Raul Escobar
328003859

# Programming Assignment 4 Report

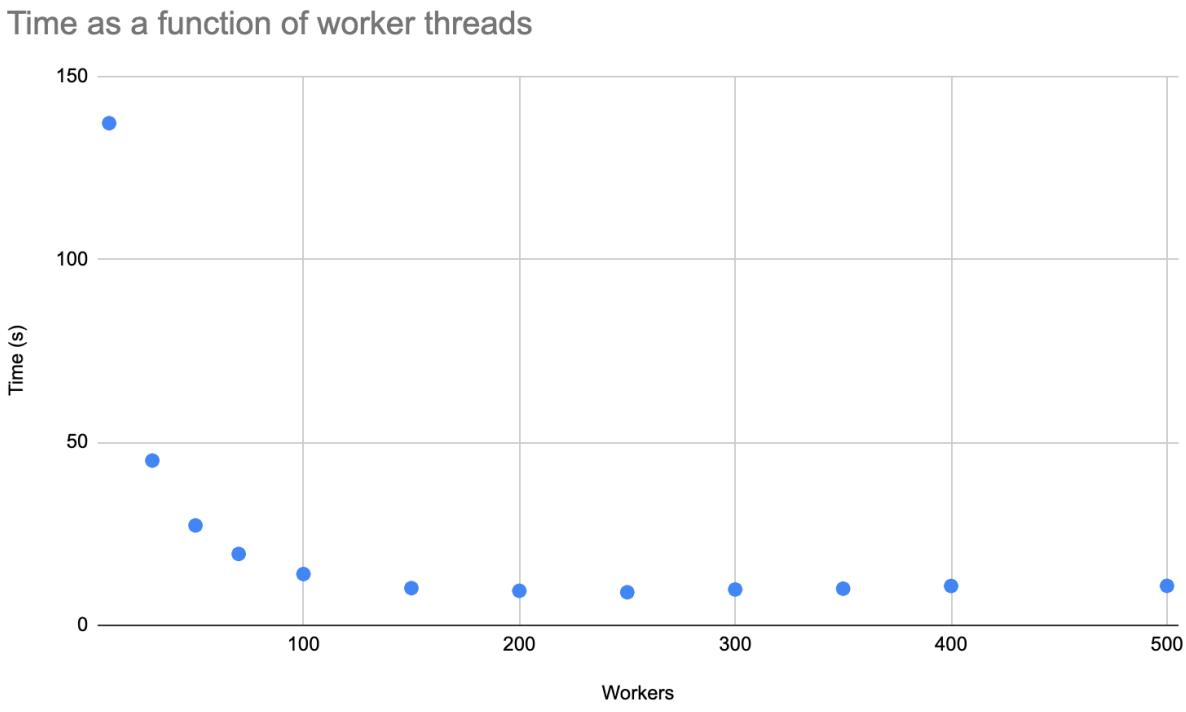
video url: did demo in person Tuesday lab

constants unless specified otherwise:

- -p 15
- -m 256
- -h 15
- -w 150
- -b 2 (for some reason it goes like 2 seconds faster at low b)
- -p 15
- -n 15,000 (chose this number because 15,000 would take too long per data point)

All the graphs are done where everything except the variable in the x-axis is constant as defined above.

