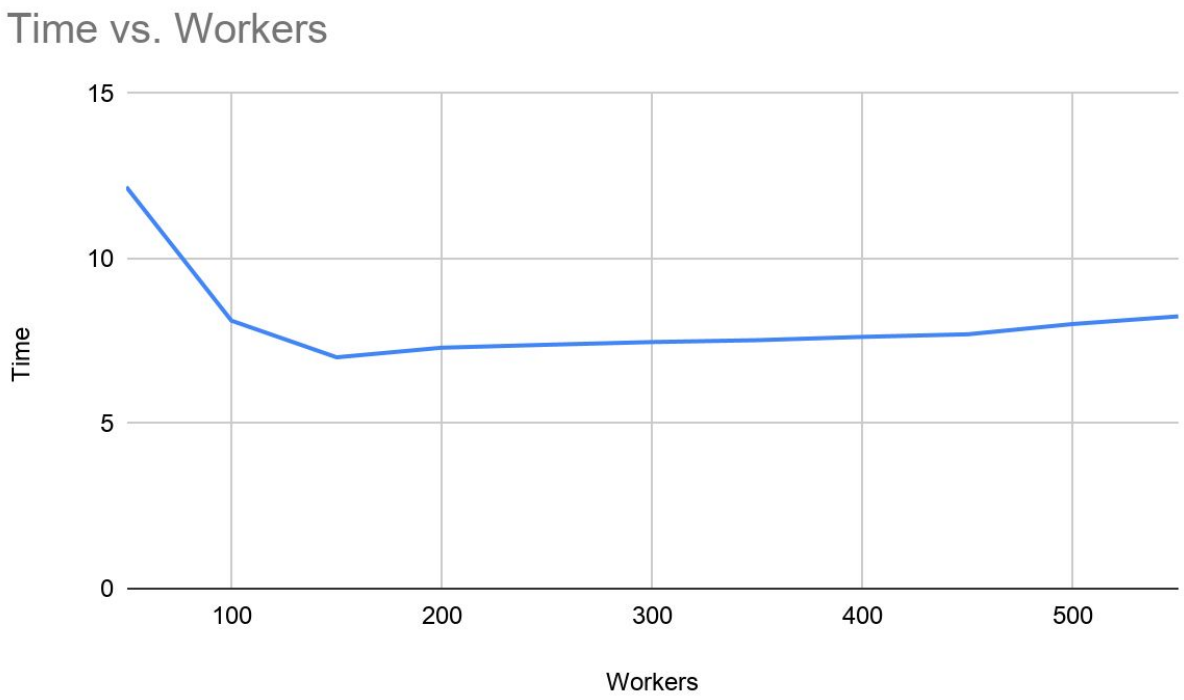
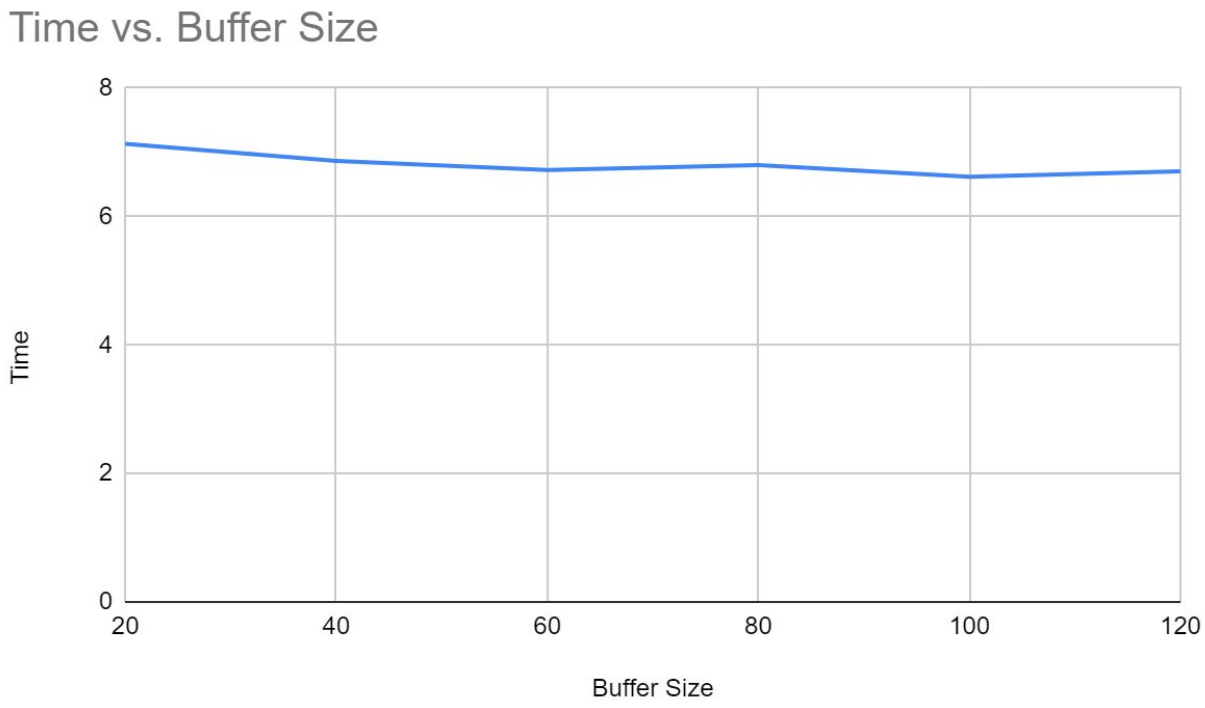
Thomas Hari Budihardjo
UIN: 126009556

