TPCH Benchmark

Harish Butani

August 28, 2015

We ran a benchmark on a set of representative queries that contrast performance of queries being rewritten to use a DruidIndex vs. running the Queries directly against the **raw event** DataSet. We have attempted to make this test as fair as possible: **by not taking any advantages of preaggregations and column pruning in Druid, and by using in memory caching when the queries run only in Spark**. We ran the 2 scenarios on the same cluster: for the Druid run we gave the cluster node resources to Druid History servers, for the Spark run we ran Spark Executors on the worker nodes.

For the benchmark we used the TPCH benchmark dataset, datascale 10G. We converted the 10G star schema into a flattened(denormalized) transaction dataset. For Spark we further processed the data to setup a Partitioned table, stored in Parquet format; the table is partitioned by month. The Druid Index was created using the HadoopDruidIndexer.

The dataset sizes are:

TPCH Flat TSV	46.80GB
Druid Index in HDFS	17.04GB
TPCH Flat Parquet	11.38GB
TPCH Flat Parquet Partition by Month	11.56GB

Queries

We chose the queries to test the performance differences for *time intervals*, *dimension filters*, *time slice aggregation*, and *dimensional aggregations*. We have tested for TPCH queries Q1, Q3, Q5, Q7 and Q8. These queries have been altered so that they can be rewritten as Druid queries. As explained below, these don't change overall benchmark conclusion, if anything the numbers will skew further in favor of Druid once we start to push Sorts and Limits to Druid.

Alterations made to queries:

- Formulas in aggregations have been removed. For e.g. `sum(l_extendedprice*(1-l_discount))` is simply written as `sum(extendedprice)`. This is because we haven't implemented the rewrite to handle aggregation formulas. This has a minor impact on query performance; so the overall benchmark analysis will not change.
- Since we haven't implemented the rewrites, we have removed **order by, limit** clauses from the queries. Again this shouldn't change the overall benchmark conclusion; in fact once we can push down order and limit to Druid, the rewritten query times will reduce.

We have summarized the Query forms below. Appendix A: Query Details describes the Queries in detail.

Query Summary:

Query	Interval	Filters	Group By	Aggregations
Basic Aggregation.	None	None	ReturnFlag LineStatus	Count(*) Sum(exdPrice) Avg(avlQty)
Ship Date Range	1995-12/1997-09	None	ReturnFlag LineStatus	Count(*)
SubQry nation, pType ShpDt Range	1995-12/1997-09	P_Type S_Nation + C_Nation	S_Nation	Count(*) Sum(exdPrice) Max(sCost) Avg(avlQty) Count(Distinct oKey)
TPCH Q1	None	None	ReturnFlag LineStatus	Count(*) Sum(exdPrice) Max(sCost) Avg(avlQty) Count(Distinct oKey)
TPCH Q3	1995-03-15-	O_Date MktSegment	OKey ODate ShipPri	Sum(exdPrice)
TPCH Q5	None	O_Date Region	S_Nation	Sum(exdPrice)
TPCH Q7	None	S_Nation + C_Nation	S_Nation C_Nation ShipDate.Year	Sum(exdPrice)
TPCH Q8	None	Region Type O_Date	ODate.Year	Sum(exdPrice)

The Queries are described in detail in the Query Details section.

Benchmark Results

Appendix B: Benchmark Environment and Appendix C: Running the Benchmark describe in detail where and how the Benchmark was run. In this section we present present our findings.

TPCH 10G dataset

Running against raw event Dataset:

Query	Avg. Time	Min. Time	Max. Time
Basic Aggregation	273371.000	78884	681879
Ship Date Range	221933.000	65074	503307
SubQuery + filters + ShpDt Range	86999.000	13762	417972
TPCH Q1	183568.000	61652	380407
TPCH Q3	270054.000	30141	522393
TPCH Q5	172155.000	50255	292185
TPCH Q7	70663.000	17134	300033
TPCH Q8	19823.000	12287	38247

Running against Druid:

Query	Avg. Time	Min. Time	Max. Time
Basic Aggregation	20324.000	19873	20776
Ship Date Range	1768.000	1712	1824
SubQuery + filters + ShpDt Range	244.000	195	294
TPCH Q1	18340.000	17783	18897
TPCH Q3	10669.000	10345	10994
TPCH Q5	16722.000	16617	16828
TPCH Q7	862.000	712	1012
TPCH Q8	20429.000	20190	20669

Observations about the performance:

- The *SubQuery + filters + ShpDt Range* query gives the most benefit when rewritten to Druid. This is not surprising, as this Query is tailor-made for Druid. The **Interval** and **Dimension** predicates fully leverage the Segment pruning and inverted index layout of Druid.

