Exchange Server Scalability Report

1. Introduction:

We implement an exchange matching engine, which will match buy and sell orders for a stock/commodities market. We developed our program using C++/C and PostgreSQL as the database. Then, we build tables including Symbol, Account, Position, Transaction, and Deal. The Transaction Table records the original deal order details and the status of the deal, while the Deal Table records the separate sub-orders of the original order details. The clients can send data in XML format to create their accounts, add symbols, make transactions, query transactions, and cancel opened transactions.

2. Implementation:

We first build our program with a single thread, which means we can only connect to one client and process one request each time. During this period, we test the functionality of our project and test the throughput and latency in the single thread process. Then we develop our program at a multi-thread and realize atomic operation in the database. For scalability, we test the throughput and latency of our program and analyze the performance of our project.

For the server, we create a thread per request and pre-create a set of threads. Every time we receive a single request, we create a single thread to handle it. This thread will exit after sending the response back to the client but this may cause overhead problems. We also create a fixed number of threads while setting up our program. The new coming requests will be distributed to those threads that are free to lower the overhead.

For the database, we add “set transaction isolation level repeatable read;” to handle the concurrency in our database.

For testing, each client will randomly send different request XML types, including <create> with <symbol>/<account> tags and <transactions> with <order>/<query>/<cancel> tags.

3. Test Result:

3.1 One client

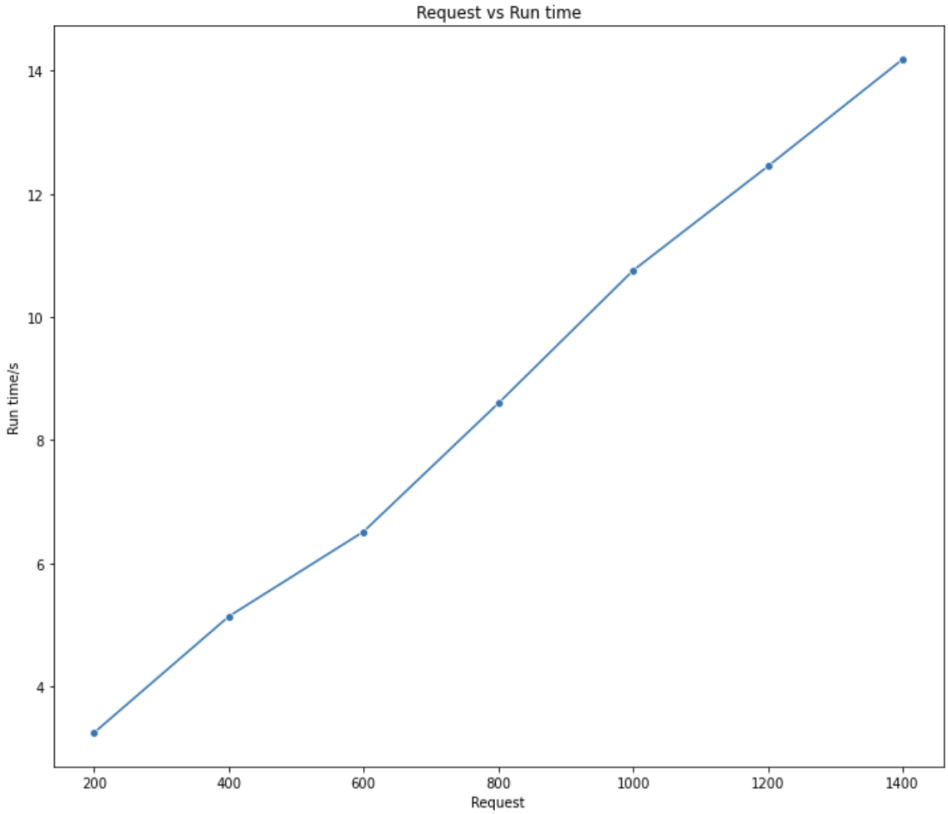
Core Num	Request Num	Total run time/s	Throughput
1	500	5.7099	87.581
2	500	5.75079	86.956
4	500	5.22666	95.785

