Assignment 2 Report

110062229 翁語辰
110081014 程詩柔
110062171 陳彥成

1. Implementation

- org.vanilladb.bench.server.procedure.as2.UpdatePriceProc
  executesql(): 把update需要的data(item id, price)用getter從UpdatePriceProcParamHelper取得。然後用StoredProcedureHelper的executeQuery()、和executeUpdate()寫上與JDBC同樣步驟的sql。
  如果沒有先實作這部分，會出現”’no operation implement”’的warning，而且結果跑不出來。
- org.vanilladb.bench.benchmarks.as2.rte.jdbc.As2BenchmarkRte
  getNextTxType(): 利用random的結果和READ_WRITE_TX_TYPE比較來決定下一個txn是update還是read

```
protected As2BenchTransactionType getNextTxType() {
    double readItemRatio = VanillaBenchParameters.READ_WRITE_TX_RATE;

    if (Math.random() < readItemRatio) {
        return As2BenchTransactionType.READ_ITEM;
    } else {
        return As2BenchTransactionType.UPDATE_PRICE;
    }
}
```

  getTxExecutor(): 新增一個case，當type是UPDATE_PRICE的時候回傳的Executor args裡要放入UpdatePriceParamGen這個class

```
protected As2BenchmarkTxExecutor getTxExeutor(As2BenchTransactionType type) {
    switch (type) {
        case READ_ITEM:
            return new As2BenchmarkTxExecutor(new As2ReadItemParamGen());
        case UPDATE_PRICE:
            return new As2BenchmarkTxExecutor(new UpdatePriceParamGen());
        default:
            throw new IllegalArgumentException("Unsupported transaction type: " + type);
    }
}
```

- org.vanilladb.bench.benchmarks.as2.rte.UpdatePriceParamGen
  generateParameter(): 在arraylist中先依序放入要update的數量(10)、(要update的index、要raise的value) * 10次，最後回傳這個array

```
@Override
public Object[] generateParameter() {
    RandomValueGenerator rvg = new RandomValueGenerator();
    ArrayList<Object> paramList = new ArrayList<Object>();

    // Set update count
    paramList.add(TOTAL_UPDATE_COUNT);

    // Generate item IDs and price raises
    for (int i = 0; i < TOTAL_UPDATE_COUNT; i++) {
        // Generate a random item ID to update
        paramList.add(rvg.number(min:1, As2BenchConstants.NUM_ITEMS));

        // Generate a random price raise value between 0.0 and 5.0
        paramList.add(rvg.fixedDecimalNumber(decimal:1, min:0.0, max:5.0));
    }

    return paramList.toArray(new Object[0]);
}
```

- org.vanilladb.bench.benchmarks.as2.rte.jdbc.As2BenchJdbcExecutor
  execute(): 增加一個case，當type是UPDATE_PRICE的時候要呼叫UpdatePriceJdbcJob.execute()

```
@Override
public SutResultSet execute(Connection conn, As2BenchTransactionType txType, Object[] pars)
        throws SQLException {
    switch (txType) {
    case UPDATE_PRICE:
        return new UpdatePriceJdbcJob().execute(conn, pars);
    case TESTBED_LOADER:
        return new LoadingTestbedJdbcJob().execute(conn, pars);
    case CHECK_DATABASE:
        return new CheckDatabaseJdbcJob().execute(conn, pars);
    case READ_ITEM:
        return new ReadItemTxnJdbcJob().execute(conn, pars);
    default:
        throw new UnsupportedOperationException(
                String.format("no JDCB implementation for '%s'", txType));
    }
}
```

- org.vanilladb.bench.server.param.as2.UpdatePriceProcParamHelper
  大致上和ReadItemProcParamHelper相同，只是prepareParameters會根據UpdatePriceParamGen產生出來的param格式不同而做不同處理，getResultSetSchema和newResultSetRecord也會因為schema不同而需要做一些更改

```java
@Override
public void prepareParameters(Object... pars) {
    int indexCnt = 0;

    updateCount = (Integer) pars[indexCnt++];
    updateItemId = new int[updateCount];
    priceRaise = new double[updateCount];

    for (int i = 0; i < updateCount; i++) {
        updateItemId[i] = (Integer) pars[indexCnt++];
        priceRaise[i] = (Double) pars[indexCnt++];
    }
}
```

```java
@Override
public Schema getResultSetSchema() {
    Schema sch = new Schema();
    sch.addField(fldName:"update_count", Type.INTEGER);
    return sch;
}

@Override
public SpResultRecord newResultSetRecord() {
    SpResultRecord rec = new SpResultRecord();
    rec.setVal(fldName:"update_count", new IntegerConstant(updateCount));
    return rec;
}
```

- org.vanilladb.bench.benchmarks.as2.rte.jdbc.UpdatePriceJdbcJob
  execute():
  - 用Update price的param helper處理之前param gen產生的param
  - create statement
  - 利用param helper取得每次要更新的i_id和raise value
  - 和db要目前的i_price並算出update後的price為何
  - 將該筆資料寫回db
  - 更新完所有item後commit此次transaction

2. Screenshot of CSV Report

```
time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms), 25th_lat(ms), median_lat(ms), 75th_lat(ms)
65,164,12,5,31,9,11,13
70,345,11,5,39,9,11,12
75,505,11,5,39,8,10,13
80,664,12,5,39,8,10,13
85,794,12,5,39,8,10,14
90,885,13,5,39,8,11,16
95,1036,13,5,39,8,11,16
100,1228,13,5,51,8,10,15
105,1422,12,5,51,8,10,14
110,1582,12,5,51,8,10,14
115,1696,12,5,51,8,10,15
120,1848,12,5,60,8,10,15
```

