ECE568 - Scalability: Exchange Matching
Engineering Robust Server Software Homework 4

Shuhao Zhang (sz279)
Shravan Mysore Seetharam (sm952)

Introduction

This assignment involves writing an exchange matching server – a software that will match buy and sell orders for stock/commodities market. The server code is developed in C++. This document presents our observations seen as part of the scalability testing.

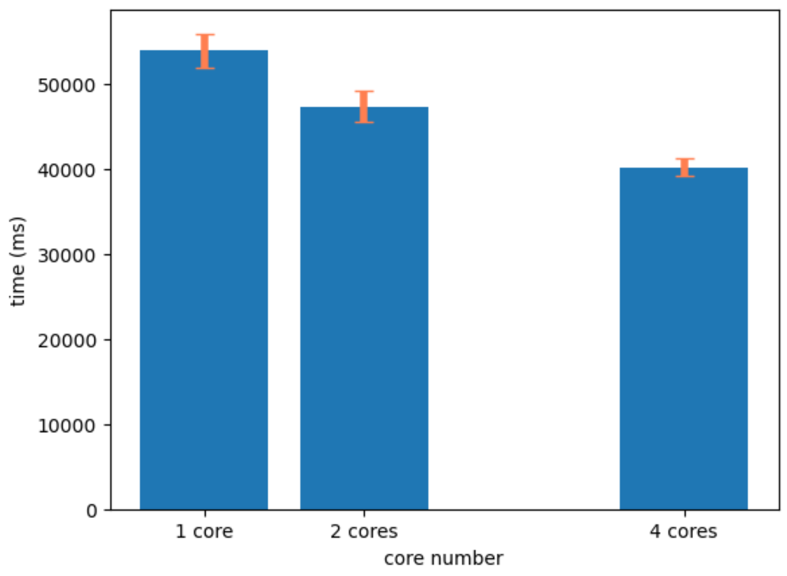
Scalability test - Background

For our test structure, we have two pieces of codes run at client side, one is ***throughput.py*** and the other one is ***scalability.py.*** We ran the client code and server code on different virtual machines.

1. The ***throughout.py*** code is developed in Python and is designed to deploy 10 processes with each spawning 3000 accounts all hitting the server concurrently. This code also captures the latency (total time taken in ms) and the throughput (queries hit per ms) of the server. We used this code to test the throughput of our server. The results are as bellowing:

Cores	Throughput (query/ms)
1	0.051302835507718
2	0.07146123942373657
4	0.1267941370391033

2. We have another code called ***scalability.py*** which run in one process and sent server different kinds of requests in iterations. In each iteration, we will send server ~30 different requests and iterated for 50 times. In the iteration, we will not wait until we receive the response after we sent the request. After we finish all the iterations of sending, we will send one single request and wait until we receive the response. In that way, we can make sure the server has finished all the requests before we measured time. Since sending request is faster than processing the request, we can utilize the concurrency feature of our server. We run this test 10 times for 1, 2, 4 cores and calculate the average time. Here are the results, we have added an error bar for standard deviation:



3. Also, the server code is launched using shell command, “taskset -c” to run on 1, 2, 4 cores.

Result Analysis

It’s evident from above that the latency decreases, and the throughput increases as we increased number of cores from 1 to 4. This observation is expected as the server code runs in parallel on multiple cores, the client receives the responses quicker, hence resulting in better throughput and reduced latency. When we increased core number from 1 core to 4 cores, the total time costed didn’t decrease hugely may because that network transporting cost some time.