Core Num	Request Num	Total run time/s	Throughput
1	1000	11.2698	88.732
2	1000	9.26781	107.991
4	1000	10.7745	92.811

Core Num	Request Num	Total run time/s	Throughput
1	1500	13.9459	107.557
2	1500	13.7841	108.821
4	1500	14.861	100.942

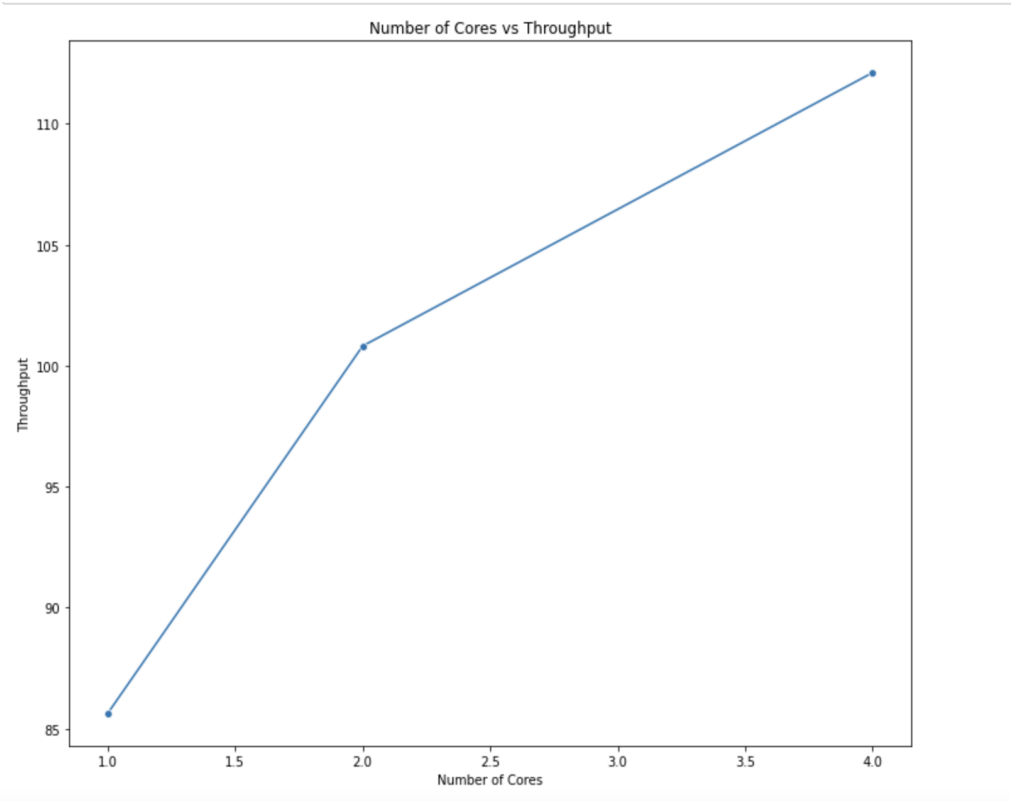
One-core:

Request	200	400	600	800	1000	1200	1400
Run time/s	3.24298	5.13298	6.51076	8.59601	10.753	12.4461	14.1846
Throughput	61.671	77.942	92.165	93.131	92.991	96.463	98.698


Request vs Run time
Run time/s
Request
14
12
10
8
6
4
200
400
600
800
1000
1200
1400

3.2 Two clients

Core Num	Request Num	Total run time/s		Avg latency	Throughput
		Client 1	Client 2		
1	500	5.84464	5.49556	0.0287302	85.616
2	500	4.96423	4.75233	0.0284167	100.806
4	500	4.46075	4.26235	0.0258710	112.108



4. Conclusion

Based on the figures above, when the number of cores increases, the throughput improves a lot. The latency will increase and the contention of hardware resources will decrease. Though these are in opposite ways, the contention of the hardware resource has a larger influence on the throughput, resulting in higher overall throughput.