- The *Ship Date Range* query also shows a significant boost when run on Druid. This is surprising since Spark partition pruning should have nullified the segment pruning Druid will do. We ensure that Spark is doing partition pruning, by explicitly adding *shipYear* predicates to Spark. Is this attributable to aggregations being quicker in Druid, or is this the overhead of predicate evaluation that Spark has to do for each row? Another aspect maybe that we didn't enable the Code Generation option for Catalyst expressions, doing so may bridge this gap.

- *TPCH Q7* shows a significant performance boost when running on Druid: milliSeconds vs. 10s of seconds. The difference is in applying the filter to each row when running on Spark vs. using the inverted index to quickly find what values to aggregate. The **year** level aggregation should have the same optimizations in play: Map side aggregation in the case of Spark, should significantly reduce the amount of rows being shuffled; and in the case of Druid the amount of data shipped to the Broker is small, because only year level aggregations are being shipped. **So queries with just dimensional predicates, and small output cardinality also get a significant boost when rewritten as Druid queries**.

- For *TPCH Q3*, *TPCH Q5* and *TPCH Q8* there is an improvement, but nowhere close to the boost for **Q7**. **This is because the OrderDate predicate is translated to a JavascriptFilter**. We need to look into ways of translating such predicates to a native java function.
- Queries *Basic Agg.* and *TPCH Q1* show some improvement. The Count-Distinct is translated to a Cardinality Aggregator which is an approximate count. This is definitely an advantage for Druid, for large cardinality dimensions. We need to test against Spark using the HyperLogLog aggregator.

We have tried to make the comparisons as fair as possible: by indexing at the lowest grain, and making the Druid index contain all columns. We also partitioned and cached the Spark DataFrame. Inspite of these steps, we find that a **Basic Aggregation** computation(even with the Count-Distinct removed) is faster in Druid. Part of the issue is that Spark needs more memory and compute resources then Druid; several Query runs had failures, causing Executors to crash and hence tasks to be redone. But the differences are not all explained by this, even if we consider minimum times from the Spark runs, Druid still performs better. They could be explained by the fact that we didn't run with Code Generation enabled. We plan to run a followup test with Code Generation turned on. We expect that for *Basic Agg.* and *TPCH Q1* Spark to perform as well as Druid. But for Queries involving **Dimensional** and **Interval** slices, Druid is still an order of magnitude faster(a seconds vs. 10 seconds). Building native functions in Druid will expand the queries that gain this kind of performance boost. Queries that have **Interval** and **Dimensional** predicates are very common in OLAP, as analysts browse the Cube applying filters, Drilling Down and Up.

TPCH Slice dataset

Running against raw event Dataset:

Query	Avg. Time	Min. Time	Max. Time
Basic Aggregation	26890.000	7165	113328
Ship Date Range	10530.000	8499	16380
SubQuery + filters + ShpDt Range	4544.000	3626	7625
TPCH Q1	7993.000	7186	10483
TPCH Q3	6004.000	3727	11246
TPCH Q5	6614.000	5481	8979
TPCH Q7	5648.000	4524	7133
TPCH Q8	3804.000	3211	4780
TPCH Q10	27150.000	23269	35494

Running against Druid:

Query	Avg. Time	Min. Time	Max. Time
Basic Aggregation	3138.000	2327	4142
Ship Date Range	633.000	474	999
SubQuery + filters + ShpDt Range	284.000	151	828
TPCH Q1	2222.000	2005	2429
TPCH Q3	2066.000	910	7176
TPCH Q5	4514.000	4129	5501
TPCH Q7	538.000	318	1282
TPCH Q8	4922.000	4541	5424
TPCH Q10	13765.000	12808	17560

Appendix A: Query Details

Query Basic Aggregation

```
SELECT l_returnflag,
       l_linestatus,
       Count(*),
       Sum(l_extendedprice) AS s,
       Max(ps_supplycost)   AS m,
       Avg(ps_availqty)     AS a,
       Count(DISTINCT o_orderkey)
FROM   orderlineitempartsupplier
GROUP  BY l_returnflag,
          l_linestatus
```

Logical Plan

```
Aggregate [l_returnflag#69,l_linestatus#70], [l_returnflag#69,l_linestatus#70,...
 Project [l_extendedprice#66,o_orderkey#53,ps_supplycost#81,l_returnflag#69,...
  Relation[o_orderkey#53,o_custkey#54,o_orderstatus#55,o_totalprice#56,o_orderdate#57,...
```

Physical Plan

```
Project [l_returnflag#69,l_linestatus#70,alias-1#155L AS c2#109L,alias-2#154 AS s#106,...
 PhysicalRDD [alias-2#154,alias-3#158,alias-7#157,..], DruidRDD[6] at RDD at DruidRDD.scala:34
```