PA5 Report

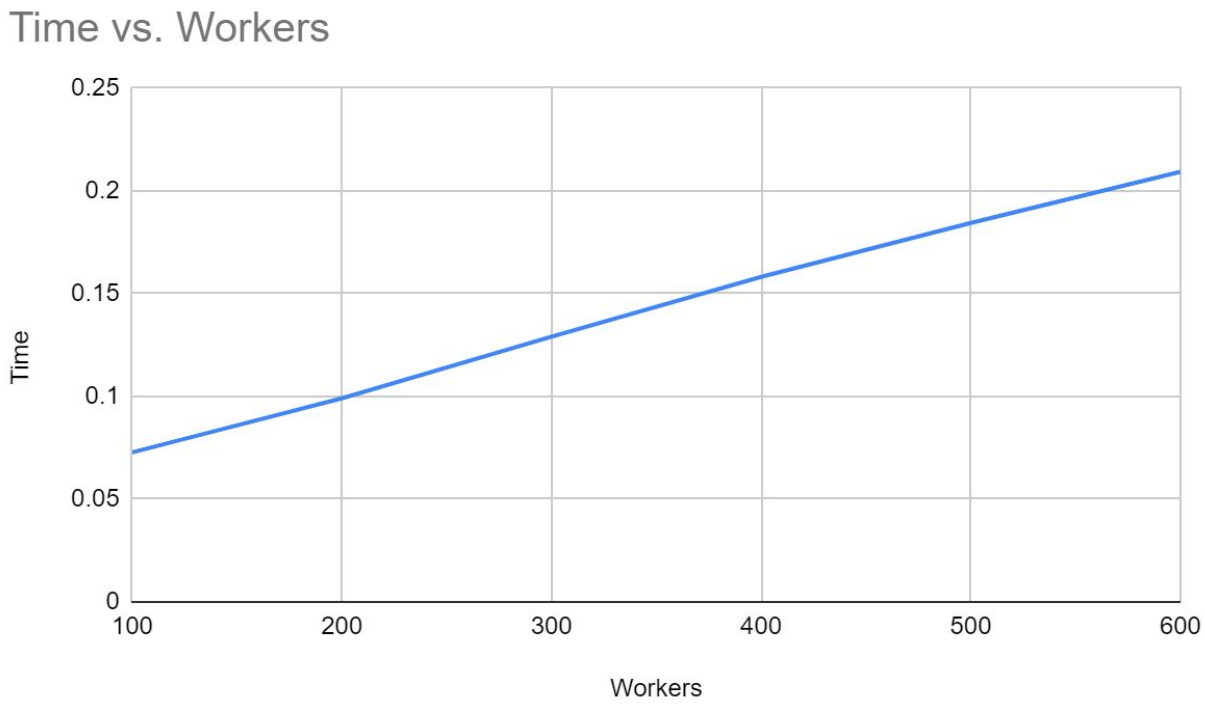
As before, the performance has a point of diminishing returns when varying the worker thread in requesting data points. It will take a long time with low amount of workers and will increase in speed as we add worker threads. This time though, the point of diminishing returns occurs at 150 workers (compared to 200 before). Increasing workers after this point will make the process take longer and longer. Initially the time it takes to complete the data transfer will decrease as we increase the workers because there will be more and more worker threads consuming the tasks in the buffer, but eventually the number of context switches that takes place due to having so many workers far outweighs the benefit of multithreading. This is due to the fact that the number of patient threads stay the same, which means the buffer will be empty almost always during the running of the program because the patient threads cannot keep up its production with the rate at which the worker threads are taking things out of the buffer.



As for varying the buffer size, again the time does not vary (it varies more than before, but not much more as we can see it is still contained between 6 and 8 seconds). Increasing the buffer size does not change the balance between the patient and worker threads, which means the time it takes to complete the tasks will stay constant. We can say that this relationship is linear.



Just as in PA3, varying the amount of workers in requesting files increases linearly. This relationship can be explained by the fact that there is only one patient thread, which means the workers are taking things out of the buffer at a rate that is faster than the patient threads are putting it in. Adding more and more workers will only increase the amount of context switches, which will add overhead that will make longer runtime.



On all three of the above graphs, the time it takes to complete the processes is larger compared to in PA3. This is due to the fact that we are using the TCP/IP protocol instead of FIFO. The reason for this is because FIFO named pipes is faster than TCP/IP on local networks (“localhost” is what I tested to produce the above data, therefore I ran the TCP/IP protocol on the local network). A reason for its faster performance is because named pipes is run in kernel mode. Another reason is because the FIFO protocol is much simpler because of the fact that it doesn’t have to deal with inter-computer communication, which makes TCP/IP more complex.

```
function runpa5 {
  local n=4
  for i in `seq 1 $n`; do
    ./client -n 2500 -b 25 -h 127.0.0.1 -r 50001 -p 10 -w 256&
  done
}
```

For the 2 point bonus, I ran the above script which runs n number of clients at the same time, varying the number of client processes I'm running by changing the n value. The results of this are as graphed below. I believe as more and more clients try to run on the same port it slows the network down because the network has to redirect the a greater and greater number of requests into different slave sockets which will require an increasing amount of time as the number of clients go up.