ECE568 Scalability: Exchange Matching Spring 2022

Description:

In the assignment, we first build the connection between Server and Client, the Server and database(PostgreSQL) using JDBC, then we add the XML parsing part in which we use DOM Parser to parse the XML file received from the client to multiple classes. After that, we handle these orders according to the result of parsing: checking their validity, adding them to the database, matching orders, updating data in databases according to the result of matching, and the cancel orders. Finally, we returned the result from the database, generate XML files, flush them back to the client and display them.

Functional Testing:

To test our Stock Exchange Matching System, we create several *test.xml and imitate to test our system, in which we have tested the following functions:

[1] Adding symbols, accounts, and Positions:

If the symbol of positions does not exist, we create them.

If the account of a position does not exist, we inform the errors.

If the position is valid, we add them to databases.

[2] Adding orders and updating orders:

If the order is valid, we add them to databases and find possible orders to match. Otherwise, we inform errors.

If the order is matched, we updated matched orders, accounts, and positions.

[3] Query transactions and cancel transactions:

If the transactionid is not belong to the account, return error line

If the query transaction is valid, we return the specific information of the transactions, including open/closed/executed transactions. Else, we inform the errors.

If the cancel transaction is valid, we cancel the open transaction and remain the executed transactions. We also return the specific information of the transactions, including the open/closed transactions and executed transactions.

Scalability Testing:

Since PostgreSQL could only accept a maximum of 100 connections, we define a fixed thread pool whose size is 100. Other requests have to wait in the queue for a free thread. To measure the performance of our Stock Exchange System, we run our system under multiple conditions: the different amount of requests and the different number of used cores. Then we calculate the time and draw the following graph:

From the Scalability graph above we can find that the latency for responding each request increases as the number of requests increases. It's mainly because we only allowed 100 threads running concurrently, and most of the requests have to wait for free threads.

Besides, we can also find that the latency of 4 core VM is less than 2 core or 1 cpre because with more processors, we are able to run more threads in parallel. But the latency in for 2 core VM is not 2 times more than 4 core because one thread have to wait for other threads when visiting database, they has the completion and not 100% work in parallel. Also, with less requests, the difference in latency is less because it is more impacted by other outside factors. We could get the conclusion that 4 core system has a larger throughput when process large amount of request.

Core\Request	1000	1500	2000
1	39.406s	67.37s	146s
1	38.4s	65.1s	144.2s
2	38.133s	55.21s	97.813s
2	39.8s	54.7s	101.967s
4	34.4s	55.94s	80.9s
4	33.9s	59.75s	83.23s