The Druid Query for this query is here

Query Ship Date Range

```

SELECT f,
       s,
       Count(*) AS count_order
FROM   (SELECT l_returnflag AS f,
               l_linestatus AS s,
               l_shipdate,
               s_region,
               s_nation,
               c_nation
        FROM   orderlineitempartsupplier) t
WHERE  Dateisbeforeorequal(Datetime('l_shipdate'),
           Dateminus(Datetime("1997-12-01"), Period("p90d")))
       AND Dateisafter(Datetime('l_shipdate'), Datetime("1995-12-01"))
GROUP  BY f,
          s
```

Logical Plan

```
Aggregate [f#127,s#128], [f#127,s#128,COUNT(1) AS count_order#120L]
 Project [l_returnflag#69 AS f#127,l_linestatus#70 AS s#128]
  Filter (scalaUDF(scalaUDF(l_shipdate#71),scalaUDF(scalaUDF(1997-12-01),scalaUDF(P90D))))...
   Relation[o_orderkey#53,o_custkey#54,o_orderstatus#55,o_totalprice#56,o_orderdate#57,...
```

Physical Plan

```
Project [f#127,s#128,alias-1#159L AS count_order#120L]
 PhysicalRDD [f#127,s#128,alias-1#159L], DruidRDD[7] at RDD at DruidRDD.scala:34
```

The Druid Query for this query is here

Query SubQuery + nation,Type predicates + ShipDate Range

```
SELECT s_nation,
       Count(*)                AS count_order,
       Sum(l_extendedprice)    AS s,
       Max(ps_supplycost)      AS m,
       Avg(ps_availqty)        AS a,
       Count(DISTINCT o_orderkey)
FROM   (SELECT l_returnflag AS f,
               l_linestatus AS s,
               l_shipdate,
               s_region,
               s_nation,
               c_nation,
               p_type,
               l_extendedprice,
               ps_supplycost,
               ps_availqty,
               o_orderkey
        FROM   orderlineitempartsupplier
        WHERE  p_type = 'ECONOMY ANODIZED STEEL') t
WHERE  Dateisbeforeorequal(Datetime('l_shipdate'),
           Dateminus(Datetime("1997-12-01"), Period("p90d")))
       AND Dateisafter(Datetime('l_shipdate'), Datetime("1995-12-01"))
       AND ( ( s_nation = 'FRANCE'
               AND c_nation = 'GERMANY' )
             OR ( c_nation = 'FRANCE'
                  AND s_nation = 'GERMANY' ) )
GROUP  BY s_nation
```

Logical Plan

```
Aggregate [s_nation#88], [s_nation#88,COUNT(1) AS count_order#129L,SUM(l_extendedprice#66) AS ...
 Project [l_extendedprice#66,o_orderkey#53,ps_supplycost#81,s_nation#88,ps_availqty#80]
  Filter ((p_type#93 = ECONOMY ANODIZED STEEL) && ((scalaUDF(scalaUDF(l_shipdate#71)...
   Relation[o_orderkey#53,o_custkey#54,o_orderstatus#55,o_totalprice#56,o_orderdate#57,...
```

Physical Plan

```
Project [s_nation#88,alias-1#161L AS count_order#129L,alias-2#160 AS s#130,...
 PhysicalRDD [alias-2#160,alias-3#164,...], DruidRDD[8] at RDD at DruidRDD.scala:34
```

The Druid Query for this query is here

Query TPCH Q1

```
SELECT l_returnflag,
       l_linestatus,
       Count(*),
       Sum(l_extendedprice) AS s,
       Max(ps_supplycost)   AS m,
       Avg(ps_availqty)     AS a,
       Count(DISTINCT o_orderkey)
```

```
FROM    orderlineitempartsupplier
GROUP   BY l_returnflag,
           l_linestatus
```

Logical Plan

```
Aggregate [l_returnflag#69,l_linestatus#70], [l_returnflag#69,l_linestatus#70,COUNT(1) AS c2#145L,...
 Project [l_extendedprice#66,o_orderkey#53,ps_supplycost#81,l_returnflag#69,l_linestatus#70,...
  Relation[o_orderkey#53,o_custkey#54,o_orderstatus#55,o_totalprice#56,o_orderdate#57,....
```

Physical Plan

```
Project [l_returnflag#69,l_linestatus#70,alias-1#166L AS c2#145L,alias-2#165 AS s#142,...
 PhysicalRDD [alias-2#165,alias-3#169,...], DruidRDD[9] at RDD at DruidRDD.scala:34
```

The Druid Query for this query is here

Query TPCH Q3 - extendePrice instead of revenue; order, limit removed

```
SELECT o_orderkey,
       Sum(l_extendedprice) AS price,
       o_orderdate,
       o_shippriority
FROM    orderlineitempartsupplier
WHERE   c_mktsegment = 'BUILDING'
       AND Dateisbefore(Datetime(`o_orderdate`), Datetime("1995-03-15"))
       AND Dateisafter(Datetime(`l_shipdate`), Datetime("1995-03-15"))
GROUP   BY o_orderkey,
           o_orderdate,
           o_shippriority
```

