HW4for590

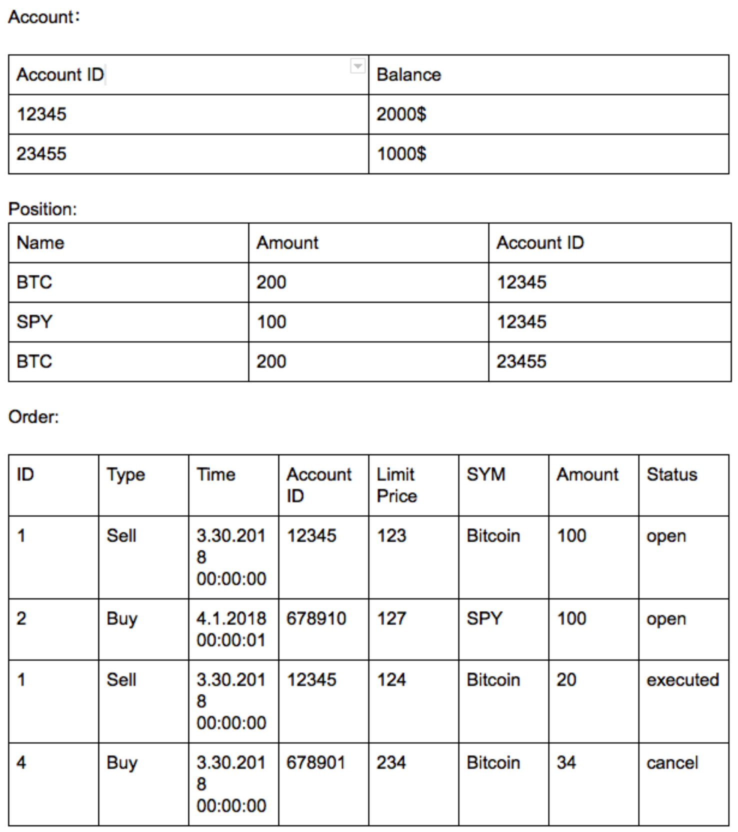
hesong huang,yuan feng

April 2018

1 Introduction

(1) Database design

In our implementation, we designed three tables to store account information, positions and orders:



Account:

Account ID	Balance
12345	2000$
23455	1000$

Position:

Name	Amount	Account ID
BTC	200	12345
SPY	100	12345
BTC	200	23455

Order:

ID	Type	Time	Account ID	Limit Price	SYM	Amount	Status
1	Sell	3.30.2018 00:00:00	12345	123	Bitcoin	100	open
2	Buy	4.1.2018 00:00:01	678910	127	SPY	100	open
1	Sell	3.30.2018 00:00:00	12345	124	Bitcoin	20	executed
4	Buy	3.30.2018 00:00:00	678901	234	Bitcoin	34	cancel

Figure 1: Three database tables

(2) XML parsing

In this project, we utilized RapidXml, parsing strings into a DOM Trees. As it is parsing, we called the functions simultaneously, and write results in a DOM Tree. And we send back the DOM Tree to the client as strings in the formate of XML.

2 Methods to improve scalability

(1) Block and Unblock Socket

We choose block module to receive xml requests.

If we block socket when receiving request, program would wait until a new request arrives. If we use the unblock method using select function to receive request, the select function would wait for specific seconds. If no request arrive during this period, select function would return and the program would keep moving on. However, in this project, using block and non block methods have tiny influence on scalability, because the main job of our program is to wait for the coming request and create a new thread to handle that request. So in our main function, the program do nothing but wait for request.

(2) Mutex Lock and Database Lock

In this program, race condition may happen when different threads. For example, two buy requests may match the same sell order.

We tried two methods to prevent race condition. One is adding mutex between each transaction. The other is using postgres row lock. With this row lock, when one thread is reading and editing a row in a table, the other thread cannot read and edit this row. Finally, we chose to use database row low.

