Shark: SQL and Rich Analytics at Scale


@andy_pavlo
SQL on MR

• Writing Java code to analyze data sets is akin to CODASYL.

• Alternatives:
  – SQL → MapReduce
  – Pig → MapReduce
Why is SQL on MR slow?

• Intermediate output on disk.
• Inferior data format and layout
• Naïve execution strategies.
• Task scheduling overhead.
Shark Overview

• Data warehouse on top of Spark.
• Scalable + fault-tolerant.
• Low-latency, interactive queries through in-memory computation.
• Compatible with Apache Hive.
Spark Overview

• General purpose distributed computing framework.
• Work with distributed collections as you would with local ones.
• Resilient distributed datasets.
RDDs

• Immutable collections of objects.
• Built through parallel transformations (e.g., map, filter).
• Automatically rebuilt on failure
• Controllable persistence
Shark Architecture

- Client: CLI, JDBC
- Driver
- SQL Parser
- Query Optimizer
- Physical Plan
- Execution

Metastore

Spark

HDFS
Shark Optimizations

• Dynamic Query Optimization.
• Columnar Memory Store.
• Co-partitioning & Co-location.
• Machine Learning Integration.
Query Optimization

• No stats for non-loaded data.
• UDFs are difficult to optimize.

```sql
SELECT * FROM table1 AS t1
JOIN table2 AS t2 ON t1.key = t2.key
WHERE assclownUDF(t1.field, t2.field)=true;
```
Partial DAG Execution

• Gather statistics per partition when materializing outputs.
• Allows for dynamic alternation of query plans based on statistics collected at runtime.
Shuffle Join

Stage 2

Stage 1

Join Result

Map Join

Table 2

Table 1

Join Result
Column Store

• Column-oriented storage using arrays of primitive types.

• Per-partition compression.

Row Storage

<table>
<thead>
<tr>
<th></th>
<th>john</th>
<th>4.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mike</td>
<td>3.5</td>
</tr>
<tr>
<td>2</td>
<td>sally</td>
<td>6.4</td>
</tr>
</tbody>
</table>

Column Storage

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>mike</td>
<td>sally</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>3.5</td>
<td>6.4</td>
<td></td>
</tr>
</tbody>
</table>
Data Co-Partitioning

- Store table partitions together in storage layer using pre-defined join attribute.

```sql
CREATE TABLE ol
AS SELECT * FROM lineitem
DISTRIBUTE BY L_ORDERKEY;

CREATE TABLE o TBLPROPERTIES
("copartition"="ol")
AS SELECT * FROM order
DISTRIBUTE BY O_ORDERKEY;
```
Machine Learning

• Unified system for query processing & machine learning.
• Query processing & ML share the same set of workers and caches.
users = sql2rdd("SELECT * FROM user u
    JOIN comment c ON c.uid=u.uid")

features = users.mapRows { row =>
    new Vector(extractFeat1(row.getInt("age")),
        extractFeat2(row.getStr("name")),
        ...)}

trainedVector = logRegressUDF(features.cache())
Experimental Analysis

• Comparison between Shark and Hive using HDFS.
• Data set from Conviva.
  – 100 EC2 nodes.
  – 1.7 TB of video viewing sessions.
  – Four different queries.
Hive Comparison

<table>
<thead>
<tr>
<th>Runtime (seconds)</th>
<th>Shark (Memory)</th>
<th>Shark (Disk)</th>
<th>Hive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.8</td>
<td>0.7</td>
<td>1.1</td>
</tr>
<tr>
<td>Q2</td>
<td>0.8</td>
<td>0.7</td>
<td>1.0</td>
</tr>
<tr>
<td>Q3</td>
<td>0.8</td>
<td>0.7</td>
<td>1.0</td>
</tr>
<tr>
<td>Q4</td>
<td>0.8</td>
<td>0.7</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Conclusion

• **Shark** is a scalable, fault-tolerant data warehouse.

• Relies on **Spark** for distributed execution, but also provides “big data” optimizations.
Sources

• Reynold Xin (Berkeley)