Logical Plan

```
Aggregate [o_orderkey#53,o_orderdate#57,o_shippriority#60], [o_orderkey#53,...
 Project [o_orderkey#53,o_orderdate#57,o_shippriority#60,l_extendedprice#66]
  Filter ((((c_mktsegment#102 = BUILDING) && scalaUDF(scalaUDF(o_orderdate#57),...
   Relation[o_orderkey#53,o_custkey#54,o_orderstatus#55,o_totalprice#56,...
```

Physical Plan

```
Project [CAST(o_orderkey#53, IntegerType) AS o_orderkey#53,alias-1#170 AS ...
 PhysicalRDD [o_orderkey#53,...], DruidRDD[10] at RDD at DruidRDD.scala:34
```

The Druid Query for this query is here

Query TPCH Q5 - extendePrice instead of revenue

```
SELECT s_nation,
       Sum(l_extendedprice) AS extendedPrice
FROM    orderlineitempartsupplier
WHERE   s_region = 'ASIA'
       AND Dateisafterorequal(Datetime(`o_orderdate`), Datetime("1994-01-01"))
       AND Dateisbefore(Datetime(`o_orderdate`),
           Dateplus(Datetime("1994-01-01"), Period(
               "p1y")))
GROUP   BY s_nation
```

Logical Plan

```
Aggregate [s_nation#88], [s_nation#88,SUM(l_extendedprice#66) AS extendedPrice#148]
 Project [s_nation#88,l_extendedprice#66]
  Filter ((((s_region#89 = ASIA) && scalaUDF(scalaUDF(o_orderdate#57),scalaUDF(1994-01-01))) ...
   Relation[o_orderkey#53,o_custkey#54,o_orderstatus#55,o_totalprice#56,o_orderdate#57,....
```

Physical Plan

```
Project [s_nation#88,alias-1#171 AS extendedPrice#148]
 PhysicalRDD [s_nation#88,alias-1#171], DruidRDD[11] at RDD at DruidRDD.scala:34
```

The Druid Query for this query is here

Query TPCH Q7 - price instead of revenue

```
SELECT s_nation,
       c_nation,
       Year(Datetime(`l_shipdate`)) AS l_year,
       Sum(l_extendedprice)         AS extendedPrice
FROM   orderlineitempartsupplier
WHERE  ( ( s_nation = 'FRANCE'
           AND c_nation = 'GERMANY' )
         OR ( c_nation = 'FRANCE'
              AND s_nation = 'GERMANY' ) )
GROUP  BY s_nation,
          c_nation,
          Year(Datetime(`l_shipdate`))
```

Logical Plan

```
Aggregate [s_nation#88,c_nation#104,scalaUDF(scalaUDF(l_shipdate#71))], ...
 Project [s_nation#88,c_nation#104,l_shipdate#71,l_extendedprice#66]
  Filter ((((s_nation#88 = FRANCE) && (c_nation#104 = GERMANY)) || ...
   Relation[o_orderkey#53,o_custkey#54,o_orderstatus#55,o_totalprice#56,o_orderdate#57,....
```

Physical Plan

```
Project [s_nation#88,c_nation#104,CAST(l_shipdate#172, IntegerType) AS...
 PhysicalRDD [s_nation#88,...], DruidRDD[12] at RDD at DruidRDD.scala:34
```

The Druid Query for this query is here

Query TPCH Q8 - extendedPrice instead of market share

```
SELECT Year(Datetime(`o_orderdate`)) AS o_year,
       Sum(l_extendedprice)          AS price
FROM   orderlineitempartsupplier
WHERE  c_region = 'AMERICA'
       AND p_type = 'ECONOMY ANODIZED STEEL'
       AND Dateisafterorequal(Datetime(`o_orderdate`), Datetime("1995-01-01"))
       AND Dateisbeforeorequal(Datetime(`o_orderdate`), Datetime("1996-12-31"))
GROUP  BY Year(Datetime(`o_orderdate`))
```

Logical Plan

```
Aggregate [scalaUDF(scalaUDF(o_orderdate#57))], [scalaUDF(scalaUDF(o_orderdate#57)) AS ...
 Project [o_orderdate#57,l_extendedprice#66]
  Filter ((((c_region#105 = AMERICA) && (p_type#93 = ECONOMY ANODIZED STEEL)) && ...
   Relation[o_orderkey#53,o_custkey#54,o_orderstatus#55,o_totalprice#56,o_orderdate#57,...
```

Physical Plan

```
Project [CAST(o_orderdate#174, IntegerType) AS o_year#151,alias-1#175 AS price#152]
 PhysicalRDD [o_orderdate#174,alias-1#175], DruidRDD[13] at RDD at DruidRDD.scala:34
```

The Druid Query for this query is here

Appendix B: Benchmark Environment

Cluster Details

The Benchmark was run on a 4 node cluster. Each node is a 2 core,16GB memory, 256GB hard drive machine running centos 6.4. The output of the `lscpu` and `hdparm` are listed below:

