Shark: Hive on Spark

Prajakta Kalmegh
Duke University
What is Shark?

Port of Apache Hive to run on Spark

Compatible with existing Hive data, metastores, and queries (HiveQL, UDFs, etc)

Similar speedups of up to 40x

Figure 1: Shark Architecture
Motivation

Hive is great, but Hadoop’s execution engine makes even the smallest queries take minutes.

Scala is good for programmers, but many data users only know SQL.

Can we extend Hive to run on Spark?
Hive Architecture

- **Meta store**
- **HDFS**
- **Client**
  - CLI
  - JDBC
- **Driver**
  - SQL Parser
  - Query Optimizer
  - Physical Plan
  - Execution
- **MapReduce**
- **HDFS**
Shark Architecture

- Meta store
- Client
  - CLI
  - JDBC
  - Driver
  - Cache Mgr.
  - SQL Parser
  - Query Optimizer
  - Physical Plan
  - Execution
- Spark
- HDFS
Shark Engine: Extensions to Hive

- PDE (Partial DAG Executions)
  - To Support dynamic query optimization
  - allows dynamic alteration of query plans based on data statistics collected at run-time
  - use PDE to optimize the global structure of the plan at stage boundaries

- Skew Handling and Degree of Parallelism
  - Importance of DoP for Mappers vs Reducers (too few can overload reducers)
  - Skew mitigation: Fine-grained partitions are assigned to coalesced partitions using a greedy bin-packing heuristic

- Distributed Data Loading
  - Loading tasks use the data schema to extract individual fields from rows
  - Marshal a partition of data into its columnar representation
  - Store those columns in memory
Shark Engine: Extensions to Hive

- Join Optimizations

Figure 3: Data flows for map join and shuffle join. Map join broadcasts the small table to all large table partitions, while shuffle join repartitions and shuffles both tables.
Efficient In-Memory Storage

Simply caching Hive records as Java objects is inefficient due to high per-object overhead.

Instead, Shark employs column-oriented storage using arrays of primitive types.

<table>
<thead>
<tr>
<th>Row Storage</th>
<th>Column Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>john</td>
</tr>
<tr>
<td>2</td>
<td>mike</td>
</tr>
<tr>
<td>3</td>
<td>sally</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>mike</td>
<td>sally</td>
</tr>
<tr>
<td>4.1</td>
<td>3.5</td>
<td>6.4</td>
</tr>
</tbody>
</table>
Efficient In-Memory Storage

Simply caching Hive records as Java objects is inefficient due to high per-object overhead.

Instead, Shark employs column-oriented storage using arrays of primitive types.

**Column Storage**

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

**Row Storage**

<table>
<thead>
<tr>
<th>3</th>
<th>Sally</th>
<th>6.4</th>
</tr>
</thead>
</table>

Benefit: similarly compact size to serialized data, but >5x faster to access.
Shark vs Spark SQL
Spark SQL Overview

• Newest component of Spark initially contributed by databricks (< 1 year old)
• Tightly integrated way to work with structured data (tables with rows/columns)
• Transform RDDs using SQL
• Data source integration: Hive, Parquet, JSON, and more
Spark SQL

A General Stack

Spark SQL
Spark Streaming real-time
GraphX graph
MLlib machine learning

Spark
Powerful Stack – Agile Development

---

non-test, non-example source lines

- amplab UC Berkeley
- databricks
Relationship to Shark

Shark modified the Hive backend to run over Spark, but had two challenges:

- Limited integration with Spark programs
- Hive optimizer not designed for Spark

Spark SQL reuses the best parts of Shark:

**Borrows**
- Hive data loading
- In-memory column store

**Adds**
- RDD-aware optimizer
- Rich language interfaces
Adding Schema to RDDs

Spark + RDDs
*Functional* transformations on partitioned collections of *opaque objects.*

SQL + SchemaRDDs
*Declarative* transformations on partitioned collections of *tuples.*
Example Dataset

A text file filled with people’s names and ages:

Michael, 30
Andy, 31
...
RDDS as Relations (Scala)

```scala
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext._

// Define the schema using a case class.
case class Person(name: String, age: Int)

// Create an RDD of Person objects and register it as a table.
val people =
  sc.textFile("examples/src/main/resources/people.txt")
    .map(_.split","))
    .map(p => Person(p(0), p(1).trim.toInt))

people.registerAsTable("people")
```
Querying Using SQL

# SQL can be run over SchemaRDDs that have been registered
# as a table.

teenagers = sqlCtx.sql(""
    SELECT name FROM people WHERE age >= 13 AND age <= 19""

# The results of SQL queries are RDDs and support all the normal
# RDD operations.

teenNames = teenagers.map(lambda p: "Name: " + p.name)
Reading Data Stored in Hive

```python
from pyspark.sql import HiveContext
hiveCtx = HiveContext(sc)

hiveCtx.hql(""
    CREATE TABLE IF NOT EXISTS src (key INT, value STRING)"
")

hiveCtx.hql(""
    LOAD DATA LOCAL INPATH 'examples/.../kv1.txt' INTO TABLE src"
")

# Queries can be expressed in HiveQL.
results = hiveCtx.hql("FROM src SELECT key, value").collect()
```
Performance vs. SHARK

TPC-DS

- Shark
- Spark SQL
References:

IEEE Computer Society, Washington, DC, USA, 25-36


Computing* (HPDC '10). ACM, New York, NY, USA, 810-818

[6] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott 
*Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (NSDI'12). USENIX Association, 
Berkeley, CA, USA, 2-2.
