PA4 Client and Server with Treads

CSCE 313

Mario Sandoval

Texas A&M University

For this assignment, we were asked to implement a client program that will connect with a server with the help of different threads. The idea behind the use of threads is to utilize the maximum amount of CPU to request data to the server. The request created by the patient's threads was stored in a request buffer to then be utilized by the worker thread and sent to the server on a new channel. The results of this request were then brought back and stored in a response buffer to be utilized to create the histograms. This bounded buffer was implemented with the help of semaphores to make it thread-safe, and prevent corruption of data. Furthermore, the client program implemented a file transfer from the server-side. This was done by creating a separate thread that would handle the creation of file requests to the server and add them to the request buffer. These requests were handled by the worker threads and sent to the server to be handled. The response was then bought back and added to the newly created file.

Data Requests:

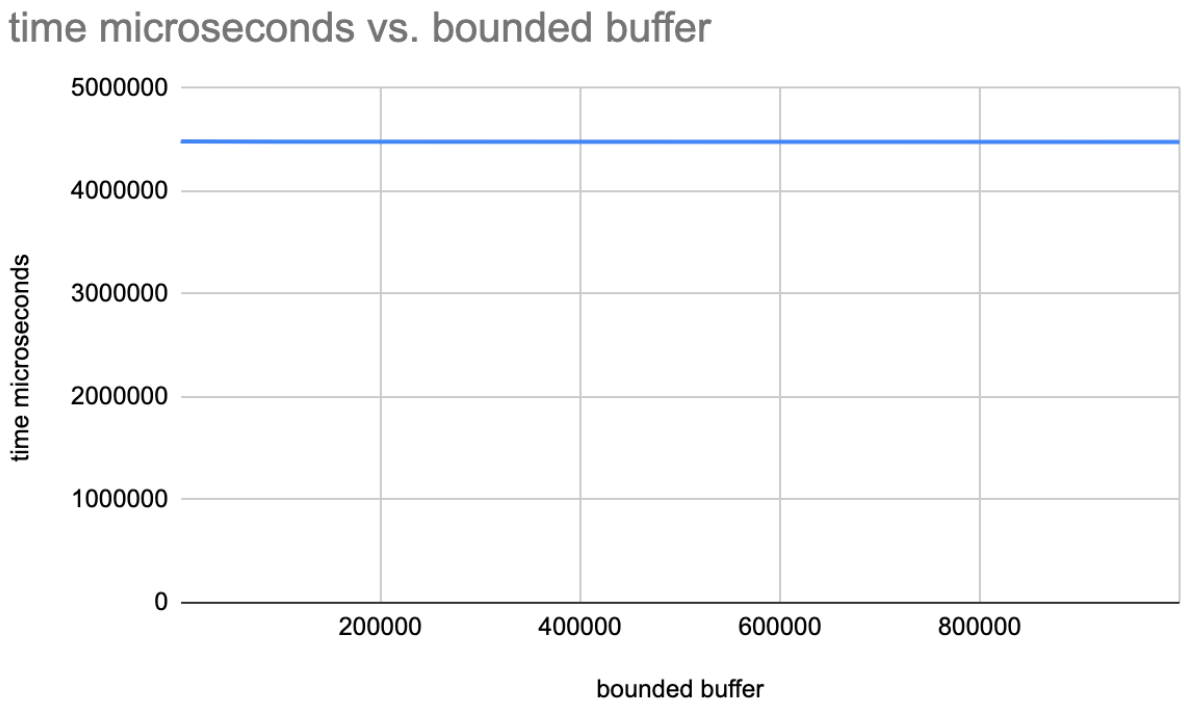
To test the effectiveness of the data transfer with threads, there was a variation of the size of the bounded buffer, as well as a different number of worker threads. Both of these variations will be requesting 15k data points from10 patient files and using 10 histogram threads.

Command

```
./client -p 10 -n 15000 -h 10 -w 100 -b XXXXX
```

bounded buffer	time microseconds
100	4494839
1000	4482597
10000	4482785
100000	4481722
1000000	4478953

Table 1. Bounded Buffer vs Time (microseconds)

