Database Assignment 2 Report

TEAM 6 107062318 李俊逸 107062202 陳敬和 107062237 張濬洋

A.Implementation

Modified files:
- As2BenchRte
- As2BenchJdbcExecutor
- As2BenchStoredProcFactory

Added files:
- UpdatePriceParamGen
- UpdateItemTxnJdbcJob
- UpdateItemProcParamHelper
- UpdatePriceProc

Output:
- StatisticMgr

As2BenchRte

Randomly generate a number from 0~99 and let Read_Write_Tx_rate*100 becomes the threshold to control the rate between ReadItem and UpdateItem, and passes the argument needed by the executor then returns.

Also, in getTxExecutor, we need to add the update_item_price type to get the proper paramGenerator.

As2BenchJdbcExecutor

Add a new case, UpdateItem, for the Transaction Executor.

```
case UPDATE_ITEM_PRICE:
    return new UpdateItemTxnJdbcJob().execute(conn,pars);
```

As2BenchStoredProcFactory

Add the updateItemPrice procedure into the SP factory so that we can execute the updateItemPrice procedure.

UpdatePriceParamGen

Generate Ids of items that need to be updated and the raises. We store raises as double and Ids as int into an object list, and they will be decomposed in the UpdateItemProcParamHelper.

UpdateItemTxnJdbcJob (execute())

Update item prices in the jdbc environment with the conditions that assignment 2 is given. Details of implementation can be found in the following contents.

First, we process the parameters(itemIds and their corresponding prices) and make them into two lists in paramHelper. After that, we get the price of an item by its itemId through the SELECT sql command.

After we get the original prices, we will determine if the prices have already exceeded MAX_PRICE or not, then decide the update price of the UPDATE sql command and execute it.

After all of the sql commands are finished(i.e. all records have been updated), we commit the transaction through the connection.

UpdateItemProcParamHelper (getxxx(), prepareParameters())

Since we need ids and raises generated from the benchmark project to update records, we initialize the id and raise list, and their get function.

As for the implementation of the prepareParameters function, the argument we send into the function is what we have created in the UpdatePriceParamGen class. The argument looks like [number of items, itemId1, raise1, itemId2, raise2, …]. We parse it and fill in the raises(double) and updateItemIds(int) arrays with the argument's element.

UpdatePriceProc (executeSql())

This implementation is quite similar to the UpdateItemTxnJdbcJob. The major difference is that we use native interface, rather than the JDBC interface here. So nothing more to be explained.

StatisticMgr (outputReport)

In the implementation, we need to be careful about the time unit and also the edge cases. For example, we should keep the last batch result, even if the execution time is not over 5.

B.CSV Result

i. JDBC with READ_WRITE_TX_RATE = 0.5

time(sec)	throughput(txs)	avg_latency(ms)	min(ms)	max(ms)	25th_lat(ms)	median_lat(ms)	75th_lat(ms)
5	797	6	4	13	5	6	6
10	733	6	4	50	5	6	6
15	828	6	4	18	5	5	6
20	855	5	4	13	4	5	6
25	838	5	4	28	5	5	6
30	800	6	4	17	5	5	6
35	819	6	4	15	5	5	6
40	832	6	4	30	4	5	6
45	861	5	4	13	4	5	6
50	857	5	4	14	4	5	6
55	836	5	4	14	5	5	6
60	833	6	4	16	4	5	6
65	840	5	4	34	4	5	6
70	846	5	4	20	4	5	6
75	840	5	4	16	4	5	6
80	811	6	4	20	5	5	6
85	820	6	4	14	5	5	6
90	674	7	4	88	5	5	7
95	848	5	4	15	4	5	6
100	850	5	4	25	4	5	6
105	836	5	4	15	5	5	6
110	809	6	4	14	5	5	6
115	816	6	4	25	5	5	6
120	779	6	4	12	5	5	6

ii. SP with READ_WRITE_TX_RATE = 0.5

time(sec)	throughput(txs)	avg_latency(ms)	min(ms)	max(ms)	25th_lat(ms)	median_lat(ms)	75th_lat(ms)
5	9457	0	0	2	0	0	0
10	9486	0	0	2	0	0	0
15	9385	0	0	2	0	0	0
20	9366	0	0	2	0	0	0
25	9340	0	0	3	0	0	0
30	9453	0	0	4	0	0	0
35	9472	0	0	4	0	0	0
40	9461	0	0	3	0	0	0
45	9461	0	0	4	0	0	0
50	9403	0	0	3	0	0	0
55	9441	0	0	3	0	0	0
60	9392	0	0	4	0	0	0
65	9413	0	0	3	0	0	0
70	9499	0	0	3	0	0	0
75	9438	0	0	3	0	0	0
80	9481	0	0	4	0	0	0
85	9426	0	0	5	0	0	0
90	9400	0	0	4	0	0	0
95	9482	0	0	3	0	0	0
100	9501	0	0	3	0	0	0
105	9486	0	0	4	0	0	0
110	9434	0	0	4	0	0	0
115	9385	0	0	4	0	0	0
120	7883	0	0	2	0	0	0

