ECE568 Homework 4 Scalability Report

Team Members:

Aohua Zhang (NetID: az147)

Jichen Zhang (NetID: jz367)

Logic of Implementation

Our exchange matching engine takes in clients' requests in XML form via socket, and parses the received XML requests with the library tinyxml2. Based on the parsed results, the server program would manipulate the database "EXCHANGE" containing four tables: "ACCOUNT", "SYMBOL", "ORDER", and "TRANSACTION", by complying with the rules for symbol exchange specified in the assignment instruction. Upon inserting/deleting/retrieving/deleting entries in the database tables or encountering errors, the server program would send corresponding responses in XML format back to the clients via socket. The content of the responses is generated by following the rules specified in the assignment instruction.

Multi-threading Techniques

Creating one thread for each client successfully connected to the server for handling all the requests from that client. Each thread would establish one connection with the database. Different threads would not interfere with each other as they are detached after being created.

Methods to Ensure Thread-Safety

1. Added "FOR UPDATE" at the end of each "SELECT" SQL statements (Except the "SELECT COUNT(*)" statements).
2. Used Mutex locks to lock all the groups of insertion/deletion operations which need to be executed atomically.

Scalability Analysis

By using the "taskset" command to specify the CPU cores on which the threads spawned by the server program should run, we conducted three experiments using 1, 2, and 4 cores respectively, in which we used 81 clients to communicate with the server concurrently, each client would first create some accounts and establishes some transactions, and then send 1000 query requests to the server via a loop. Each time when the server establishes the connection with a new client, the server would detach a new thread for processing the requests from that client. We recorded the start, end, and total processing time (difference between end and start) for running each client (thread). It

can be observed that the difference between the latest end time and earliest start time among all the clients (threads) is the total duration taken by the server to process all the requests from all the clients (threads). In our practical experiments, we observed that the start time for each client is almost the same (the difference between the latest and earliest start time does not exceed 2 seconds). Therefore, for the sake of simplicity, we retrieved the longest total processing time among all the threads as the total time for the server to process the requests from all the servers, and such total time is used to compare and analyze the scalability of our server program.

The results (responses received by the clients) of our three experiments are as follows: (As the output results are too long, here we only show the start, end, and total processing time of some of the clients for your reference)

1. Running the server program on 1 CPU core:

```
306  Total time used: 88s
307  Start time: 1648823592
308  End time: 1648823680
309  before return
310  Total time used: 88s
311  Start time: 1648823592
312  End time: 1648823680
313  before return
314  Total time used: 88s
315  Start time: 1648823592
316  End time: 1648823680
317  before return
318  Total time used: 88s
319  Start time: 1648823592
320  End time: 1648823680
321  before return
```

   It can be observed from the above results that the total processing time for the requests of all the clients is 88s.

2. Running the server program on 2 CPU cores:

```
306  Total time used: 70s
307  Start time: 1648823994
308  End time: 1648824064
309  before return
310  Total time used: 70s
311  Start time: 1648823994
312  End time: 1648824064
313  before return
314  Total time used: 70s
315  Start time: 1648823994
316  End time: 1648824064
317  before return
318  Total time used: 70s
319  Start time: 1648823994
320  End time: 1648824064
321  before return
```

It can be observed from the above results that the total processing time for the requests of all the clients is 70s.

3. Running the server program on 4 CPU cores:

```
306  Total time used: 44s
307  Start time: 1648850267
308  End time: 1648850311
309  before return
310  Total time used: 44s
311  Start time: 1648850267
312  End time: 1648850311
313  before return
314  Total time used: 44s
315  Start time: 1648850267
316  End time: 1648850311
317  before return
318  Total time used: 44s
319  Start time: 1648850267
320  End time: 1648850311
321  before return
```

It can be observed from the above results that the total processing time for the requests of all the clients is 61s.

Based on the above experimental results, we can conclude that:
The speed of the server for concurrently processing the requests from all the clients would increase if more CPU cores are used to run the server program (including all the threads spawned by the server).

When increasing the number of CPU cores from 1 to 2, the server's processing speed increased by (total processing time decreased by):

$$\frac{88-70}{88} \approx 20.45\%$$

When increasing the number of CPU cores from 2 to 4, the server's processing speed increased by (total processing time decreased by):

$$\frac{70-61}{70} = 12.86\%$$

When increasing the number of CPU cores from 1 to 4, the server's processing speed increased by (total processing time decreased by):

$$\frac{88-61}{88} \approx 30.68\%$$

From the above results, we can conclude that our server exhibits great scalability.