Listing 1: Machine Details

```

lscpu

Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             2
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 42
Stepping:              1
CPU MHz:               1999.999
BogoMIPS:              3999.99
Virtualization:        VT-x
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
NUMA node0 CPU(s):     0,1

sudo hdparm -tT /dev/vdb

/dev/vdb:
 Timing cached reads:   12798 MB in  2.00 seconds = 6408.97 MB/sec
 Timing buffered disk reads: 540 MB in  3.00 seconds = 179.98 MB/sec
```

The machines are setup with HDP 2.3 using Ambari. Also installed Druid 0.8 on the machines. The cluster is configured to use Yarn; we installed and setup Spark 1.4.1 to run using the Yarn Resource Manager.

TPCH Flattened Dataset, scale 10

For the benchmark we used the TPCH benchmark dataset, datascale 10G. We converted the 10G star schema into a flattened(denormalized) transaction dataset using a tool we wrote TpchGenFlattenedData, for example we ran it like this:

```
spark/bin/spark-submit -num-executors 7 \
-properties-file spark-druid/spark.properties \
-packages com.databricks:spark-csv2.10:1.1.0 \
-jars spark-druid/spark-datetime-assembly-0.0.1.jar,\
      spark-druid/spark-druid-olap-assembly-0.0.1.jar \
-class org.sparklinedata.tpch.hadoop.TpchGenFlattenedData \
spark-druid/tpchdata-assembly-0.0.1.jar \
tpchflatorc10 tpchflattened
```

Dataset for Spark Queries

For spark we further processed the data to setup a Partitioned table, stored in Parquet format; the table is partitioned by month. We use the TpchBuildParquetPartitioned to do this.

Druid Index for TPCH Flattened Dataset

The Druid Index was created using the HadoopDruidIndexer with the following command:

```
java -Xmx256m -Dhdp.version=2.3.0.0-2557 -Duser.timezone=UTC \
-Dfile.encoding=UTF-8 -classpath \
$DIR/config/_common:$HADOOP_CONF_DIR:$DIR/lib/* \
io.druid.cli.Main index hadoop <spec_file>
```

See Druid TPCH Index Specification for detailed specification of the TPCH index in Druid. Key points of the Index:

- *l_shipdate* is chosen as the time dimension. Based on the TPCH Query set, there is a significant number of queries that are time sliced based on the Ship Date.
- We indexed all the dimensions. The metrics are: ~o_totalprice,~ `l_quantity, l_extendedprice, ps_availqty, ps_supplycost, c_acctbal`. Rest of the columns are modeled as dimensions.
- **The index is created at the grain of raw events.**
- The Index time segment is chosen to be month.

Note by choosing to model all dimensions and by choosing to index at the grain of events, we have made the Druid Index as big as possible. **We are note giving Druid any advantages of preaggregation or column pruning.**

DataSource setup

The raw event DataSource and Druid datasource are defined in the following way:

Listing 2: Raw Event DataSource

```
1
2  // parquet based partitioned table
3  val df = sqlCtx.read.parquet(cfg.tpchFlatDir)
4  df.cache()
5  df.registerTempTable("orderLineItemPartSupplier")
6
7  // Druid Datasource
8  CREATE TEMPORARY TABLE orderLineItemPartSupplier
9        USING org.sparklinedata.druid
10       OPTIONS (sourceDataframe "$baseFlatTableName",
11       timeDimensionColumn "l_shipdate",
12       druidDatasource "tpch",
13       druidHost "${cfg.druidBroker}",
14       druidPort "8082");
```

Appendix C: Running the Benchmark

Running against Druid Datasource

For the Druid Datasource experiment the queries are run on spark using the Druid TpchBenchMark tool. It is run using the following command:

Listing 3: Running Tpchbenchmark on Druid Datasource

```
1
2  ~/spark-1.4.1-bin-hadoop2.6/bin/spark-submit \
3  --properties-file spark.properties \
4  --packages com.databricks:spark-csv_2.10:1.1.0 \
5  --jars sparkjars/spark-datetime-assembly-0.0.1.jar  \
6  --class org.sparklinedata.druid.tools.TpchBenchMark \
7  sparkjars/spark-druid-olap-assembly-0.0.1.jar \
8  -n hb-1.openstacklocal \
9  -t tpchFlattenedData_10/orderLineItemPartSupplierCustomer \
10 -d hb-1.openstacklocal
```

The cluster is setup to run a historical server on each node. Each historical server is configure with 8GB of memory:

```
JAVA_HISTORICAL_OPTIONS="-server \
 -Xmx8g \
 -Xms8g \
 -XX:NewSize=1g \
 -XX:MaxNewSize=1g \
 -XX:MaxDirectMemorySize=10g \
 -XX:+UseConcMarkSweepGC \
 -XX:+PrintGCDetails \
 -XX:+PrintGCTimeStamps \
 -XX:+HeapDumpOnOutOfMemoryError \
 -Duser.timezone=UTC \
 -Dfile.encoding=UTF-8"
```

The spark shell is run in local mode on one of the nodes, so that Spark uses as little cluster resources as possible.

