ECE 568 Engineering Robust Server Software

Homework 4: Exchange Matching Server
Report

Name: Wenjun Zeng, Fangtin Ma
NetID: wz165, mf128

1 Project Summary

In this project, a multi-threaded exchange matching server was developed in Java. This server can interact with multiple clients through TCP socket communication. It can also parse two type of requests in XML form, "create" and "transaction", and connect to a PostgreSQL database using Java Database Connectivity (JDBC) API.

The following part of the report includes an overview of the implementation, results from test cases, as well as an analysis of the results.

2 Implementation

2.1 Client/Server Model

The communication between the client and the server is established based on Java's TCP API. The server side assigns a thread to each client, and each thread implements Java's Runnable interface to parse the XML request from the client and interacts with the database by implementing.

2.2 XML Parser and Generator

We use Java's DOM API to parse the client's XML request. The DOM is able to load the complete content of the XML document and create its complete hierarchical tree in memory. After parsing the XML request, we also use DOM to create and write the resulting XML to the client's socket.

2.3 Database

We use PostgreSQL as our server's database to store information from our clients and use the JDBC API to process transactions in a thread-safe manner.

3 Performance Analysis

We deployed the server on a 2-core 2GB Ubuntu 18 server and a 4-core 8GB Ubuntu 20.04 server respectively, and studied its performance by measuring the latency and throughput.

3.1 Create Requests

The throughput (requests per ms) of Create requests measured on the two VMs is shown in Fig. 1.

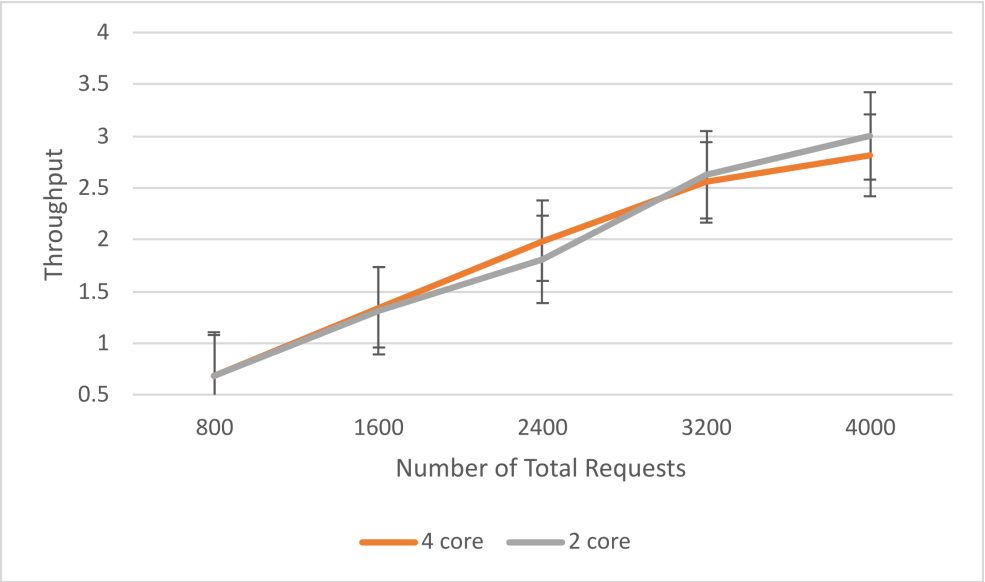

Figure 1: The throughput versus the number of Create requests

The latency (ms per 200 requests) of Create requests measured on the two VMs is given in Fig. 2.

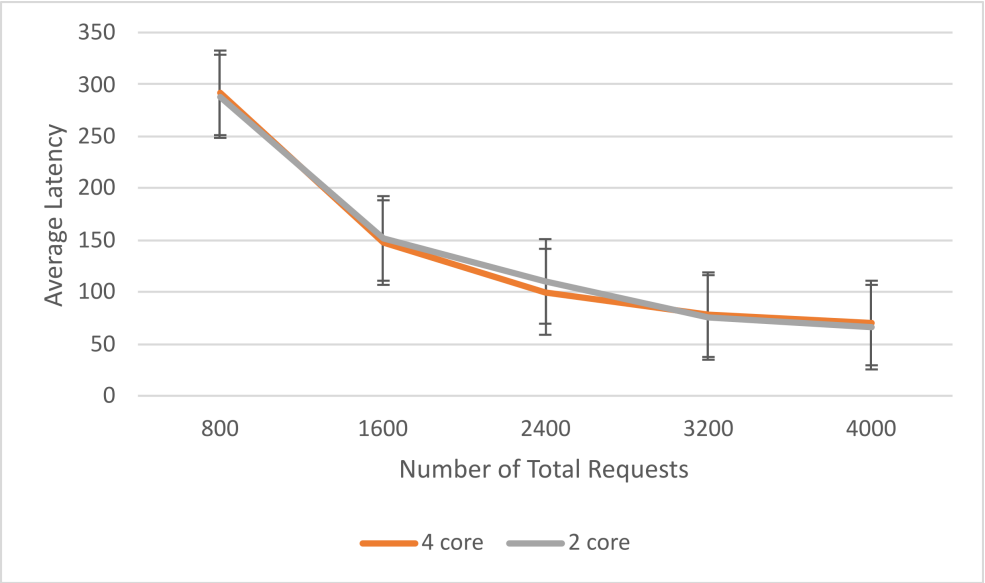

Figure 2: The latency versus the number of Create requests

As can be seen from Fig. 1 and 2, the throughput increase as the number of Create requests increases while the latency decreases as the number of Create requests increases.

