ECE 568 HW 4 Scalability Report

Team member: Yiyang Li (yl883), Yue Zeng (yz723)

1. Introduction

In this report, we present the scalability analysis of a custom-built exchange matching engine designed to efficiently match buy and sell orders in a stock/commodities market. The matching engine is implemented in C++ and leverages a thread pool to manage multiple client connections concurrently. To ensure data integrity and safety, the system incorporates robust database handling mechanisms for reading, writing, and querying operations. This study aims to evaluate the performance of the matching engine under varying CPU core counts.

2. Methodology

Details of the methodology used to analyze the scalability under varying CPU core counts are explained as follows. The testing environment consists of an Ubuntu 20 virtual machine with 8 GB of base memory and four processors. To simulate varying levels of client connections, we create three scenarios with 1000, 500, 100 and 10 clients connecting to the server.

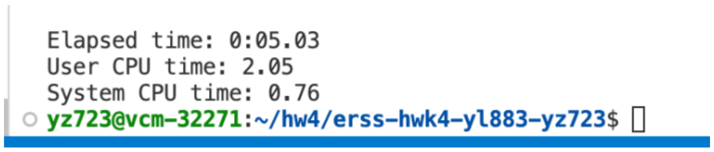
For each of these scenarios, we run the server on one core, two cores, and four cores to investigate the impact of CPU core count on the server's performance. To collect relevant data, we measure three time attributes: Elapsed time, User CPU time, and System CPU time. These metrics are printed on the terminal for further analysis.

Elapsed time represents the total duration of the test, while User CPU time refers to the time spent executing user-level code. System CPU time denotes the time spent on kernel-level operations. By comparing these three attributes under different client connection scenarios and CPU core counts, we aim to gain a comprehensive understanding of the scalability of our exchange matching engine.

The experimental methodology includes running each scenario multiple times to obtain reliable and accurate results. The data gathered will be used to create graphs that showcase the server's scalability and performance. Additionally, error bars will be plotted to represent the variability in the results, ensuring a thorough and rigorous analysis of the matching engine's scalability.

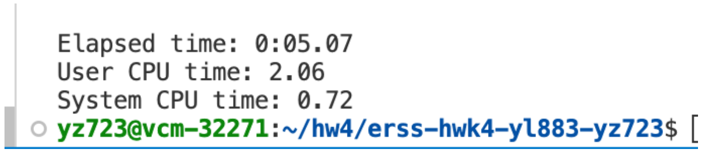
3. Results

Figure 1, 2 and 3 are three screenshots taken from the terminal as examples showing the results of our scalability tests, which support the analysis and conclusions.



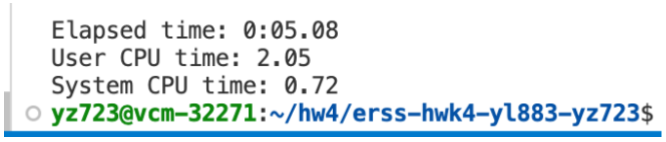
```
Elapsed time: 0:05.03
User CPU time: 2.05
System CPU time: 0.76
yz723@vcm-32271:~/hw4/erss-hwk4-yl883-yz723$
```

Figure 1 Run Server with 1 Cores and 100 Clients



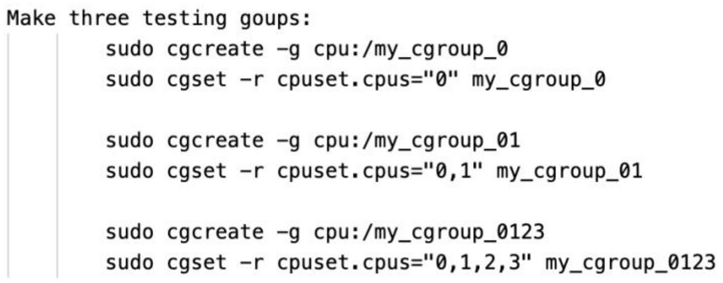
```
Elapsed time: 0:05.07
User CPU time: 2.06
System CPU time: 0.72
yz723@vcm-32271:~/hw4/erss-hwk4-yl883-yz723$
```