(3) Index

We created three tables using postgresql: account, position and order. Index in database is used for speed up the performance of queries. In our program, we add index for each table: ACCOUNT(ACCOUNT ID); POSITION(ACCOUNT ID,SYMBOL); ORDERS(TYPE,STATUS,SYMBOL);

3 Test cases

We sent lots of xml files a time to our server using .sh. Following are the some of the xml we select to represent the test cases. Test cases are divided in two parts:

1. Test cases:

   (a) Create account and symbol requests:

```
161
<?xml version=1.0 encoding=UTF-8?>
<create>
```

```
  <account id="26" balance="1000"/>
  <symbol sym="SYM26">
    <account id="26">1000</account>
  </symbol>
</create>
```

(b) Transaction requests:

```
230
<?xml version=1.0 encoding=UTF-8?>
<transactions accountid="1">
  <order sym="SYM1" amount="-500" limit="11"></order>
</transactions>
<transactions accountid="1">
  <order sym="SYM0" amount="50" limit="20"></order>
</transactions>
```

(c) Cancel requests:

```
140
<?xml version=1.0 encoding=UTF-8?>
<transactions accountid="12334">
  <cancel uid="2" ></cancel>
  <cancel uid="9" ></cancel>
</transactions>
```

(d) Query requests:

```
135
<?xml version=1.0 encoding=UTF-8?>
<transactions accountid="1">
  <query uid="1" ></query>
  <query uid="2" ></query>
</transactions>
```

2. Results:

(a) Created

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
  <created ID="26"/>
  <created ID="26" SYM="SYM26"/>
</results>
```

(b) Opened

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
  <opened SYM="SYM1" amount="-500" limit="11" uid="9"/>
</results>
<results>
  <opened SYM="SYM0" amount="50" limit="20" uid="10"/>
</results>
```

(c) Canceled

```
     <?xml version="1.0" encoding="UTF-8"?>
<results>
        <canceled>
                <canceled shares="50" time="1523054343.09397"/>
        </canceled>
        <canceled>
                <canceled shares="500" time="1523054343.16968"/>
        </canceled>
</results>
```

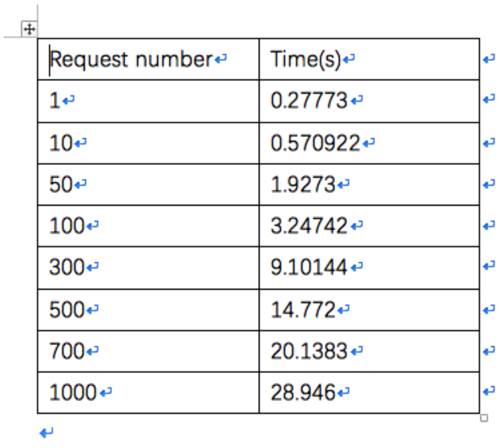
(d) Query

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
  <status>
    <open shares="450"/>
    <executed shares="50" price="11" time="1523054086.92808"/>
  </status>
  <status>
    <canceled shares="50" time="1523054343.09397"/>
  </status>
</results>
```

4 Scalability

To test scalability, we need to compare the running time of serialization method and parallel method. We send a bunch of xml files at a time to our server. We increase the number of xml files from one to one thousand, and the execution time is shown in following table. this table only shows the execution time of parallel method:

As we can see in these figures, parallel method increased the scalability of program. As the request number increases, the capability to improve scalability tends to be a stable value.



Request number	Time(s)
1	0.27773
10	0.570922
50	1.9273
100	3.24742
300	9.10144
500	14.772
700	20.1383
1000	28.946

Figure 2: Request number and whole executed time

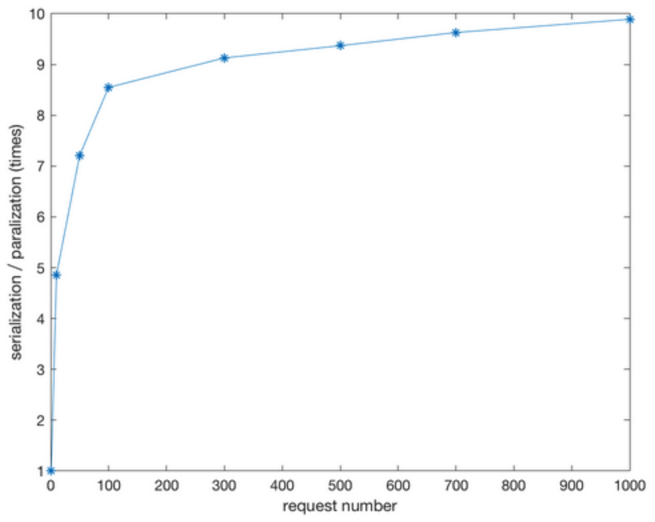


Figure 3: use serialization method / use parallel method