The reason is that most of our test requests are duplicated. Although the create requests are write operations and can only be executed in serial to avoid conflicts in the database, more write operations actually won't be executed as the number of requests increases because, for example, an account has already been created by a previous requests will not be allowed to be created again. Therefore, the subsequent requests only need to do read operations (to check if an account has already been created) in parallel, which increases the throughput and reduces the latency.

3.2 Transaction Requests

The throughput (requests per ms) of Transaction requests measured on the two VMs is shown in Fig. 3.

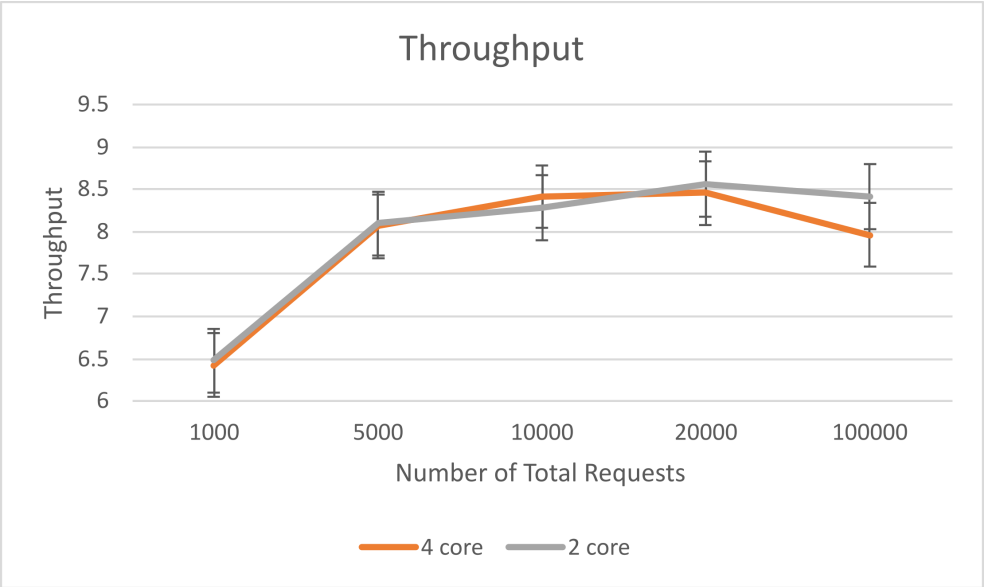

Figure 3: The throughput versus the number of Transaction requests

The latency (ms per 200 requests) of Transaction requests measured on the two VMs is given in Fig. 4.

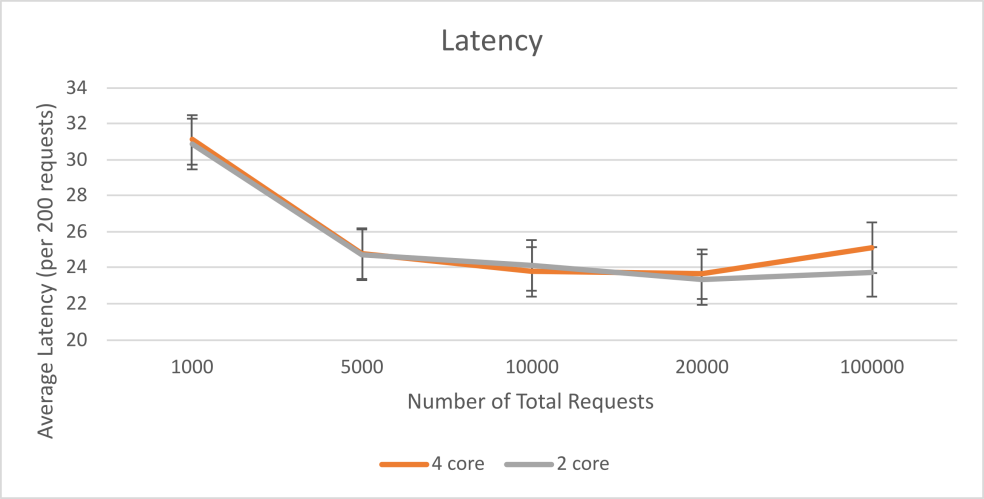

Figure 4: The latency versus the number of Transaction requests

As can be seen from Fig. 3 and 4, the throughput increase as the number of Create requests increases while the latency decreases as the number of Transaction requests increases.

The reason is similar to the case of Create requests. As the number of requests increases, more duplicated write operations won't be executed, and the rest of the read operations can be executed in parallel, thus increasing the throughput and decreasing the latency.

4 Results and Analysis

As can be seen from the above results, the measured throughput and latency measurements on the 2-core machine are almost identical to those on the 4-core machine, indicating that our server is less dependent on the computing power of the device and has relatively strong scalability.