Figure 2 Run Server with 2 Cores and 100 Clients



```
Elapsed time: 0:05.08
User CPU time: 2.05
System CPU time: 0.72
yz723@vcm-32271:~/hw4/erss-hwk4-yl883-yz723$
```

Figure 3 Run Server with 4 Cores and 100 Clients

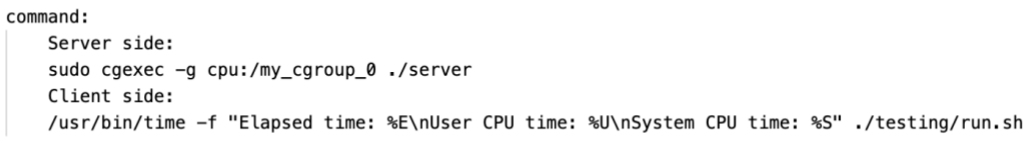


```
Make three testing goups:
        sudo cgcreate -g cpu:/my_cgroup_0
        sudo cgset -r cpuset.cpus="0" my_cgroup_0

        sudo cgcreate -g cpu:/my_cgroup_01
        sudo cgset -r cpuset.cpus="0,1" my_cgroup_01

        sudo cgcreate -g cpu:/my_cgroup_0123
        sudo cgset -r cpuset.cpus="0,1,2,3" my_cgroup_0123
```

Figure 4 Commands to Make Testing Groups



```
command:
    Server side:
    sudo cgexec -g cpu:/my_cgroup_0 ./server
    Client side:
    /usr/bin/time -f "Elapsed time: %E\nUser CPU time: %U\nSystem CPU time: %S" ./testing/run.sh
```

Figure 5 Commands to Run Testing Group and Start Clients

In the Figure 4, my_cgroup_0 indicates we run the server with core No.0; my_cgroup_01 indicates we run the server with core No.0 and No.1; my_cgroup_0123 indicates we run the server with core No.0, No.1, No.2, and No.3.
In the

