# ECE568 HW4 Scalability Analysis

**Congjia Yu| cy146**
**Yingxu Wang | yw473**

## 1. Introduction

For this project, we write a Stock Exchange Matching Server. In this file we talk about the scalability of our server. We use different cores (1, 2 and 4) in our server to test the processing speed.

## 2. Scalability Testing Method

In our scalability testing, in the client side, we use multiple processing (because in Python multi-threading is not working to use multi-cores) to mimic the situation that multiple different users send requests to our server. For each process, we request to create different accounts. It avoids the error: create same account. And it ensures that each time server has same execution process for each request. Before sending request, we will delete and re-create all the tables in databases. It also avoids the possibility that server has different execution process.

We test different number of cores using in server side: 1, 2 and 4 using the command:
taskset -c 0 python3 ~/ece568/hw4/server.py
taskset -c 0,1 python3 ~/ece568/hw4/server.py
taskset -c 0-3 python3 ~/ece568/hw4/server.py
to allocate different cores to this program each time.

In client side, we will create 10 processes. Each process will create 2000 accounts and different processes will send at the same time. We measure the time it costs to send and receive all the requests (latency) and the number of processes that server process, i.e., 2000 * 10 / latency (throughput).

We run each experiment 2 times and get the average.

## 3. Result and Analysis

| | 1st | | | | 2nd | | | average | |
|---|---|---|---|---|---|---|---|---|---|
| | cores | time(ms) | query/ms | cores | time(ms) | query/ms | time(ms) | query/ms |
| | 1 | 37365 | 0.53526027 | 1 | 40673 | 0.4917267 | 39019 | 0.51349348 |
| | 2 | 33221 | 0.60202884 | 2 | 30567 | 0.65430039 | 31894 | 0.62816461 |
| | 4 | 29618 | 0.67526504 | 4 | 29207 | 0.68476735 | 29412.5 | 0.6800162 |

From the result we can see that with more cores, the time spent to process multiple requests decreases obviously from 39019ms to 29412.5ms with the change from 1 core to 4 core utilizations in virtual machine. (**Latency**) At the same time, I calculate the query per millisecond also increases from 0.51349 to 0.6800 queries per milliseconds. (**Throughput**) Converting it from query/milliseconds to query/seconds, the queries/seconds improves a lot! From 513 queries/seconds to 680queries/seconds.

## 4. Conclusion

From the above result, we can observe that with more cores, the latency will decrease, and the throughput will increase. In other words, with more cores, the server can process more clients'

request in each unit. Meanwhile, clients can receive response from the server side more quickly. Our server becomes more powerful when there are more cores assigned.