Running against cached Spark DataFrame

We compare the rewritten queries against the case of not having a Druid Index. In this case we try to give the Spark engine as much advantage as we can.

We give the Spark executors as much of the Yarn cluster as possible. The Spark configuration is:

```
spark.serializer=org.apache.spark.serializer.KryoSerializer
#spark.sql.autoBroadcastJoinThreshold=100000000
spark.sql.autoBroadcastJoinThreshold=-1
spark.sql.planner.externalSort=true

spark.executor.memory=9g
spark.driver.memory=2g
#spark.executor.cores=2
```

As part of the initialization, the orderLineItemPartSupplier DataFrame is cached in memory.

For the queries going against Spark we used the Spark TpchBenchmark tool. It is run with the following command:

Listing 4: Running the Benchmark, on the Raw Event DataFrame

```
1
2  ~/spark-1.4.1-bin-hadoop2.6/bin/spark-submit \
3  --properties-file spark.properties \
4  --packages com.databricks:spark-csv_2.10:1.1.0 \
5  --jars sparkjars/spark-datetime-assembly-0.0.1.jar,\
6         sparkjars/spark-druid-olap-assembly-0.0.1.jar,\
7         sparkjars/tpchdata-assembly-0.0.1.jar    \
8  --num-executors 4 --master yarn-client \
9  --class org.sparklinedata.tpch.hadoop.TpchParquetBenchmark \
10 sparkjars/tpchdata-assembly-0.0.1.jar \
11 -t tpchFlattenedData_10/\
12 orderLineItemPartSupplierCustomer.parquet.partitioned
```

