ECE590 ERSS HW4 Scalability Report

ts325, xl351

March 31, 2022

1.Introduction

In this assignment , we first implement our Exchange Matching system as a single thread server, which also only has one connection to the database. So the server can only handle one request each time. Later, we implement the multiple thread method on the server part, so our server will have different thread to handle different request we received from the client. We also implement a database connection for each request, so they can access to the data base simultaneously. To avoid chaos occur in our data base we also implement !!! and realize the atomic operation on the database. To test our concurrency strategy, we test our systems throughput and latency. To carry our the test on both functionality and scalability we implemented client part of our system as the testing infrastructure to test the server by sending and receiving requests and response.

This report mainly focus on the scalability of our server, and

To get the throughput the client-side infrastructure sends requests rapidly enough to saturate the server. We finally collected and analyzed the measurement result of our system.

2.Implementation:

a) Server side: For the server part, we use two strategy to increase the scalability:
   i. Create thread per request: this strategy is to create a single thread to handle the request every time after we receiving it. The responsible thread will exits after processing this request and send back a response. The overhead of the initial set up is lower, however, it will be larger for every single request it received later.
   ii. Pre-create a set of threads: this strategy is to create a specific threads during the first set up process. Every thread will continually receive and process incoming request. We use a while loop for every thread, so when it finished processing, it will continually to receive the next request rather than exit. So the overhead for each request will be lower.

b) Database side:

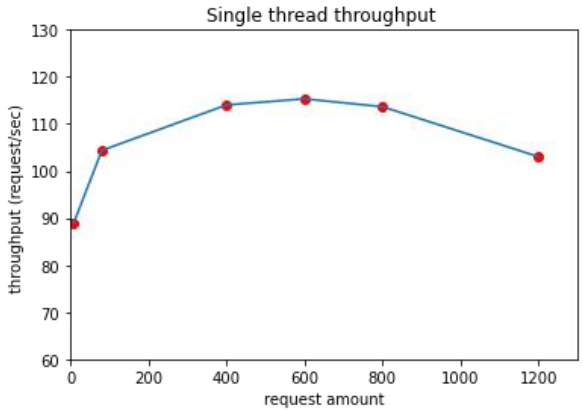
For the database side, in order to implement the concurrency reading and writing in database, we implement multiple connections to our data base to realize the concurrent behavior. However, as we need to isolation between each different transactions, we also focus on the atomic behavior of our database. We use the row lock every time we will update the value in it. And every time a transaction fails, our database will roll back to the last commit.

c) Client side: This infrastructure is what we mainly build to do our testing job. There are two different version of client part. The first one is to test the functionality of our system which is running sequentially with only one thread and one connection to the database on server. The client also only send several request sequentially. The other version of client part is to test the scalability of our server, so it will concurrently send requests to our server. In order to simulate the real working conditions for our server. Our client generate the random request and send to our server.

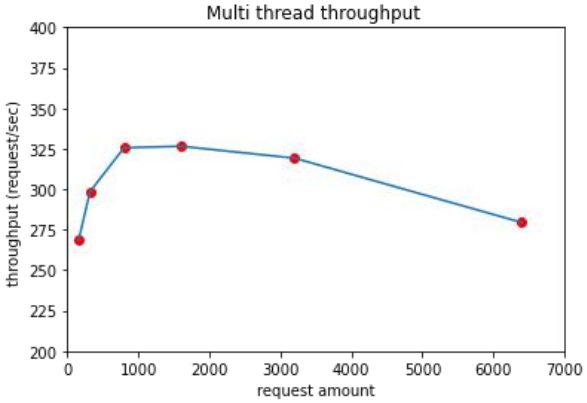
3.Result:

In this part, we mainly focus on the throughput of our server. In order to get the accurate throughput, we firstly increase the amount of request been sent to the client, and try to find the saturate status of our system. We then test the throughput under different strategy and measure the throughput with different number of cores under saturate status. The result are shown as follows:

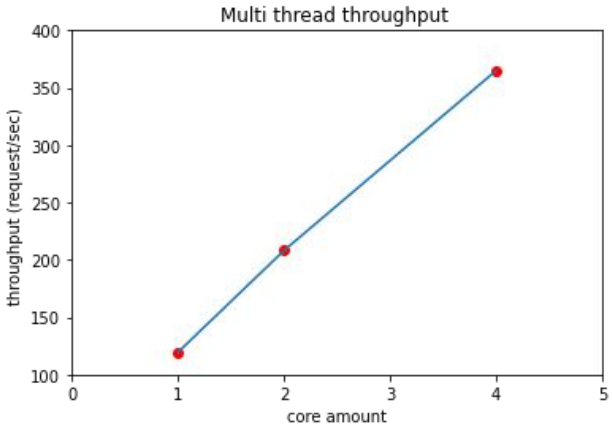
For the single thread we find the saturate status with a maximum throughput at 115.334827 request/sec



For the multiple thread strategy we use the create per request for the robust reason. And the result are shown as bellow: the multiple thread we find the saturate status with a maximum throughput at 326.531405 request/sec



By limit the amount of cores of our server we can get the amount as the graph:



In this experiment we use the create thread per request and strategy and by limit the thread number create on client to send request, we can also limit the process number on server side.

4.Conclusion:

From the result we can see that: By implementing the multiple threading strategy, we increase the final performance of our system with a larger throughput. And the throughput of our system is linear to the number of core on our server side.