Team24_Assignment 2

106062116 黃晨
106062216 馮謙
106034061 曾靖渝

Implemtation

BenchMark Flow：

經過漫長的Trace code之後，我們發現是**由As2BenchRte**這個class去實作client的thread，每個class會根據getNextType()回傳**的As2BenchTxnType**，呼叫不同種類**的 As2BenchTxnExecutor**，Executor則會根據connection_mode決定要呼叫jdbc還是sp的method。

TxnType and read_write_rate:

要實作**UdpatePrice**的txn，從上方的流程可以得知需要先新增一個As2BenchTxnType，名為**UPDATE_ITEM**。另外根據作業要求，我們在**valinabench.property**中新增了**READ_WRITE_TX_RATE**這個參數，會在後面提到它的用法。

As2BenchRte:

每個Rte中有兩個executor，分別負責READ以及UPDATE。Rte會先call **getNextType()**來決定要執行哪種txn。**getNextType()**會根據**READ_WRITE_RATE**的數值隨機回傳READ或UPDATE。舉例來說，若數值為0.8，則有80%機率回傳UPDATE，20%回傳READ。之後Rte則根據回傳的txnType決定要用哪種executor執行txn。

As2BenchTxnExecutor:

每個As2BenchTxnExecutor中有一種ParameterGenerator(簡稱pg)，一種pg對應一種txnType，為這種Txn提供所需要的參數(item的price、id)。經由Rte call execute() method之後，**As2BenchTxnExecutor**會根據connection_mode決定呼叫JDBC或SP的method，並傳入txntype作為參數，決定要執行哪種txn。

As2UpdateItemPriceParamGen:

本次作業的要求為修改隨機10個item的price，而修改的值也是隨機的(0.0~5.0)。由此可知我們需要的參數包括UpdateCount(10)，10個item的id，10個修改值。前面兩種都跟ReadItem的部分一樣，而10個修改值我們則用**rvg.fixedDecimalNumber(1, 0.0, 5.0)**實作。這種method會回傳小數點後一位，0.0~5.0的值。

JDBC：

若connection_mode為JDBC，**則As2BenchTxnExecutor會呼叫As2BenchJdbcExecutor執行。As2BenchJdbcExecutor**則會根據txnType呼叫不同的jdbcjob執行，因此我們需要新增一種**UpdateItemPriceTxnJdbcJob**的class。在這個class中，我們首先根據pg的產生的參數格式，將其讀取到itemIds，priceRaise這兩個陣列中。再來將id符合的項目從item table中select出來。接下來判斷price是否超過**由As2BenchConstants** import的MAX_PRICE，如果有，則將原本的price改成MIN_PRICE。如果沒有，則將原本的price加上priceRaise中的值。之後將update過後的price透過**statement.executeUpdate()**將資料庫中的price update。

Store Procedure：

As2BenchStoredProcFactory

再進入**TransactionExecutor**之後，會JDBC 和 Store procudeure會分歧，如果server使用stored procedures連接，初始化時會創建 **StoredProcedureFactory。StoredProcedureFactory**會call **callStoredProc，其中getStroredProcedure(int pid)**，為了 UpdateItemPriceTxn新增他的case，可以發現他需要stored procedures的實作。

UpdatePriceProc

UpdateItemPriceTxn實作，與jdbc不同這是server上自己對自己下query，使用的是 Native query interface。會根據READ or UPDAT 去創建**paramHelper**，並以function call的方式回傳這個 query所用到的 parameter。產生 query string去create plan，後 scan拿到 result set，作價格判斷。 產生更新價格的 query string，然後call **VanillaDb.newPlanner().executeUpdate(sql, tx)**去更新Planner tree。關閉scan。

UpdateItemProcParamHelper

stored procedures與 jdbc 分歧其中在 **org.vanilladb.bench.rte.TransactionExecutor**之後，他要**callStoredProc()**，用TxnType enum 拿pid，用Transaction的Object list當作parameter。而 這裡就是幫助server處理Object list的地方。然後提供**paramHelper**給上面真正做 stored procedures所使用，分解list同jdbc。然而他要**createResultSet**，有可能是要回給client訊息或是去認定這個 query有沒有執行成功，這裡要回一個**SpResultRecord**。