Appendix D: Query Results

Raw Event Dataset Query Results

```
Basic Aggregation
[A,F,1478160,5.653926919402009E10,1000.0,4997.955256535151,644914]
[R,F,1480195,5.6579322994170044E10,1000.0,4997.424184651347,645781]
[N,F,38767,1.4841356872899961E9,999.96,4990.055330564655,30750]
[N,O,2998603,1.1461753710263998E11,1000.0,4999.978290890792,768912]
Ship Date Range
[N,O,1599288]
SubQuery + nation,Type predicates + ShipDate Range
[FRANCE,16,687985.3700000001,933.5,5169.0,16]
[GERMANY,19,753095.0299999999,994.08,5400.421052631579,19]
TPCH Q1
[A,F,1478160,5.653926919402008E10,1000.0,4997.955256535151,644914]
[R,F,1480195,5.657932299417004E10,1000.0,4997.424184651347,645781]
[N,F,38767,1.4841356872899961E9,999.96,4990.055330564655,30750]
[N,O,2998603,1.1461753710263995E11,1000.0,4999.978290890792,768912]
TPCH Q3 - extendePrice instead of revenue
[33059171,432867.97000000003,1995-02-16T00:00:00.000Z,0]
[20524164,431359.91000000003,1995-03-04T00:00:00.000Z,0]
[1083941,415404.73,1995-02-21T00:00:00.000Z,0]
[16341859,409805.29,1995-02-25T00:00:00.000Z,0]
[31374434,409371.3,1995-02-17T00:00:00.000Z,0]
[34405031,402898.7,1995-03-09T00:00:00.000Z,0]
[56594855,400094.6,1995-03-13T00:00:00.000Z,0]
[2452422,394861.0,1995-02-22T00:00:00.000Z,0]
[32884775,391267.39,1995-02-18T00:00:00.000Z,0]
[26900320,390404.95999999996,1995-03-12T00:00:00.000Z,0]
TPCH Q5 - extendePrice instead of revenue
[INDIA,1.4134655952000005E9]
[JAPAN,1.4034246889200006E9]
[VIETNAM,1.3858049337799993E9]
[CHINA,1.3806778567499995E9]
[INDONESIA,1.3690848978599997E9]
TPCH Q7 - price instead of revenue
[FRANCE,GERMANY,1992,4.873365616E7]
```

```
[FRANCE,GERMANY,1993,5.3386908010000005E7]
[FRANCE,GERMANY,1994,5.6118161500000015E7]
[FRANCE,GERMANY,1995,5.6177264720000006E7]
[FRANCE,GERMANY,1996,5.411796232E7]
[FRANCE,GERMANY,1997,5.697326894E7]
[FRANCE,GERMANY,1998,4.195966787E7]
[GERMANY,FRANCE,1992,4.596645559E7]
[GERMANY,FRANCE,1993,5.7467286599999994E7]
[GERMANY,FRANCE,1994,5.9568380260000005E7]
[GERMANY,FRANCE,1995,5.574328572E7]
[GERMANY,FRANCE,1996,5.817071533000001E7]
[GERMANY,FRANCE,1997,5.685662362000001E7]
[GERMANY,FRANCE,1998,4.0363215349999994E7]
TPCH Q8 - extendePrice instead of market share
[1995,4.372841152E7]
[1996,4.7175562359999997E7]
TPCH Q10 - extendePrice instead of revenue, no group by on acctBal (first 20 rows)
[Customer#001485241,UNITED KINGDOM,Fq07MLElZBC54DXzVq9YbP2WP,t,33-116-170-5647,al accounts
[Customer#001486201,JORDAN,TJhgKueFwrrtXLtenhlw sC2N,23-611-797-4750,ag. deposits along th
[Customer#000663001,RUSSIA,18LlI2l6hGpiVkn,32-998-943-3573,ests nag above the accounts. ca
[Customer#000361891,SAUDI ARABIA,wKPyClaA8FXzVmOE7OH68Cn ujxyP,30-105-377-1699,y unusual f
[Customer#000846871,CANADA,qBgCdbbiT0dPMHXaW3ejfzyDhJlf9I3UdrlvG,13-818-536-4472,r, blithe
[Customer#001477261,ETHIOPIA,Ug4Chh6HgFuFuH4kKuDX,y,15-477-193-2424,ticingly around the fu
[Customer#000996901,UNITED STATES,vspZ5Sp5c5 Z5vDpHIqYXj lNylKNYdf,Hn,34-278-198-3024,its.
[Customer#000109531,MOROCCO,ctw,V3Lg WsnSF,25-806-287-6640, frets. special packages sleep
[Customer#000116371,MOROCCO,MnsTThR yJf3 VUGbdh2g7Ls,25-461-687-3461,courts nag quickly ac
[Customer#000478261,JORDAN,nleur50a6uwrpHy5M1aUI6YlTJ3GxdvYr,23-344-728-8021, closely amon
[Customer#000822841,KENYA,YegKDa24ghUHejhD9GUgL6GNpToTlaKD8bTBZ,24-127-277-1726, the regul
[Customer#000596851,ARGENTINA,a1DTx1D4ltckAM8,11-470-165-2441,es along the furiously even
[Customer#000225091,IRAN,ESxBAyRn8EwDJKlPkc,DvPHYFsa85MatFyUzscWY,20-157-662-6929,ic, regu
[Customer#001080691,UNITED KINGDOM,TbiBgAVDMQhNHKZOb4qwZUN0tIYhTWGDwQTdym,33-375-446-6539,
[Customer#001160881,RUSSIA,ACbaUek4MwaAm QpcQAtEN7PUjw3FBBElZIThrC,32-316-891-6777,odolite
[Customer#000001711,MOROCCO,Mhg8c9IAFb8G,25-302-946-6337,gle carefully. final, even deposi
[Customer#001474861,VIETNAM,6cg7FtblHmXnMIjqK11pT47Lsx2,31-595-929-5136,ideas sleep furiou
[Customer#001324291,IRAQ,wajEKFPCC6A8Maf450IkC,21-210-951-6699,le furiously blithely unusu
[Customer#000977191,GERMANY,o5XL tB NK8AGE95AuOwL0oz,17-184-695-3349,ully furiously unusua
[Customer#000850441,ROMANIA,Bq7O5tRkwNHqA37,z1nZ2Ngrg,29-645-452-6044,ickly even theodolit
```