3. Experiments

- Environment:
  11th Gen Intel(R) Core(TM) i5-1135G7@ 2.40GHz,16GB RAM,512GB NVMe INTEL,Windows11
- Performance Comparison
  - 90%READ_ITEM 10%UPDATE_PRICE

jdbc

time(sec)	throughpu	avg_laten	min(ms)	max(ms)	25th_lat(r	median_la	75th_lat(n
65	365	5	4	14	4	5	5
70	726	5	4	16	4	5	5
75	1098	5	4	17	4	5	5
80	1464	5	4	17	4	5	5
85	1831	5	4	17	4	5	5
90	2188	5	4	18	4	5	5
95	2550	5	4	18	4	5	5
100	2910	5	4	19	4	5	5
105	3263	5	4	19	4	5	5
110	3599	5	4	19	4	5	5
115	3934	5	4	19	4	5	5
120	4263	5	4	19	4	5	5

sp

time(sec)	throughput(txs)	avg_latency(ms)	min(ms)	max(ms)	25th_lat(ms)	median_lat(ms)	75th_lat(ms)
65	3922	0	0	6	0	0	0
70	8290	0	0	19	0	0	0
75	12362	0	0	19	0	0	0
80	16974	0	0	19	0	0	0
85	20764	0	0	19	0	0	0
90	24792	0	0	30	0	0	0
95	29440	0	0	30	0	0	0
100	33227	0	0	30	0	0	0
105	36746	0	0	30	0	0	0
110	40458	0	0	30	0	0	0
115	43615	0	0	30	0	0	0
120	46746	0	0	30	0	0	0

- 50%READ_ITEM 50%UPDATE_PRICE

jdbc

time(sec)	throughpu	avg_laten	min(ms)	max(ms)	25th_lat(r	median_la	75th_lat(ms)
65	312	6	4	49	5	6	6
70	624	6	4	49	5	6	6
75	929	6	4	49	5	6	7
80	1221	6	4	49	5	6	7
85	1506	6	4	49	5	6	7
90	1804	6	4	49	5	6	7
95	2081	6	4	49	5	6	7
100	2373	6	4	49	5	6	7
105	2639	6	4	49	5	6	7
110	2865	6	4	49	5	6	7
115	3083	7	4	49	5	6	7

sp

time(sec)	throughpu	avg_laten	min(ms)	max(ms)	25th_lat(r	median_la	75th_lat(ms)
65	3357	0	0	16	0	0	0
70	6859	0	0	27	0	0	0
75	10467	0	0	27	0	0	0
80	14090	0	0	27	0	0	0
85	17709	0	0	27	0	0	0
90	21288	0	0	27	0	0	0
95	24358	0	0	27	0	0	0
100	27485	0	0	27	0	0	0
105	30623	0	0	27	0	0	0
110	33869	0	0	27	0	0	0
115	37144	0	0	27	0	0	0
120	40442	0	0	27	0	0	0

- 0%READ_ITEM 100%UPDATE_PRICE

jdbc

time(sec)	throughpu	avg_laten	min(ms)	max(ms)	25th_lat(r	median_la	75th_lat(ms)
65	231	8	6	31	7	8	9
70	460	8	6	31	7	8	9
75	704	8	6	31	7	8	9
80	937	8	6	31	7	8	9
85	1143	8	6	35	7	8	9
90	1336	8	6	35	7	8	9
95	1544	9	6	35	7	8	9
100	1777	8	6	35	7	8	9
105	2032	8	6	35	7	8	9
110	2290	8	6	35	7	8	9
115	2544	8	6	35	7	8	9

sp

time(sec)	throughpu	avg_laten	min(ms)	max(ms)	25th_lat(r	median_la	75th_lat(ms)
65	2868	0	0	3	0	0	0
70	5709	0	0	4	0	0	0
75	8582	0	0	4	0	0	0
80	11352	0	0	5	0	0	0
85	14223	0	0	7	0	0	0
90	17228	0	0	7	0	0	0
95	20004	0	0	7	0	0	0
100	22811	0	0	7	0	0	0
105	25747	0	0	7	0	0	0
110	28605	0	0	7	0	0	0
115	31545	0	0	7	0	0	0
120	34657	0	0	7	0	0	0

- Analysis and Explanation
  - JDBC vs stored procedure

    stored procedure的throughput一向都高出jdbc十幾倍，且latency也比jdbc低很多。那是因為jdbc相較之下，多出許多資料處理。
    - 需要連線到server
    - 在執行executeQuery時，呼叫 JdbcStatement::executeQuery再呼叫 RemoteStatementImpl::executeQuery 取得transcation後才呼叫 VanillaDb::newPlanner 取得 planner 後呼叫 Planner::createQueryPlan再包成JdbcResultSet 傳送回去(store procedure則是直接呼叫 VanillaDb::newPlanner 取得 planner 後呼叫 Planner::createQueryPlan，再呼叫 Plan::openVanillaDb::newPlanner 直接完成)

  - READ_WRITE_TX_RATE
    - 當UPDATE_PRICE比例從10%提高到100%，throught會逐漸降低，avg_latency略為提高，原因是因為UPDATE_PRICE比READ_ITEM多做了更新，導致執行時間延長，throughtput下降。