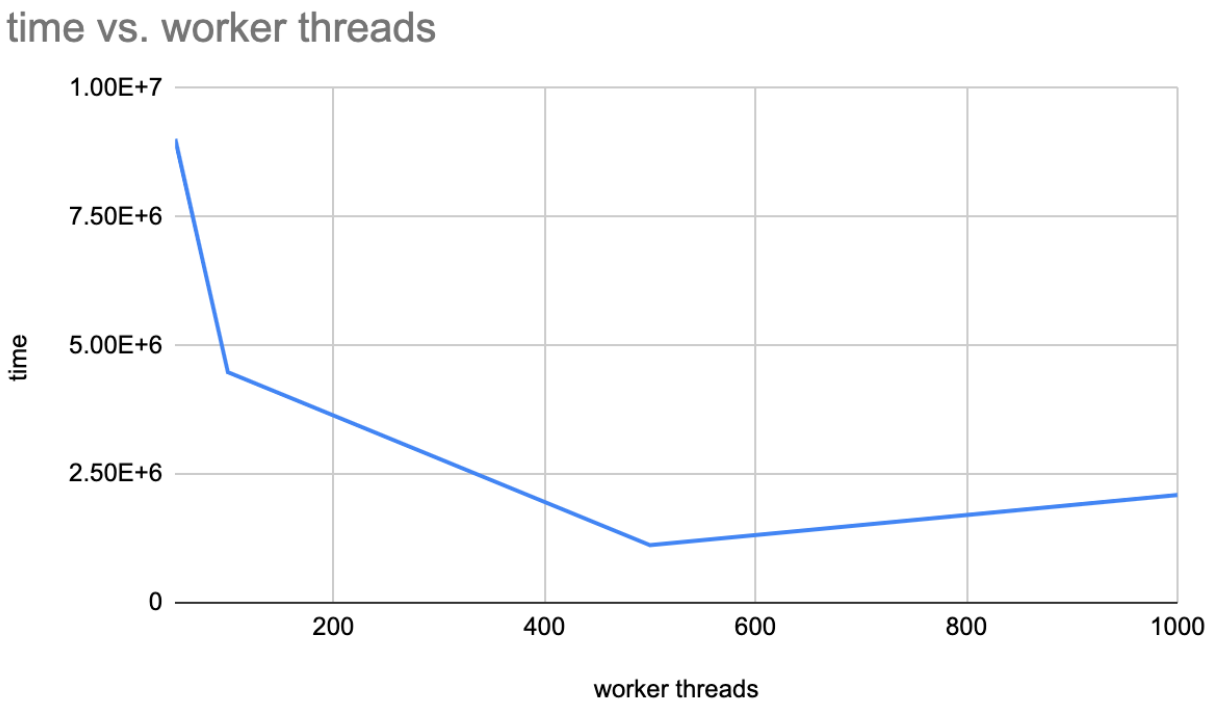


Graph 1. Bounded buffer size vs Time

Command:
./client -p 10 -n 15000 -h 10 -b 1000 -w XXXXX

worker threads	time
50	9013622
100	4482001
500	1119303
1000	2091794

Table 2. Worker Threads vs Time



Graph 2. Number of Worker Threads vs Time

File Request:

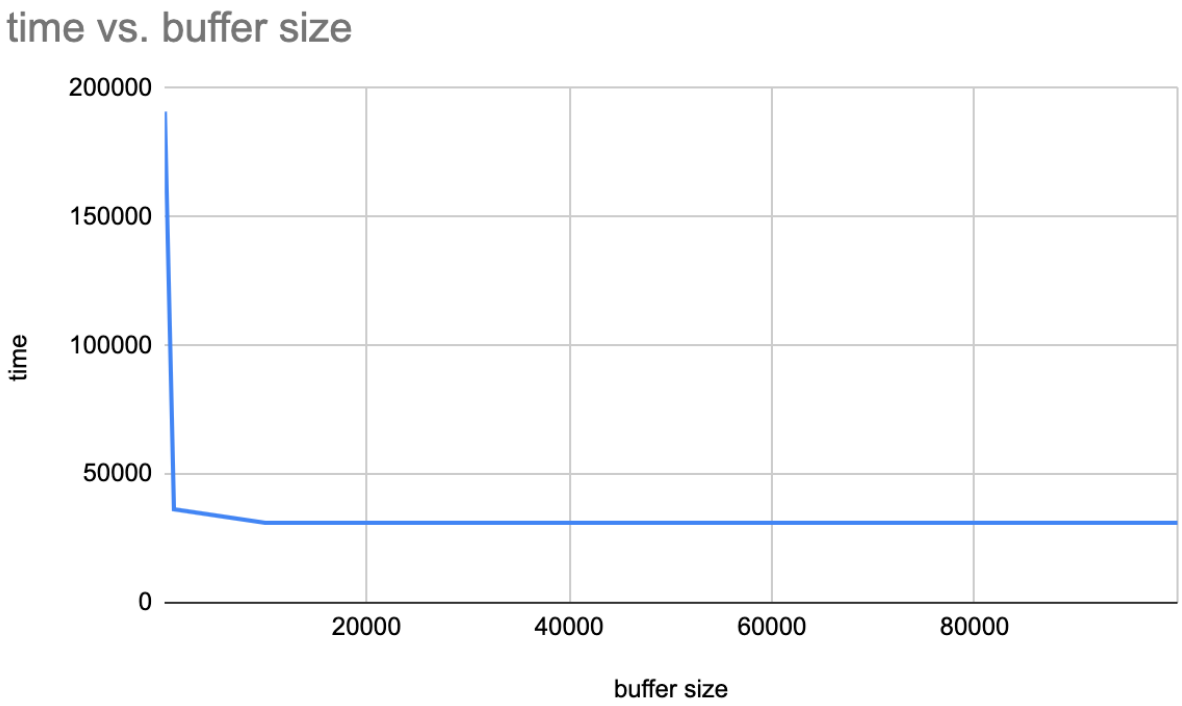
To test the effectiveness of the file transfer, there were a different number of worker threads and different buffer sizes. For this test, I transferred the 1.csv file from the server with no other different parameters.

