Homework4: Scalability Analysis

Team members: Jingzhi Zhao (jz422) Zhicheng Jiang (zj78)

1. Database Design

In this homework, we created a database called “exchange” in postgres, and designed 5 relations to handle the exchange services complying with the rules for symbol exchange specified in the assignment instruction.

account (account_id, balance)
position (symbol, amount, owner_id)
open (trans_id, account_id, symbol, limit_value, amount)
executed (trans_id, exect_id, amount, price, time, seller_id, buyer_id, symbol)
cancelled (trans_id, cancel_id, amount, time, account_id)

2. XML Parser

Our group chosed to use pugixml as the XML parser library.
Here is the link to its github repo: https://github.com/zeux/pugixml

3. Concurrency Techniques

We used a thread pool and lock_guard to deal with concurrency and avoid race condition.

```
void Server::startRun() {
  std::cout << "start Run server" << std::endl;
  std::threadpool executor{50};
  Database db("exchange", "postgres", "passw0rd");
```

4. Scalability Analysis

4.1 Methodology

We designed three experiments corresponding to the number of visible CPU cores of 1, 2, and 4, respectively. We used Linux taskset command to specify the CPU cores on which the threads spawned by the server program should run. Only the number of CPU cores visible to the server program is the variable for each experiment.

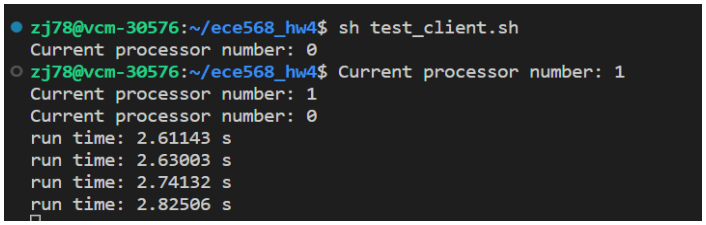
In each experiment, we used multiple threads to send 100, 200, and 400 requests to the server, and recorded the server's response time for different requests. We repeated this step 4 times to get the corresponding server's average response time.

After obtaining the response time of the server for different numbers of requests, we compare it with the number of CPU cores in the experiment and make a graph for further analysis.

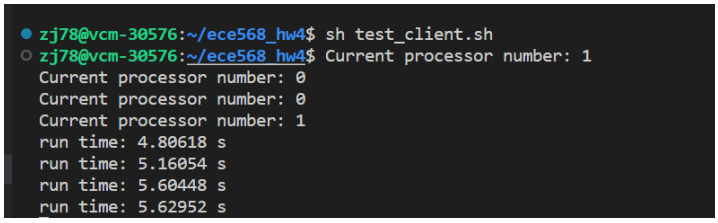
4.2 Results

4.2.1 One core

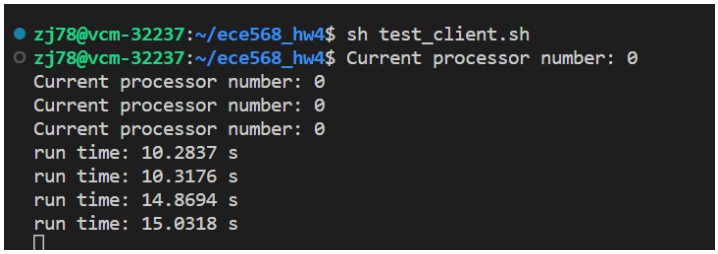
4 processes, each process has 25 clients, and each client sends 1 request, total of 100 requests



4 processes, each process has 50 clients, and each client sends 1 request, total of 200 requests



4 processes, each process has 100 clients, and each client sends 1request, total of 400 requests



4.2.2 Two cores

4 processes, each process has 25 clients, and each client sends 1 request, total of 100 requests

```
zj78@vcm-30576:~/ece568_hw4$ sh test_client.sh
Current processor number: 0
zj78@vcm-30576:~/ece568_hw4$ Current processor number: 1
Current processor number: 1
Current processor number: 0
run time: 2.28412 s
run time: 2.31624 s
run time: 2.33891 s
run time: 2.46432 s
```

4 processes, each process has 50 clients, and each client sends 1 request, total of 200 requests

```
zj78@vcm-30576:~/ece568_hw4$ sh test_client.sh
zj78@vcm-30576:~/ece568_hw4$ Current processor number: 1
Current processor number: 0
Current processor number: 1
Current processor number: 0
run time: 4.89955 s
run time: 5.34387 s
run time: 5.3767 s
run time: 5.44564 s
```

4 processes, each process has 100 clients, and each client sends 1request, total of 400 requests

```
zj78@vcm-32237:~/ece568_hw4$ sh test_client.sh
Current processor number: 0
zj78@vcm-32237:~/ece568_hw4$ Current processor number: 0
Current processor number: 0
Current processor number: 0
run time: 6.51359 s
run time: 14.8513 s
run time: 14.9663 s
run time: 14.9666 s
```

4.2.3 Four cores

4 processes, each process has 25 clients, and each client sends 1 request, total of 100 requests

```
zj78@vcm-30576:~/ece568_hw4$ sh test_client.sh
zj78@vcm-30576:~/ece568_hw4$ Current processor number: 0
Current processor number: 1
Current processor number: 1
Current processor number: 1
run time: 1.92826 s
run time: 2.0386 s
run time: 2.05845 s
run time: 2.62707 s
```

4 processes, each process has 50 clients, and each client sends 1 request, total of 200 requests

```
zj78@vcm-30576:~/ece568_hw4$ sh test_client.sh
Current processor number: 1
zj78@vcm-30576:~/ece568_hw4$ Current processor number: 1
Current processor number: 1
Current processor number: 1
run time: 2.20799 s
run time: 4.54022 s
run time: 5.16944 s
run time: 5.20411 s
```

4 processes, each process has 100 clients, and each client sends 1request, total of 400 requests

```
zj78@vcm-30576:~/ece568_hw4$ sh test_client.sh
Current processor number: 1
Current processor number: 0
zj78@vcm-30576:~/ece568_hw4$ Current processor number: 1
Current processor number: 0
run time: 10.0013 s
run time: 10.2082 s
run time: 10.3856 s
run time: 10.3892 s
```

The average response time of the server is shown below in second.

	100 requests	200 requests	400 requests
1 core	2.70 s	5.30 s	12.63 s
2 cores	2.35 s	5.27 s	12.82 s
4 cores	2.16 s	4.28 s	10.25 s

4.3 Analysis

From the results, we can conclude the trend 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, which means the throughput of the server would increase if more CPU cores are available.