Druid Query Results

```
Basic Aggregation
[A,F,1478160,5.6539271168E10,1000.0,4997.955256535151,618896]
[N,F,38767,1.48413568E9,999.9600219726562,4990.055330564655,31436]
[N,O,2998581,1.14616795136E11,1000.0,4999.969491236021,754442]
[R,F,1480195,5.657932288E10,1000.0,4997.424184651347,642387]
Ship Date Range
[N,O,1599288]
SubQuery + nation,Type predicates + ShipDate Range
[FRANCE,16,687985.3671875,933.5,5169.0,16]
[GERMANY,19,753095.0,994.0800170898438,5400.421052631579,19]
```

```
TPCH Q1
[A,F,1478160,5.6539268096E10,1000.0,4997.955256535151,618896]
[N,F,38767,1.48413568E9,999.9600219726562,4990.055330564655,31436]
[N,O,2998581,1.14616793088E11,1000.0,4999.969491236021,754442]
[R,F,1480195,5.6579323904E10,1000.0,4997.424184651347,642387]
TPCH Q3 - extendePrice instead of revenue
[33059171,432867.9765625,1995-02-16T00:00:00.000Z,0]
[20524164,431359.8984375,1995-03-04T00:00:00.000Z,0]
[1083941,415404.73828125,1995-02-21T00:00:00.000Z,0]
[16341859,409805.3046875,1995-02-25T00:00:00.000Z,0]
[31374434,409371.296875,1995-02-17T00:00:00.000Z,0]
[34405031,402898.6953125,1995-03-09T00:00:00.000Z,0]
[56594855,400094.60546875,1995-03-13T00:00:00.000Z,0]
[2452422,394860.984375,1995-02-22T00:00:00.000Z,0]
[32884775,391267.39453125,1995-02-18T00:00:00.000Z,0]
[26900320,390404.97265625,1995-03-12T00:00:00.000Z,0]
TPCH Q5 - extendePrice instead of revenue
[INDIA,1.413465584E9]
[JAPAN,1.403424672E9]
[VIETNAM,1.385804928E9]
[CHINA,1.380677856E9]
[INDONESIA,1.369084912E9]
TPCH Q7 - price instead of revenue
[FRANCE,GERMANY,1992,4.87336565E7]
[FRANCE,GERMANY,1993,5.3386908E7]
[FRANCE,GERMANY,1994,5.6118162E7]
[FRANCE,GERMANY,1995,5.61772655E7]
[FRANCE,GERMANY,1996,5.4117962E7]
[FRANCE,GERMANY,1997,5.6973269E7]
[FRANCE,GERMANY,1998,4.1959668E7]
[GERMANY,FRANCE,1992,4.5966456E7]
[GERMANY,FRANCE,1993,5.7467286E7]
[GERMANY,FRANCE,1994,5.9568382E7]
[GERMANY,FRANCE,1995,5.5743286E7]
[GERMANY,FRANCE,1996,5.8170716E7]
[GERMANY,FRANCE,1997,5.6856623E7]
[GERMANY,FRANCE,1998,4.0363216E7]
TPCH Q8 - extendePrice instead of market share
[1995,4.3728411E7]
[1996,4.71755625E7]
TPCH Q10 - extendePrice instead of revenue, no group by on acctBal (first 20 rows)
[Customer#001485241,UNITED KINGDOM,Fq07MLElZBC54DXzVq9YbP2WP,t,33-116-170-5647,al accounts
[Customer#001486201,JORDAN,TJhgKueFwrrtXLtenhlw sC2N,23-611-797-4750,ag. deposits along th
[Customer#000663001,RUSSIA,18LlI2l6hGpiVkn,32-998-943-3573,ests nag above the accounts. ca
[Customer#000361891,SAUDI ARABIA,wKPyClaA8FXzVmOE7OH68Cn ujxyP,30-105-377-1699,y unusual f
[Customer#000846871,CANADA,qBgCdbbiT0dPMHXaW3ejfzyDhJlf9I3UdrlvG,13-818-536-4472,r, blithe
[Customer#001477261,ETHIOPIA,Ug4Chh6HgFuFuH4kKuDX,y,15-477-193-2424,ticingly around the fu
[Customer#000996901,UNITED STATES,vspZ5Sp5c5 Z5vDpHIqYXj lNylKNYdf,Hn,34-278-198-3024,its.
[Customer#000109531,MOROCCO,ctw,V3Lg WsnSF,25-806-287-6640," frets. special packages sleep
[Customer#000116371,MOROCCO,MnsTThR yJf3 VUGbdh2g7Ls,25-461-687-3461,courts nag quickly ac
```

```
[Customer#000478261,JORDAN,nleur50a6uwrpHy5M1aUI6YlTJ3GxdvYr,23-344-728-8021," closely amo
[Customer#000822841,KENYA,YegKDa24ghUHejhD9GUgL6GNpToTlaKD8bTBZ,24-127-277-1726," the regu
[Customer#000596851,ARGENTINA,a1DTx1D4ltckAM8,11-470-165-2441,"es along the furiously even
[Customer#000225091,IRAN,ESxBAyRn8EwDJKlPkc,DvPHYFsa85MatFyUzscWY,20-157-662-6929,ic, regu
[Customer#001080691,UNITED KINGDOM,TbiBgAVDMQhNHKZOb4qwZUNOtIYhTWGDwQTdym,33-375-446-6539,
[Customer#001160881,RUSSIA,ACbaUek4MwaAm QpcQAtEN7PUjw3FBBElZIThrC,32-316-891-6777,odolite
[Customer#000001711,MOROCCO,Mhg8c9IAFb8G,25-302-946-6337,gle carefully. final, even deposi
[Customer#001474861,VIETNAM,6cg7FtblHmXnMIjqK11pT47Lsx2,31-595-929-5136,"ideas sleep furio
[Customer#001324291,IRAQ,wajEKFPCC6A8Maf450IkC,21-210-951-6699,le furiously blithely unusu
[Customer#000977191,GERMANY,o5XL tB NK8AGE95AuOwL0oz,17-184-695-3349,ully furiously unusua
[Customer#000850441,ROMANIA,Bq7O5tRkwNHqA37,z1nZ2Ngrg,29-645-452-6044,ickly even theodolit
```