C.Performance Experiments

1. Experiment environment：Intel core i7-8750H@2.20GHz,16GB Ram, 1T HDD, Windows 10

2. The performance comparison

Throughput(Txn/min)	JDBC	SP
100% ReadItem	11413	153582.5
50% ReadItem 50%Update	9829	112473.5
100% UpdateItem	9087.5	90711.5

3. Analysis and explanation
   a. JDBC and Store Procedures

      如上圖的CSV result，可以看出SP的結果比JDBC的還要快，我們認為是因為SP在server端跑，而JDBC則是在client端上運作，需要額外的傳輸時間。

   b. Different Ratios of ReadItemTxn and UpdateItemPriceTxn
      i. All read

time(sec)	throughput(txs)	avg_latency(ms)	min(ms)	max(ms)	25th_lat(ms)	median_lat(ms)	75th_lat(ms)
5	836	5	4	18	5	5	5
10	922	5	4	34	4	5	5
15	903	5	4	13	4	5	5
20	981	5	4	11	4	5	5
25	952	5	4	13	4	5	5
30	940	5	4	14	5	5	5
35	969	5	4	12	4	5	5
40	966	5	4	13	4	5	5
45	941	5	4	13	4	5	5
50	967	5	4	12	4	5	5
55	949	5	4	22	4	5	5
60	959	5	4	11	4	5	5
65	971	5	4	13	4	5	5
70	960	5	4	13	4	5	5
75	953	5	4	27	4	5	5
80	984	5	4	13	4	4	5
85	963	5	4	12	4	5	5
90	964	5	4	12	4	5	5
95	952	5	4	32	4	5	5
100	980	5	4	11	4	5	5
105	995	5	4	13	4	4	5
110	999	5	4	16	4	4	5
115	907	5	4	44	4	4	5
120	913	5	4	38	4	5	5

ii. 50% read / 50% update

time(sec)	throughput(txs)	avg_latency(ms)	min(ms)	max(ms)	25th_lat(ms)	median_lat(ms)	75th_lat(ms)
5	797	6	4	13	5	6	6
10	733	6	4	50	5	6	6
15	828	6	4	18	5	5	6
20	855	5	4	13	4	5	6
25	838	5	4	28	5	5	6
30	800	6	4	17	5	5	6
35	819	6	4	15	5	5	6
40	832	6	4	30	4	5	6
45	861	5	4	13	4	5	6
50	857	5	4	14	4	5	6
55	836	5	4	14	5	5	6
60	833	6	4	16	4	5	6
65	840	5	4	34	4	5	6
70	846	5	4	20	4	5	6
75	840	5	4	16	4	5	6
80	811	6	4	20	5	5	6
85	820	6	4	14	5	5	6
90	674	7	4	88	5	5	7
95	848	5	4	15	4	5	6
100	850	5	4	25	4	5	6
105	836	5	4	15	5	5	6
110	809	6	4	14	5	5	6
115	816	6	4	25	5	5	6
120	779	6	4	12	5	5	6

III. All update

time(sec)	throughput(txs)	avg_latency(ms)	min(ms)	max(ms)	25th_lat(ms)	median_lat(ms)	75th_lat(ms)
5	753	6	5	34	5	6	7
10	778	6	5	12	5	5	7
15	787	6	5	13	5	5	7
20	788	6	5	12	5	5	7
25	773	6	5	14	5	5	7
30	742	6	5	60	5	6	7
35	644	7	5	66	5	6	7
40	755	6	5	28	5	6	7
45	766	6	5	13	5	6	7
50	755	6	5	14	5	6	7
55	765	6	5	13	5	6	7
60	749	6	5	20	5	6	7
65	753	6	5	13	5	6	7
70	754	6	5	17	5	6	7
75	780	6	5	22	5	5	6
80	745	6	5	14	5	6	7
85	774	6	5	13	5	5	7
90	781	6	5	20	5	5	7
95	676	7	5	31	5	6	7
100	755	6	5	29	5	5	7
105	784	6	5	13	5	5	7
110	776	6	5	15	5	5	7
115	774	6	5	36	5	5	7
120	768	6	5	27	5	5	7

藉由上面三個不同ratio的csv result可以得出，update所花費的時間比read還要久，因為update除了read外，還需要更新price的value，因此需要更長的時間。