## Graphs for data requests:

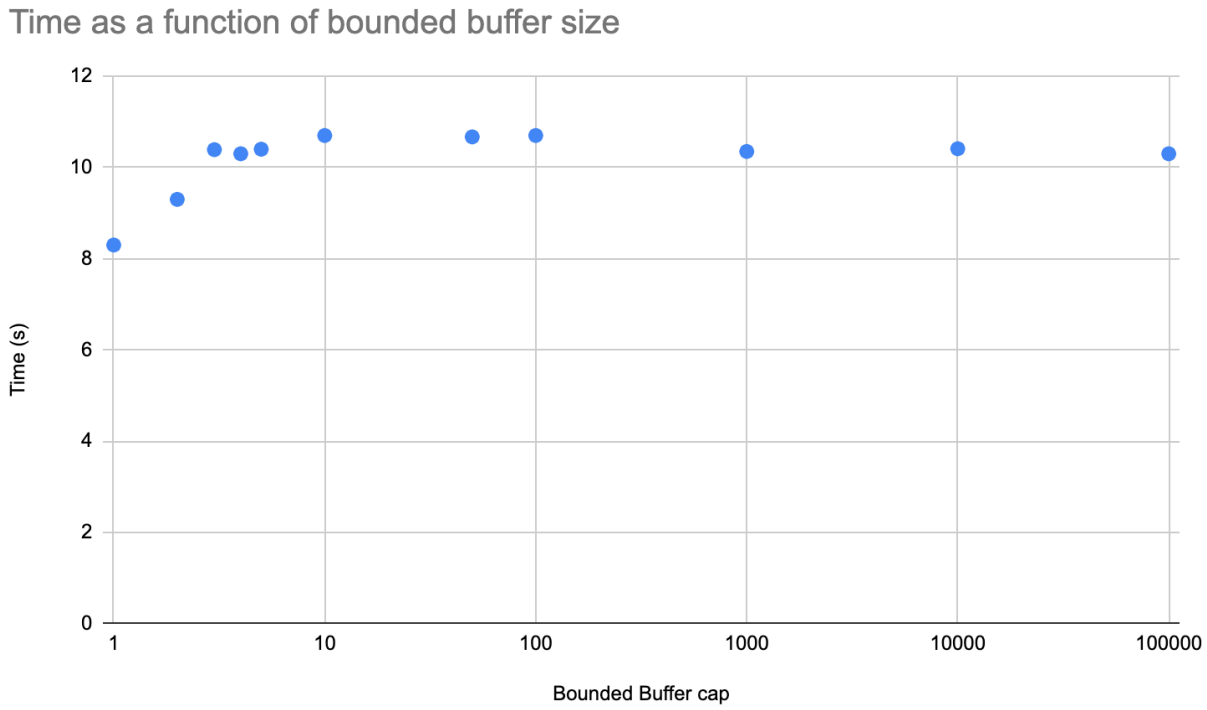



From the above graph, we can see how at 10 worker, it takes ~140 seconds, then at 30 workers, the time is reduced to 50, then at 50 workers it takes about a 30 seconds. So like as expected, there is a w-fold speed up as w is increased. We can also see how there are no more

Raul Escobar
328003859

speed ups after 250 workers, after 250 workers the task takes about 9 seconds no matter how many workers are added. After 150 workers there are diminishing returns in the sense that, for any extra workers we only get a small performance boost.

They do not split up the work evenly from the start, rather, they each do as much as one can do. So if we have a slow worker for whatever reason, he picks up less work from the bounded buffer, and fast workers pick up more “tasks”. This squeezes as much performance from each worker as possible.



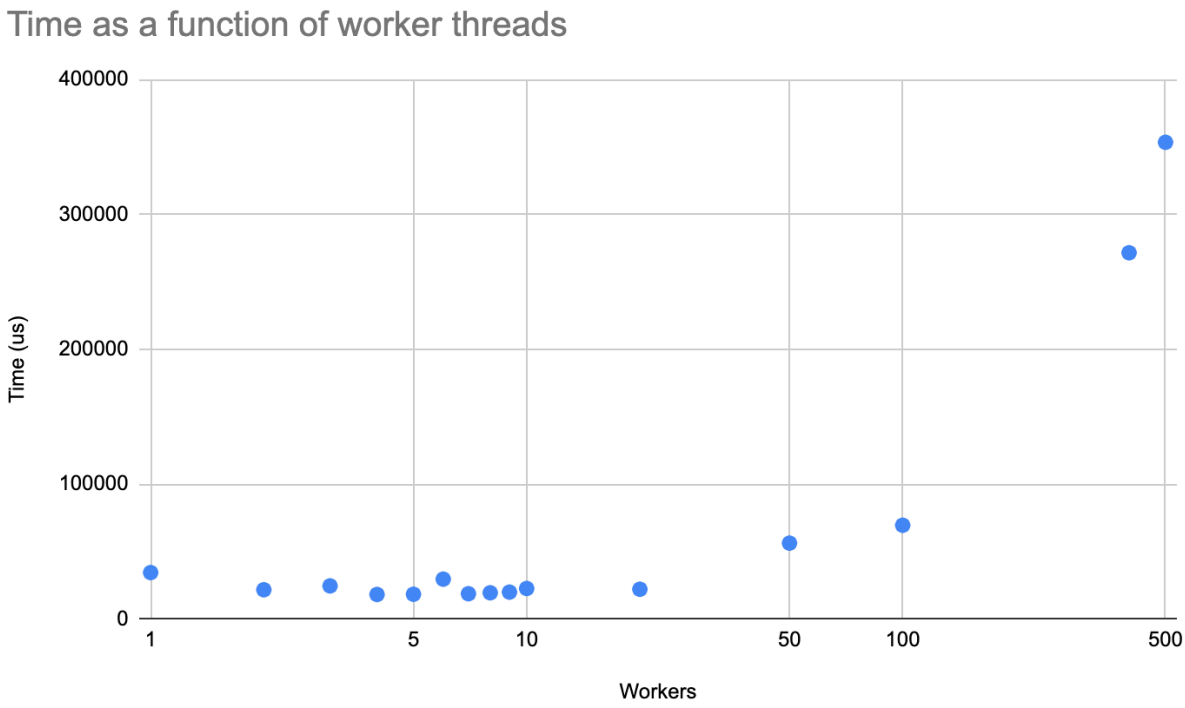

On the other hand, as we increase the size of the bounded buffer, there is no noticeable performance difference. One explanation is because the bounded buffer can only queue or deque one item at a time if its size 1 or if its size 1,000,000.

