ECE 568 - HW4

Scalability: Exchange Matching

Team Members: Afsana Chowdhury (ahc35), Zezhong Zhang (zz258)
Date: 04/01/2022

Introduction:

In this report, we will discuss the scalability of the stock exchange matching server we developed.

Test Setup:

The MultiClient program can spawn a specific number of threads given by the user, open a socket to connect to and send requests to the server concurrently. This test setup allows us to evaluate the multi-threaded operations of the server as the server processes each request in a separate thread. The MultiClient program also calculates the latency and throughput of the server.

In order to test the scalability of the server program, we ran the server with a different number of cores that is visible to the program. The Duke VCM we used had 4 cores available, and we tested with 1, 2 , 3 and 4 cores and compared the results.

We ran two different tests - one with 10,000 client requests and another with 20,000 - for 1 to 4 cores so there were (8) different tests in total. We ran each test for 3 times and calculated the average results which are being shown in Figure 1 and 2.

Results:

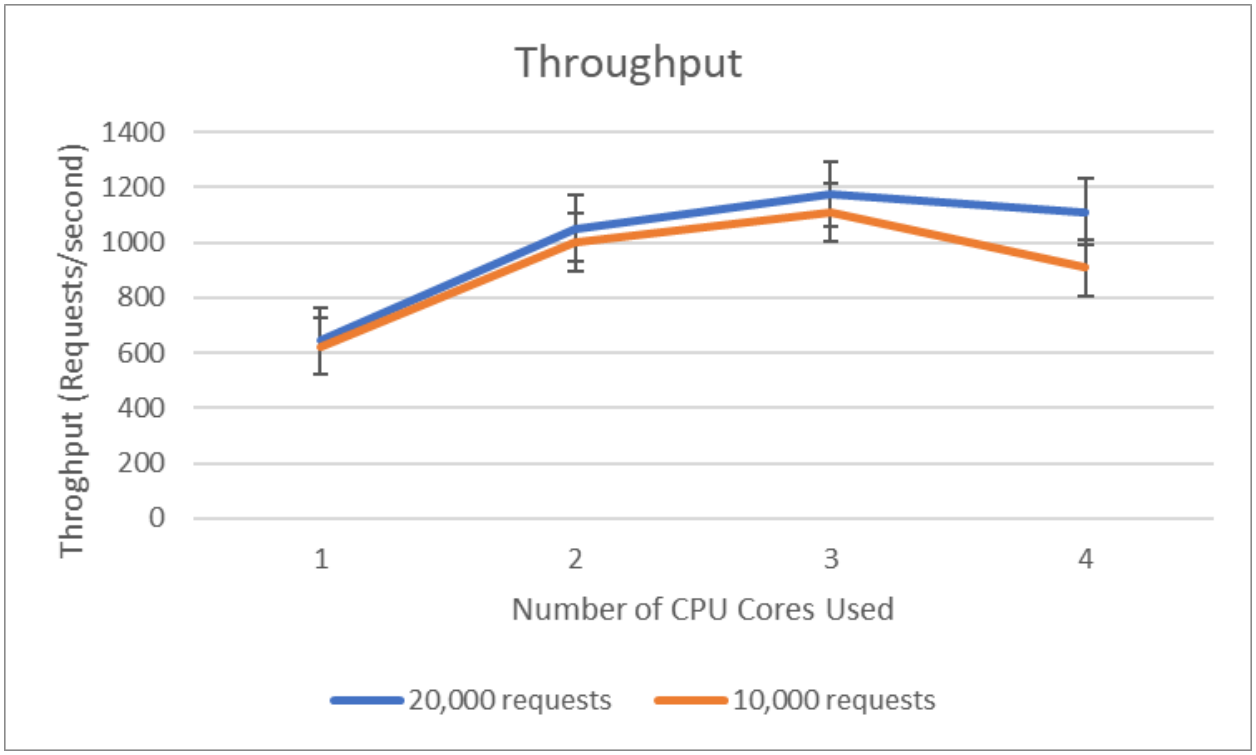

Figure 1: Throughput vs Number of CPU Cores

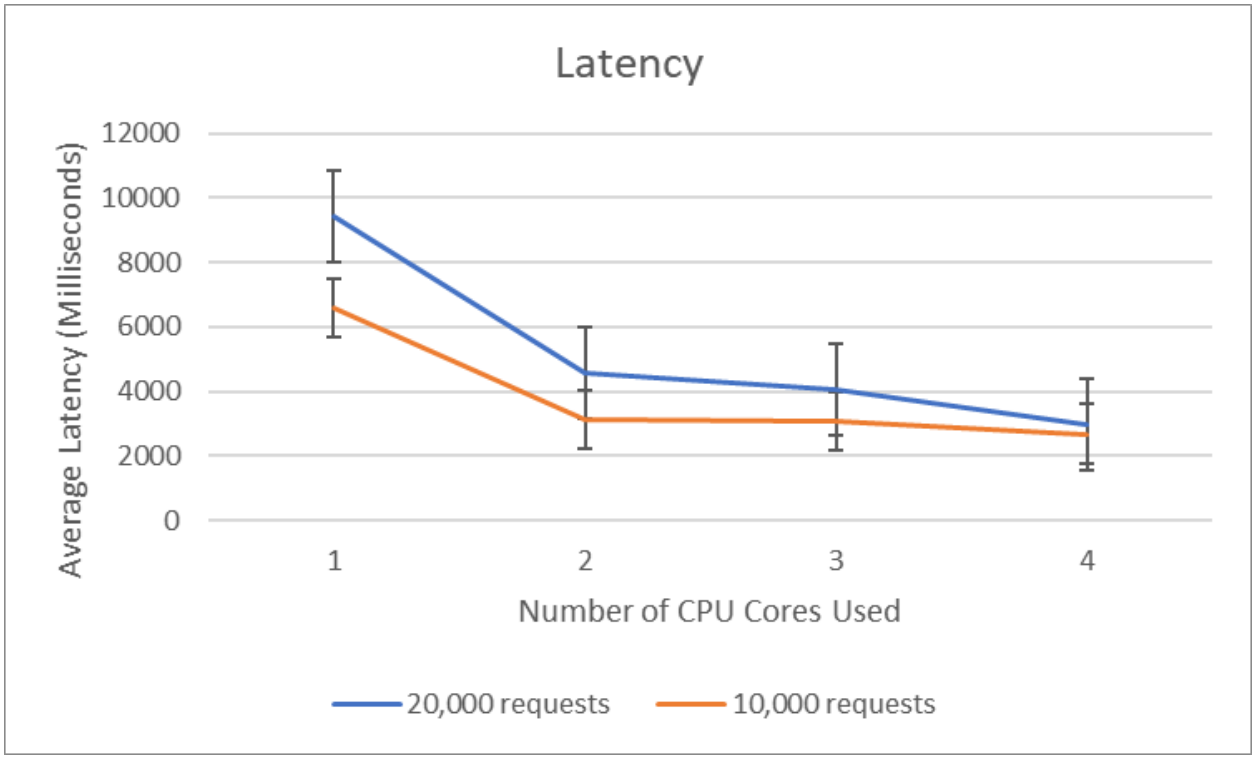

Figure 2: Latency vs Number of CPU Cores

Discussion:

i) Throughput:

According to Figure 1, the throughput seems to be directly related to the number of cores used by the server application. Since more available cores means more opportunities for parallelism, this relationship makes sense. One thing to notice is how (3) and (4) cores have similar throughput which shows how the performance can reach a plateau because of the overhead of multiple threads (e.g. context switching).

Also, the plotted line doesn't change much for 10,000 and 20,000 client requests, and that supports the fact that no matter how many requests are in the queue, the server's limit to process a certain number of requests per second doesn't change.

ii) Latency:

According to Figure 2, the latency seems to be inversely related to the number of cores used by the server application. Since more available cores means more opportunities for parallelism, this relationship makes sense. One thing to notice is how (3) and (4) cores have similar throughput which shows how the performance can reach a plateau because of the overhead of multiple threads (e.g. context switching).

Also, the difference between 10,000 threads and 20,000 threads becomes more significant for lower number of cores. A possible reason can be - when there is only one core which needs to schedule the threads to create the illusion of concurrency, the requests waiting in the queue to be processed have an effect on the scheduling and thus the latency increases for higher number of client requests.

Conclusion:

From the test results, it is pretty evident that the number of cores available to the server application has a big effect on the throughput and latency of the program. However, because of the overhead that comes with a multi-threaded application, both metrics may reach a plateau. So, in order to find the most suitable number of cores for any specific application, we need to test and compare the results.