ECE 568 homework4 Scalability Analysis
NetID: zf70, hb174

1. Introduction

Our system is based on C++ 11 and uses Postgres as our database. Our system supports creating accounts, adding symbols and making transactions, queries, and canceling their opened transactions. The system will match the opened trades and find a suitable price for the user.

After implementing the transaction system, we need to test the scalability of our server.

2. Methods

We designed that we should test our system for 1,2,4 core with single and multiple client requests. To be more specific, we plan to loop 10 times and get the average time and throughput for each test case. For each test case, we make the sum of the request (client number * requests per client) the same.

3. Result and Analysis

We set that the sum of the request is 1000, which means 1000 creating account requests.

Table 3.1 The result of the scalability test for 1000 requests

ThreadPool Size	Cores	Client Number	requests per Client	Average Total Time (ms)	Average Time per Client (ms)	Throughput (request per ms)
1	1	1	1000	5617	5617	0.178
1	2	1	1000	5566	5566	0.1796
1	4	1	1000	5408	5408	0.1849
1	1	5	200	5969	1193.8	0.1675
1	2	5	200	5832	1166	0.1714
1	4	5	200	5664	1132	0.1765

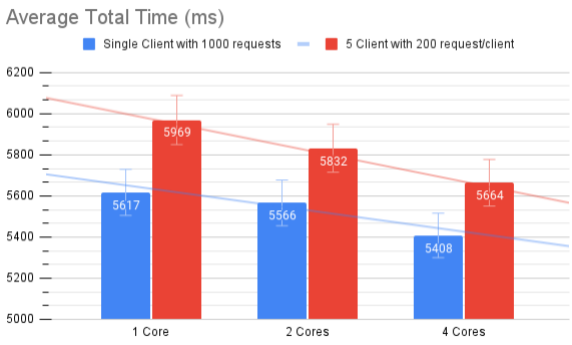

Figure 3.1 Average Total time of all requests

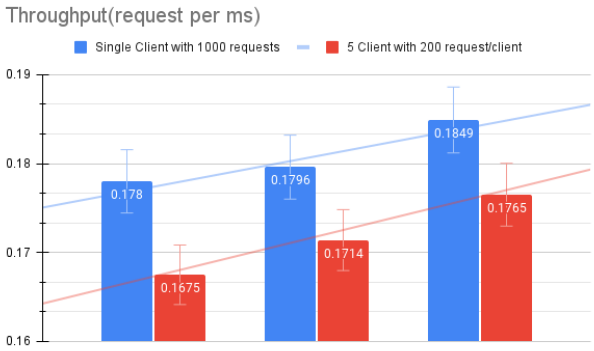

Figure 3.2 Throughput

From Figure 3.1 we see that with the increase of core numbers, the average total time of all requests does reduce. As each time the number of cores doubled, the average time will reduce about 100 ms, which we can calculate that each time the time will be saved about 1.5~2.5%,

which is not a significant decrease in total time used. And we also set threadpoolsize to be 5, here is the result:

Table 3.2 The result of the scalability test for 1000 requests and ThreadPoolSize = 5

ThreadPool Size	Cores	Client Number	requests per Client	Average Total Time (ms)	Average Time per Client (ms)	Throughput(request per ms)
5	1	1	1000	5983	5983	0.1671
5	2	1	1000	6013	6013	0.1663
5	4	1	1000	5937	5937	0.1684
5	1	5	200	4838	967.6	0.2066
5	2	5	200	4695	939	0.2129
5	4	5	200	4643	928.6	0.2153

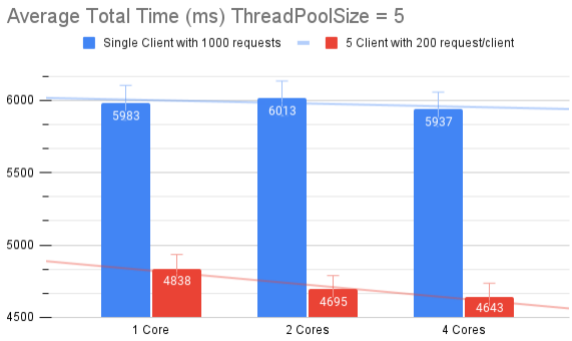

Figure 3.3 Average Total time of all requests

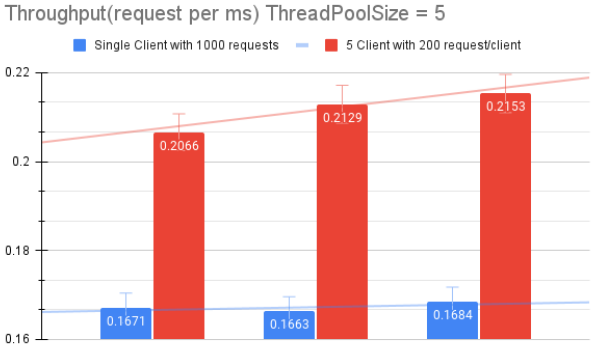

Figure 3.4 Throughput

From Table 3.2 and figures above, we can see that increasing thread pool size will improve multiple client request speed very much, but it will not influence single client very much. This is consistent with the basic principle of multithreading.

To be more reliable, we perform another test with 10K requests:

Table 3.3 The result of the scalability test for 10K requests

ThreadPool Size	Cores	Client Number	requests per Client	Average Total Time (ms)	Average Time per Client (ms)	Throughput (request per ms)
1	1	1	10000	62672	62672	0.1595
1	2	1	10000	58174	58174	0.1718
1	4	1	10000	61963	61963	0.1613
1	1	5	2000	62793	12558.6	0.1592
1	2	5	2000	60344	12068.8	0.1657
1	4	5	2000	61379	12275.8	0.1629

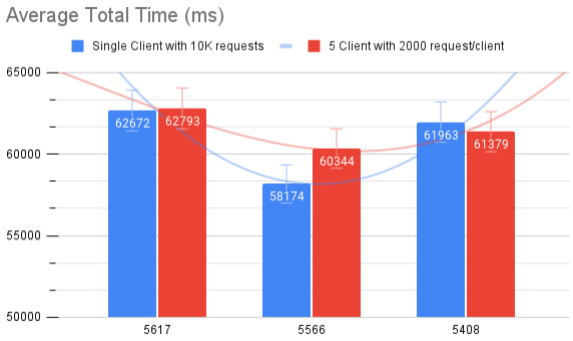

Figure 3.5 Average Total time of all requests

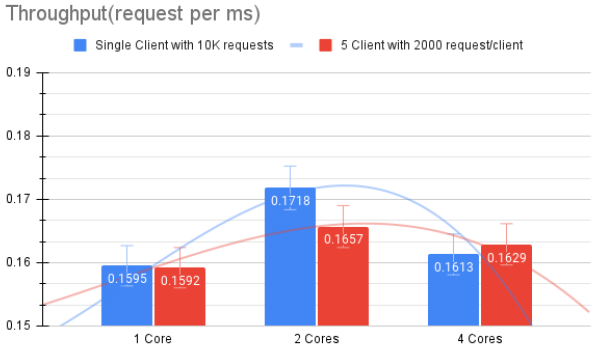

Figure 3.6 Throughput

From the above experiment, we can know that different number of cores do influence the performance of our server. The less total of request will consume less more time.

When it comes to the same number requests, the more core will improve the speed of processing according to Fig 3.1 and Fig 3.2, muti-Thread will also improve the speed when comparing to single client according to Fig 3.3 and Fig 3.4.

However, 10K request test case did not act as a linear change, we guess that the transformation from raw date to Node will impact the time. And, for the large number requests, the context switching between cores will also consume more time than less requests.