If the bounded buffer is of size 1, then the workers simply wait for the queue to be empty for them to queue their task. This in essence creates a queue of workers waiting for the queue to be empty.

The task is unusually fast when the bounded buffer is of size 1 or 2. Perhaps this has to do with how the queue data structure behaves when there are a lot of queues and dequeues onto it when it is small.

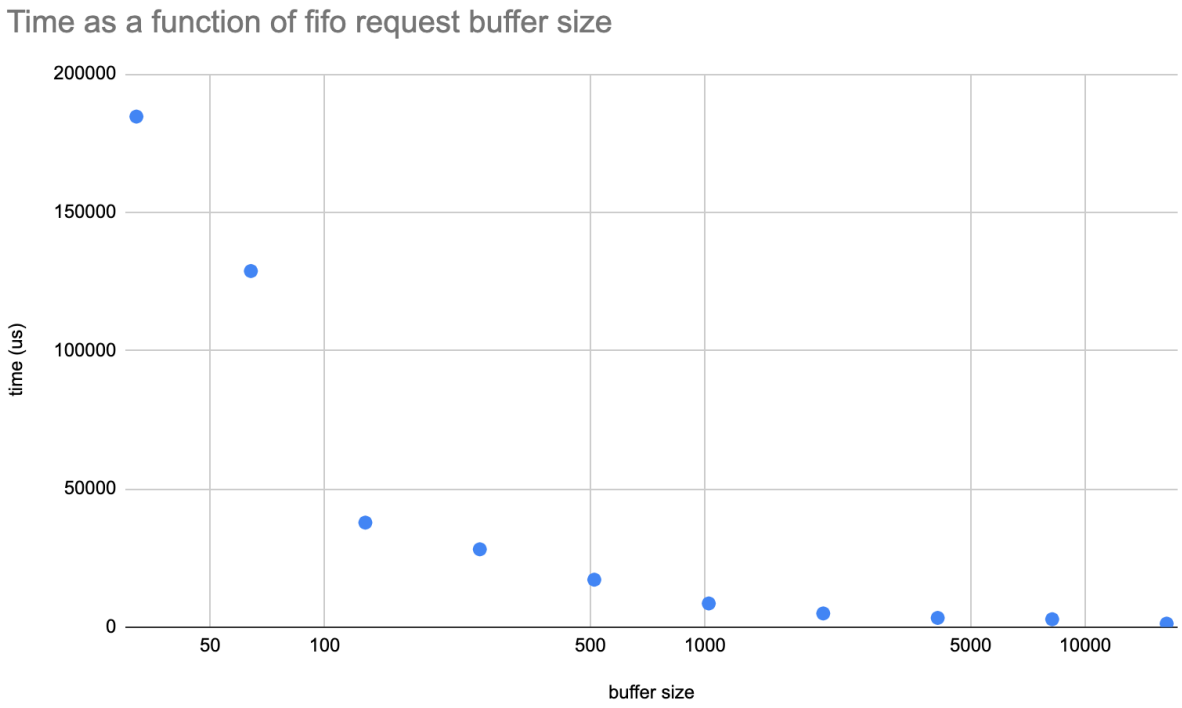
Raul Escobar
328003859

# Graphs for file requests:




No noticeable speed up with increased workers, but at large worker numbers there is a slow down perhaps due to too many context switches. Having too many workers would mean that each one has to wait more for other workers to give up their time on the cpu, not to mention the time it takes to join each worker thread and send a quit message for each channel opened to each worker. There is probably no speed up between 1-50 workers because they all have to write to the same file. Since it is thread safe, that means that they have to wait for one worker to finish writing for them to be able to write to it.

Raul Escobar
328003859




We can see again another inverse relationship (buffer in the sense for the fifo requests not the bounded buffer). As buffer size increases, the file thread has to push less file requests, and the workers each have to make less and less requests. This decrease in requests is what causes the speed up. Instead of requesting very small packets of data, the workers get them in larger chunks so there is less overhead. Bottoms out at about 16kB packet sizes.