Team16 Assignment 2 Phase 1 Report

109062123 曹瀚文 109062323 吳聲宏 109062330 黃鈺臻

I. Implementation

I.1 Implement the UpdateItemPriceTxn

註：path開頭都是bench/src/main/java/org/vanilladb/bench/，統一縮略為.../。

file: .../benchmarks/as2/As2BenchTransactionType.java

- **brief**：新增type，用以辨識UpdateItemPriceTxn。
- **detail**：在enum As2BenchTransactionType新增一項UPDATE_ITEM(true)，true表示isBenchProc。有此標記在benchmarking時才知道要統計哪些txn。

file: .../benchmarks/as2/rte/UpdateItemParamGen.java

- **brief**：藉由在Client端產生參數來生成client request，用於benchmarking。
- **detail**：照著ReadItem的去寫，差別只在UpdateItem需要的參數是NUM_ITEMS和十個隨機產生的ID、價格。先放NUM_ITEMS，再用RandomValueGenerator產生，再依序放進paramList即完成。也寫了getTxnType()，會return UPDATE_ITEM。

file: .../server/param/as2/UpdateItemPriceParamHelper.java

- **brief**：Server端解析UpdateItemPriceTxn的參數用。
- **detail**

  - 同樣參照ReadItem的版本來寫。在prepareParameters()中，改成先取出num_items，再取出十個ID、價格，存在private variable中。並創立相應的getNumberOfItems、getRandomId、getUpdateValue等function讓這些params可以被別人獲取。
  - 另外，由於一般的UPDATE query不會在result set中包含每一個updated row info，而是回傳updated count，這邊的result set 我們選擇只回傳成功與否。因此getResultSetSchema、newResultSetRecord就都直接construct並回傳就好。isCommit的資訊會在別處包進resultset中，形成最後的resultset。

file: .../benchmarks/as2/rte/jdbc/UpdateItemPriceTxnJdbcJob.java

- **brief**： UpdateItemPriceTxn的JDBC實作
- **detail**：改自ReadItemTxnJdbcJob。先呼叫paramhelper準備好params，再loop 10次，每次都 select i_price。再跟As2BenchConstants.MAX_PRICE比較，決定new value是MIN_PRICE還是i_Price + i_updateValue。計算好後再呼叫executeUpdate來UPDATE即可。其餘架構都一樣。在return的部分，因為設計成只要回傳狀態，因此return VanillaDbJdbcResultSet(true, "Success")。

file: .../server/procedure/as2/UpdateItemPriceTxnProc.java

- brief：UpdateItemPriceTxn的SP實作
- detail：仿照ReadItemTxnProc去改，改動部分的邏輯、用到的function和JDBC實作都類似。同樣呼叫executeUpdate()，只是是StoredProcedureHelper的而非用statement去呼叫。

file: .../benchmarks/as2/rte/jdbc/As2BenchJdbcExecutor.java

- brief：把使用JDBC者接到UpdateItemPriceTxn的JDBC實作
- detail：在As2BenchJdbcExecutor的execute()中，會查找txn type並execute相應的JDBC job。在此我們新增case UPDATE_ITEM，執行UpdateItemPriceTxnJdbcJob()並回傳result set。

file: .../server/procedure/as2/As2BenchStoredProcFactory.java

- brief：把使用SP者接到UpdateItemPriceTxn的SP實作
- detail：想要找SP時，就呼叫getStroredProcedure()，會查找txn type並對應到SP實作中。在此我們新增case UPDATE_ITEM並回傳UpdateItemPriceTxnProc的實例。呼叫者可以用此實例再進一步execute。

file: .../benchmarks/as2/rte/As2BenchmarkRte.java

- brief：把executor往呼叫者的方向接起來
- detail：執行不同的txn時，會呼叫不同的paramgen，也因此executor的初始化會不同。所以在As2BenchmarkRte()中為read,update各開一個executor，並在getTxExeutor中新增UPDATE_ITEM會回傳update executor這個case。

I.2 Add READ_WRITE_TX_RATE Property

file: .../vanillabench.properties

- 新增READ_WRITE_TX_RATE這個property，並設為0~1間的數字。

file: .../benchmarks/as2/rte/As2BenchmarkRte.java

- brief：讓read, write的比例可以被READ_WRITE_TX_RATE調整。
- detail：改動getNextType function，用機率選擇下一個txn要做甚麼時。隨機生成的數字<READ_WRITE_TX_RATE就選read，反之選write，就可以大致控制兩者的比例。

I.3 Produce an Additional Report

file: .../StatisticMgr.java

- brief：加入avg, min, …等更多throughput資訊並輸出成CSV
- detail

  - 我們另開一個outputCsvReport()來實作，沒有更動原有的outputDetailReport()。

  - 作業要求的是計算throughput(txn/min)，而在outputDetailReport()中是用response time總和/txn數，我們覺得可能不太準。txn之間有可能concurrently執行，導致同一段時間被算兩次，或不是一個執行完就緊接著另一個執行，中間空缺的時間會被少算。throughput鐘的時間計算應該是真實環境時間，但用response time推估的不準，因此我們改用end time。

  - 在outputCsvReport()中，把最後一個txn end time - 第一個txn end time，再除以(txn數目-1)，就是avg throughput。減去第一個txn end time是因為我們沒有start time，只好用乾脆不算第一個txn。

  - latency則跟原本算法一樣。

II.Experiment

II.1 Setting

測試環境：AMD Ryzen 7 4700U @ 2.00 GHz, 8 GB RAM, 512 GB SSD, Windows 11
實驗對象：分別控制READ_WRITE_TX_RATE和CONNECTION_MODE，其他設定都沒變。
觀察項目：

1. throughput(txn/min)：使用我們用end, record_start_time算出的throughput。每5ms會有一個統計結果，再取平均即表中數值。
2. aborted/committed：分別顯示read/update的aborted總數/committed總數
3. avg_latency：原本report就有用response time算整體的平均，直接拿來用。

II.2 Data and analysis

mode/rate	throughput	avg_latency	aborted	committed
SP/0.3	~18700	0 ms	0 / 5	224073
SP/0.6	~19000	0 ms	0 / 1	229146
SP/1.0	~23500	0 ms	0 / 0	282484
JDBC/0.3	~1205	8 ms	0 / 0	14474
JDBC/0.6	~1250	8 ms	0 / 0	15048
JDBC/1.0	~1230	8 ms	0 / 0	14812

- JDBC和SP相比，throuput明顯大很多。因為SP是資料庫系統本身就寫好的一套流程，要使用時不用透過傳送parameter、輸入SQL等流程，因此會比JDBC快。

- 我們的update txn是先讀出再判別是否更新，因此同時有read, write操作，有可能holding read device lock 卻在等write的lock而造成deadlock。但只有read時是不會deadlock的，因為用完就保證會放掉，也不會有抓一個等一個的問題。因此在SP中可以看到，隨著update比例下降，為了預防deadlock而abort的數量變少了。同時也因為read更簡單、且不用對他做deadlock檢測，read比例上升時throughput也上升。

- 至於為何ratio在SP上影響幅度可達一兩成，卻在JDBC上影響幅度更小? 可能是因為SP的CPU time更少，所以memory time、lock的狀況對他影響更大，但對於要花很多時間在CPU time上的JDBC來說，read, update這兩種memory access行為變化帶來的影響就沒有那麼大。

- 若要比較mode, ratio兩種因子的影響力，前者還是大很多的，可從avg_latency看出來。avg_latency基本上取決於mode，JDBC都8 ms。