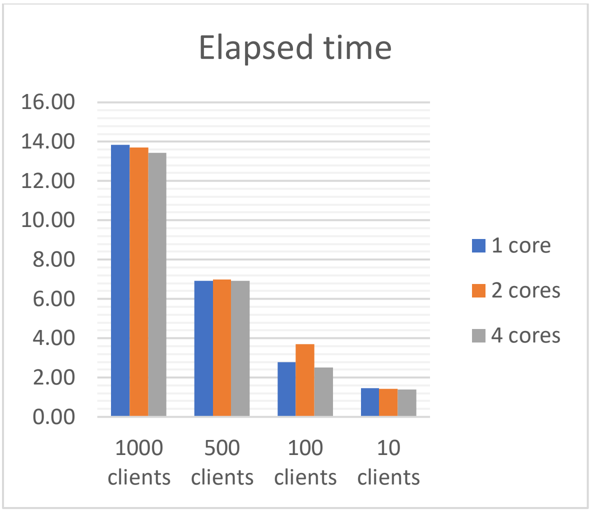


Figure 6 Average Elapsed Time

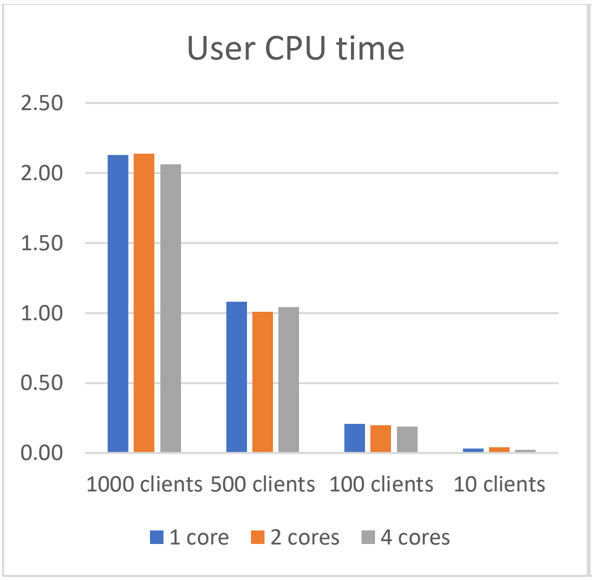


Figure 7 Average User CPU Time

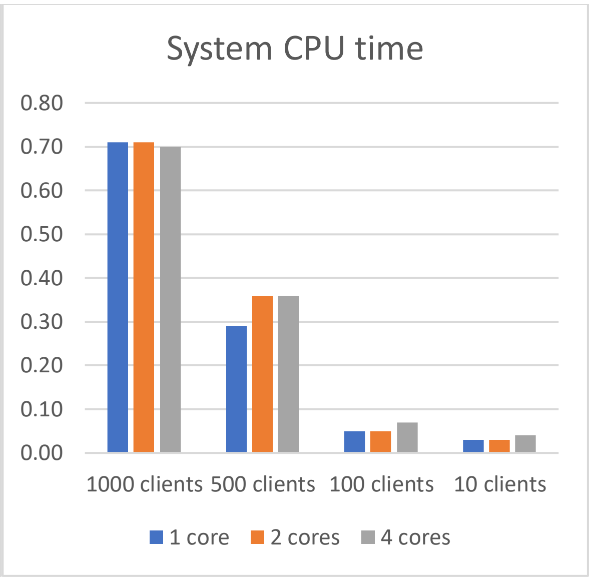


Figure 8 Average System CPU Time

4. Analysis

When looking at the data, it is noticeable that the elapsed time does not decrease significantly as the number of cores used to run the server increases. Let's analyze each test scenario and provide possible reasons for this behavior.

4.1 Test connecting 1000 clients

Increasing the server cores from 1 to 2 and then to 4 resulted in only a marginal decrease in the elapsed time. This might indicate that there is a bottleneck in the server's implementation, such as contention for shared resources like memory or network bandwidth, that is preventing the server from taking full advantage of the increased number of cores.

4.2 Test connecting 500 clients

The elapsed time remains almost the same for all core configurations. This suggests that the server is not able to fully utilize the additional cores, possibly due to the same bottlenecks mentioned earlier.

4.3 Test connecting 100 clients

The elapsed time slightly increases when moving from 1 core to 2 cores, and then it decreases when using 4 cores. This might be due to the server's performance being influenced by other factors such as the clients' behavior, the network conditions, or the CPU's scheduler.

4.4 Test connecting 10 clients

In this case, the elapsed time increases as the number of cores increases. This could be attributed to the overhead of managing multiple cores, which becomes more significant with a smaller number of clients. It may also indicate that the server's workload is not large enough to fully utilize the additional cores.

In conclusion, the server's performance does not improve as much as expected when adding more cores. The reasons for this could be bottlenecks in the server's implementation, contention for shared resources, or overhead of managing multiple cores. To improve the server's scalability, it would be important to investigate these potential issues further and optimize the server's design to better utilize the available CPU resources.

5. Conclusion

In conclusion, the scalability test results indicate that the server's performance does not improve significantly when increasing the number of CPU cores. This may be due to bottlenecks in the server's implementation, contention for shared resources like memory or network bandwidth, or the overhead of managing multiple cores. To enhance the server's scalability and make the most of the available CPU resources, it is essential to investigate and address these potential issues. By optimizing the server's design and addressing the factors limiting its performance, we can achieve better scalability and more efficiently handle a larger number of clients, ultimately delivering a more robust and responsive system.