Command:

```
./client  -w 100 -f 1.csv -b 100 -m XXXX
```

buffer size	time
100	190887
1000	36301
10000	31082
100000	31038

Table 3 Buffer size -m vs Time

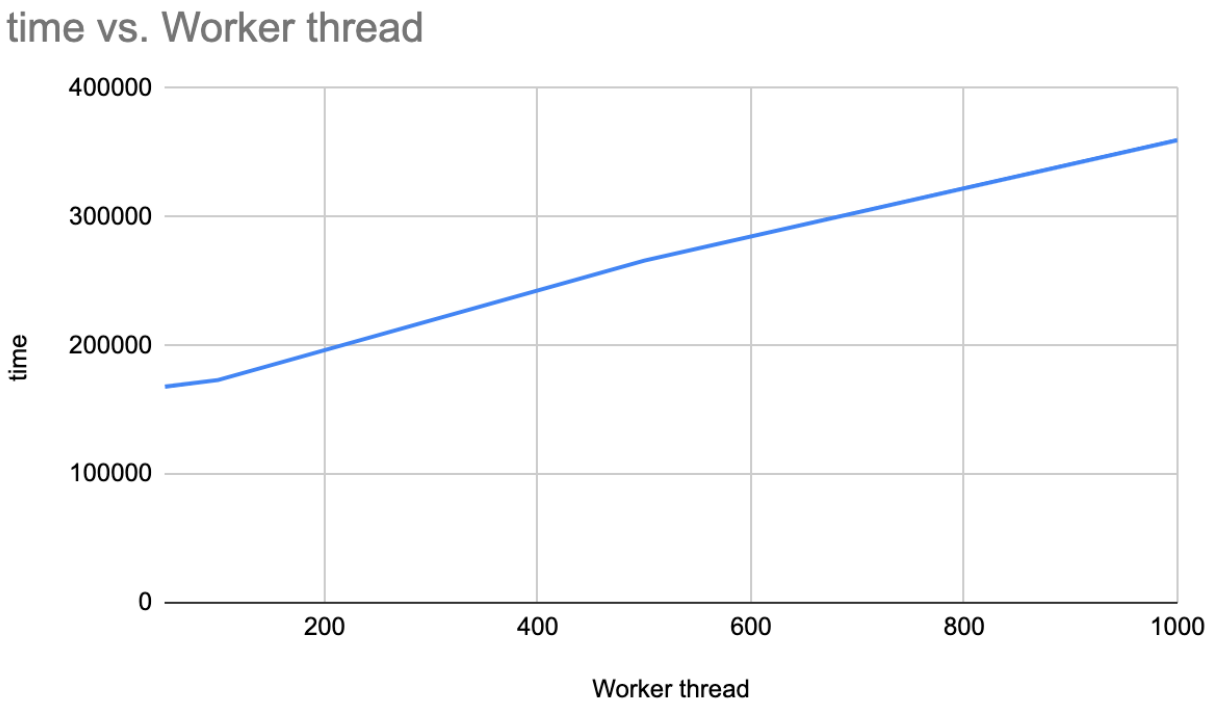


Graph 3 Buffer size vs Time

Command:

```
./client  -b 100 -f 1.csv -m 100 -w XXXX
```

Worker thread	time
50	167894
100	173102
500	266049
1000	359715



Number of worker threads vs Time

Conclusion:

The acquired data showed the results of running the program with different buffer sizes -b and different number of worker threads -w. For data requesting there seems to be a slight difference in the time taken to execute the program, however the difference in time is so minimal that the graph appears to be contant. While we saw a time decreased as the number of threads increased meaning it has an opposite relationship; The time however hits a plateau of effectiveness after 500 worker threads, for the time starts to increase after 500 threads.

For the file request I changed the size of the buffer in the server -m and the amount of worker threads to construct the file. The data shows that after the first 100 bytes the time starts decreasing up until reaches 1000 bytes then it remain somewhat constant. While the time seemed to increase with the number of worker threads created. This attributed to there only being 1 single thread creating the file requests and adding it to the request buffer, so a lot of those worker threads had to remain stalled and consume time trying to access the buffer. This is also attributed to the change in buffer size creating an almost constant runtime, since there is only one worker thread accessing the file request, regardless of the buffer size it would still take approximately the same amount of time to complete the task.