StatisticMrg：

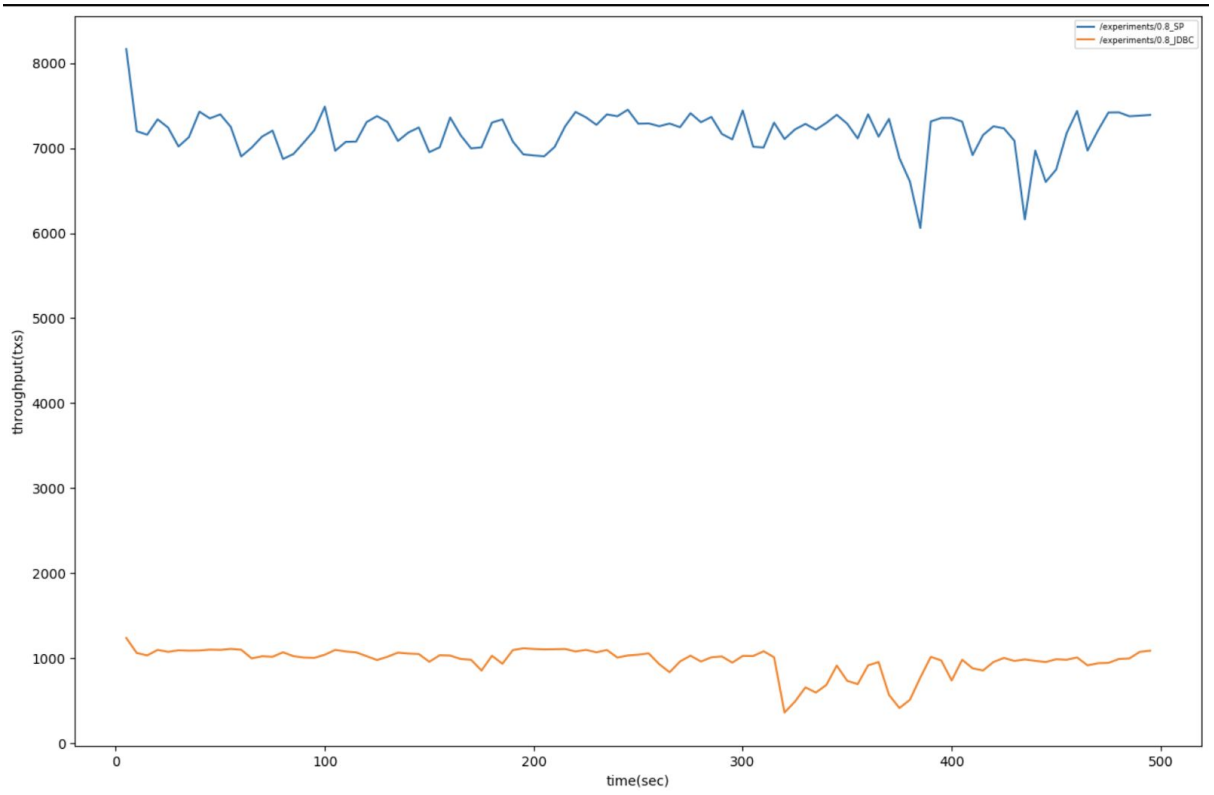
因為生成區間是以每 5 秒為一個單位，所以一開始先初始化 time 為 5，然後開始累積這區間內的資料，如果資料的 endTime - startTime > time，就代表說這區間的資料已經收集完畢了，所以就可以輸出成一行的 [throughput(txs), avg_latency(ms), min(ms), max(ms), 25th_lat(ms), median_lat(ms), 75th_lat(ms)]，至於四分位數的算法，我們是另外寫成一個 fourDivision 的函式（其實就只是把資料由小 sort 到大然後取25%、50%、75%）。

Experiments

0.2_JDBC

time(sec)	throughput(txs)	avg_latency(ms)	min(ms)	max(ms)	25th_lat(ms)	median_lat(ms)	75th_lat(ms)
5	1503	3	3	12	3	3	4
10	1260	3	3	15	3	3	3
15	1290	3	3	9	3	3	3
20	1277	3	3	12	3	3	3
25	1258	3	3	12	3	3	3
30	1245	4	3	12	3	3	4
35	1257	3	3	12	3	3	4
40	1128	4	3	28	3	3	4
45	1182	4	3	11	3	3	4
50	1204	4	3	11	3	3	4
55	1272	3	3	14	3	3	3
60	1198	4	3	14	3	3	4
65	1166	4	3	16	3	3	4
70	1199	4	3	13	3	3	4
75	1171	4	3	12	3	3	4
80	1161	4	3	14	3	3	4
85	1168	4	3	14	3	3	4
90	1170	4	3	38	3	3	4
95	1206	4	3	20	3	3	4
100	1236	4	3	12	3	3	4
105	1276	3	3	13	3	3	3
110	1256	3	3	14	3	3	3
115	1200	4	3	17	3	3	4
120	1167	4	3	17	3	3	4
125	1222	4	3	28	3	3	4

1. Environment
    a. 2.7 GHz 四核心Intel Core i7
    b. 8 GB 2133 MHz LPDDR3
    c. 256G SSD
    d. Mac OS Catalina 10.15.12
2. JDBC vs Store Procedure
    a. SP 的 throughput 明顯大於 JDBC
    b. 因為SP 是直接在server端下指令（native interface），只要花傳送參數的時間，不需要花傳送指令的時間，所以節省了很多時間。JDBC傳送參數與指令都需要花時間，故比較久。



3. Update vs Read
    a. 從ratio的大小也可以發現，越小的ratio速度也比較快，因此可以得出READ 的速度比 UPDATE 快。
    b. update 比read指令慢可能是因為update傳的參數比多或是其在實作中下的sql指令也比較多，所以建置並修改的planner tree的次數比較多。