ECE568 HW3 Report

Kaiyuan Li, Shuyan Sun

March 21, 2022

Introduction

In HW4, our group implemented a server that receives requests from clients, and then performs a series of related operations in the database. Then the server gives a response to the client. In this assignment, we also tested our server from scalability perspective.

XML Parser

Our group used Xerces as the XML parser library.

Before parsing the received XML, the program would first check the content of the XML in order to make sure that it is valid.

Database

Our group created three relations. The first relation is ACCOUNT, which records the unique account_id and balance in the account. The other two relations are POSITION and ORDERS. Their attributes are as following.

ACCOUNT(<u>(ACCOUNT_ID)</u>, BALANCE)

POSITION(<u>(POSITION_ID)</u>, ACCOUNT_ID, SYMBOL, AMOUNT)

ORDERS(<u>(ORDER_ID)</u>, TRANS_ID, ACCOUNT_ID, SYMBOL, AMOUNT, PRICE, TIME, STATUS)

We also created a type called status which has three statuses: 'open', 'canceled', and 'executed'.

Creating Tables

After the program started the database, it would first checked if those three relations are exist. If they are exist, then they would be dropped automatically and database creates new ones.

Matching Orders

After a new order has been created, it would be matched with the previous orders in the databases automatically. The program would first judge it is a sell order or a buy one, and then match eligible ones. If one sell order is matched with one buy order with the same amount, they would both be updated to 'executed' status. If their amounts are not equal, one new order would be inserted into the relation with 'open' status.

Canceling Orders

If a cancellation operation occurs, the program would find the order and update its status to 'canceled'. Meanwhile, the information in related account and position would be updated. If it is a sell order, the amount in the canceled order would increase in the related position, while if it is a buy order, the cost the order took would back to the related account.

Test Part

create per request

This kind of thread strategy is one thread is created to handle a request when the server receives a new request. In addition, the created thread would exist after that.

Using the following commands separately to run the server.

// running in one core server

taskset -c 0 ./server

// running two-core server

taskset -c 0,1 ./server

// running four-core server

taskset -c 0-3 ./server

Test Results

Thread Strategy	Core Number	Handled Request	Runtime	Thread Number
Create per Req	1	1160	10.53	1000
Create per Req	2	2316	10.20	1000
Create per Req	4	4650	10.12	1000

Scalability

According to the following figure on request numbers and latency, we could learn that with the increasing of request numbers, the latency increases linearly. However, when the engine is run by 2-core system, the slope of curve particularly larger than 4-core system, which means higher latency. The reason is that 4-core system has larger throughout, and thus could handle requests in the shorter time.