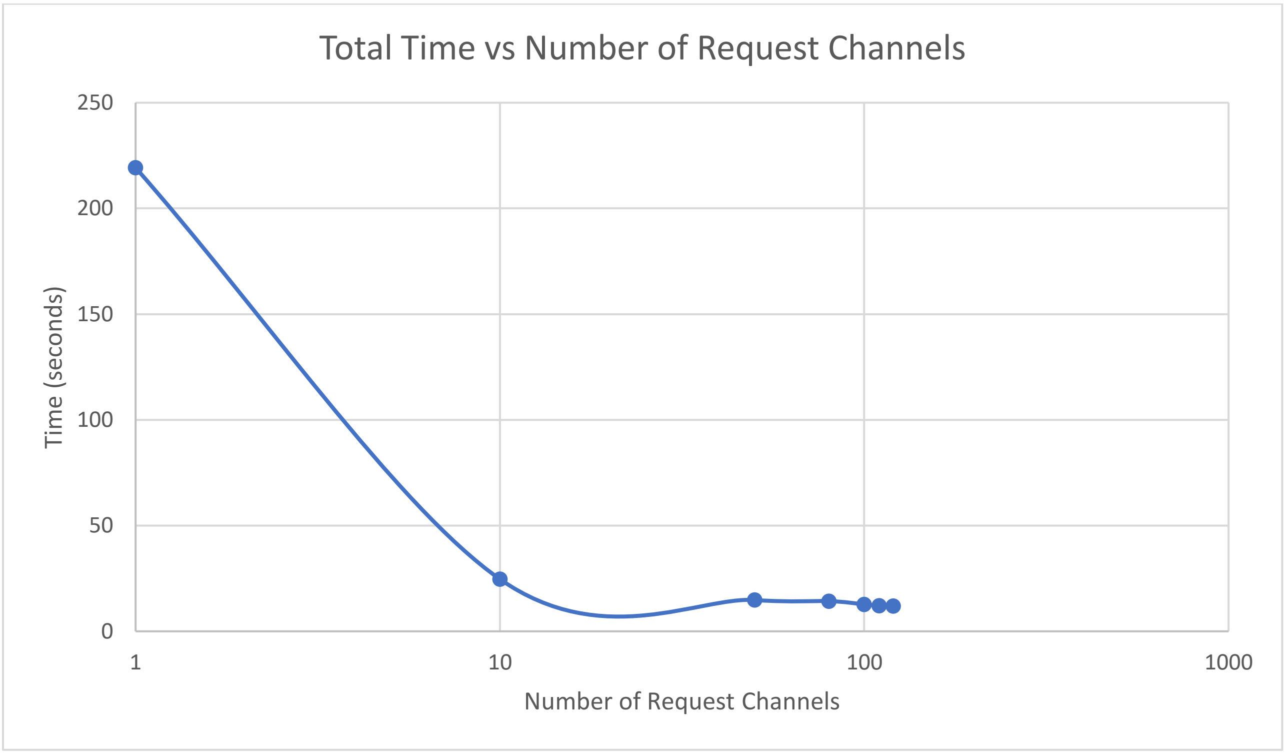


Here we see that the size of the buffer affects the execution time up to a certain point. This is the same result as for MP3. I varied the size of the buffer from 10 to 300. We see an initial drop of time however it plateaus, and the size of the buffer has no effect on the time. I kept the number of request channels to 120. These results make sense, because if we increase the size of the buffer, then this allows for more requests to be pushed on to the buffer, but up to a point.

Total Time vs Number of Request Channels

Constant: Size of the Buffer: 50

Constant: Number of Requests per patient: 15000



From the graph above we see that if we keep the size of the buffer constant and the number of requests constant and vary the number of request channels, we get a drastically faster total time of program completion. I varied the number of request channels to the following quantities: 1, 10, 50, 80, 100, 110, and 120. We see that eventually the number of request channels will only affect time up to a certain point and the results of the graph will hit a plateau. This makes sense, because if we increase the number of request channels, we in turn increase the number of requests the data server will respond to.

Comparing MP3 and MP4

My results in MP4 are very similar to that of MP3 however, I tested my MP4 on my local machine using the windows subsystem for linux and that might have affected my final results compared to that of MP3 which I had tested using the TAMU server. However, we see that very similar results. Increasing the number of requests without a doubt still improves performance by a significant amount in terms of total completion time. There is a point in which increasing the request channels does not affect performance, that point is around 120 request channels. If we varied the size of the buffer, we see the same results. Increasing the size improves performance up to a point. However, the major difference between MP3 and MP4 is the amount of threads we had to create. In MP4 we only have a total of 8 threads including the main thread. Thus we achieve the same results with fewer threads.