Scalability Report on Stock Matching Server

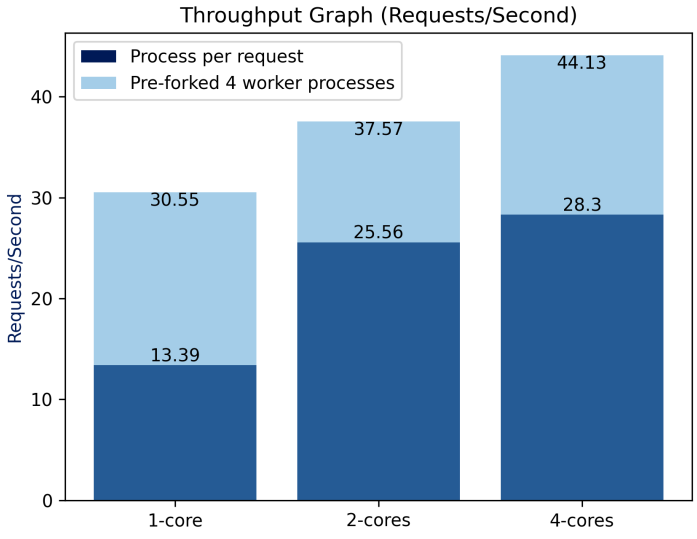
Xincheng Zhong
Rui Cao
April 5, 2023

Experiment Setup

We create 3 testing client programs to test both our functionality and scalability. The program “client.py” mainly tests that our server can generate appropriate responses according to different requests, including different error messages. The program “client2.py” is to test that our server can create correct number of accounts and positions under concurrency situations. A bash file is used to run 10 “client2.py” concurrently. The program “client3.py” mixes all requests together under concurrent environment, with the same method to run as ” client2.py”

Experiment 1:Process per request, and Pre-forked processes under different cores.

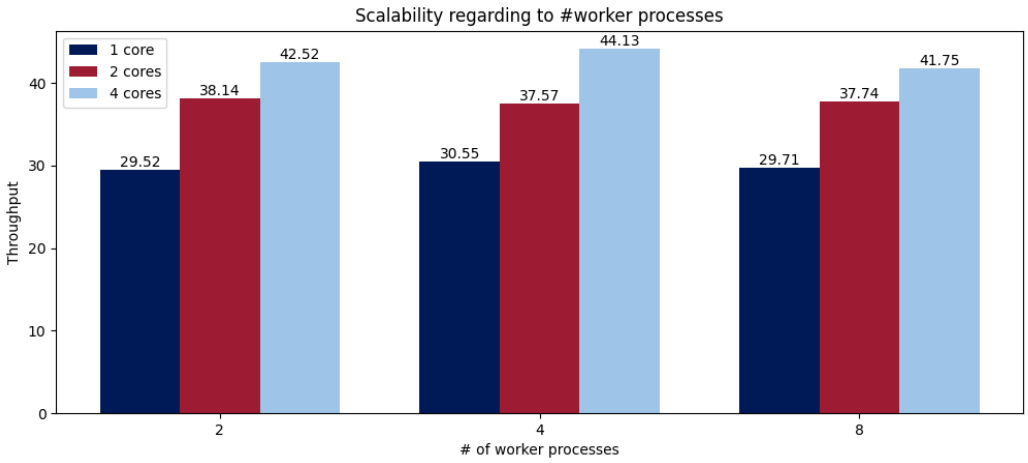
Firstly, we made our server asynchronous by fork a process every time there is a new request. However, we realize that it creates arbitrary number of processes if there are many connections, and the latency of fork() will largely degrade the performance of the server. We improved the performance largely by making a process pool with many pre-forked processes. The figure of throughputs measured by running 10 “client3.py” concurrently is shown below.



As shown in the above figure, the throughput of the server is largely improved by pre-creating some worker processes. And the multiple cores also helps to increase the throughput.

Experiment 2: Scalability under different number of pre-forked processes

We tested how many pre-forked worker processes is the best choice by measuring the throughput under different cores. The figure of the throughput of different worker processes is shown below.

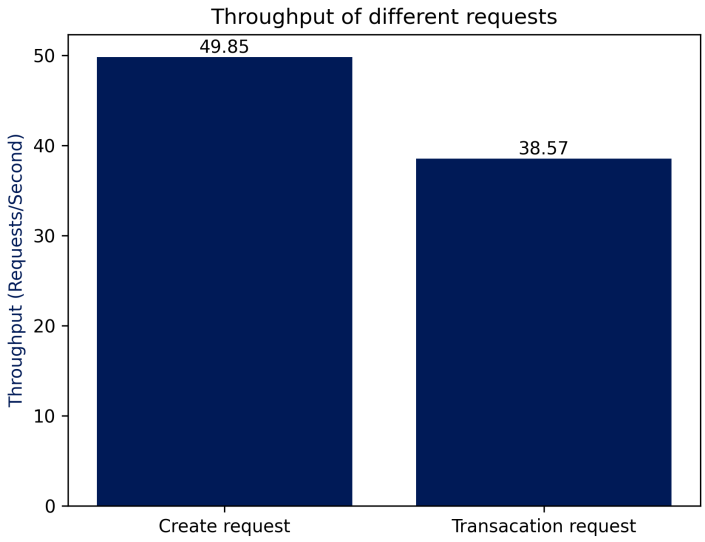


As shown in the above figure, it shows that more pre-forked processes didn't improve the performance. From our experiment, the best choice is to choose the process number similar to the number of cores. We infer that the context switching under one cpu core will create some overhead, thus the performance is influenced.

Experiment 3: Study on which type of request costs the most

We decide to study on which type of request is the slowest so that we can focus on improving that according to Amdahl's Law. We firstly run "client2.py" to test the throughput of creating accounts and positions and then run "client3.py"

without the “creat” request. Then we calculate the throughput on create request and transaction request separately. The results are shown below.



From the above figure, we can see that the slower request is the transaction request. Therefore, it is meaningful to improve the scalability on that type of request. For example, we can parallelize “open”, “query” and “cancel” because their order doesn’t matter. We can also have some thread to continuously run matching process, so we can reply faster to our client if there are “open” requests. However, the most promising change to improve our scalability is to get rid of the python multiprocessing locks, and instead make use of row level table locks.

Conclusion

In this homework we built a stock matching server and test its scalability under different situations. We also improve our program by pre-creating some worker processes and adjust its number according to experiment results. We also mentioned where our program can be improved to increase its scalability.