Shark: SQL and Rich Analytics at Scale

Michael Xueyuan Han
Ronny Hajoon Ko
What Are The Problems?

Data volumes are expanding dramatically

Why Is It Hard?

Needs to scale out → Managing hundreds of machines is hard
What Are The Problems?

Data volumes are expanding dramatically

Increasing incidence of faults and stragglers

Why Is It Hard?

High scale increase possibility of faults and stragglers → Complicate parallel design
What Are The Problems?

Data volumes are expanding dramatically
Increasing incidence of faults and stragglers
Data analysis becomes more complex

Why Is It Hard?

Database systems need to handle statistical methods beyond their simple roll-up, drill-down capability
What Are The Problems?

Data volumes are expanding dramatically
Increasing incidence of faults and stragglers
Data analysis becomes more complex

Why Is It Hard?

Users expect to query at interactive speeds

The increase of scale and complexity makes it hard
Existing Solutions
-Class A-
MapReduce-based Systems
Hive, Dryad, etc.
Why don’t they work?

Although they have:
- Scalability
- Fine-grained fault tolerance
- Allows complex statistical analysis (e.g. ML)

But:
- Slow SQL query processing latency
Existing Solutions
-Class B-
Massive Parallel Processing Databases
Vertica, Teradata, etc.
Why don’t they work?

Although they have:
- Fast parallel SQL query processing

But:
- Not scalable
- Expensive/impossible complex analysis (e.g. UDFS, ML)
- Coarse-grained fault tolerance
Shark: A Hybrid Approach
- Class A+B -

MapReduce parallel query processing
+ RDBS data processing
+ $a$

= SHARK
Features of Shark

Build on top of Spark using RDD

Dynamic Query Optimization (PDE)

Supports low-latency, interactive SQL queries

Support efficient complex analytics such as ML

Compatibility with Apache Hive, Metastores, HiveQL, etc.
Shark Architecture
Example: SQL on RDD

Step 1: User submits a query
Step 2: Hive query compiler parses the query

Query

SELECT id, price
FROM Customer, Transactions
JOIN Customer
ON customer_ID.id=Transaction.buyer_id
WHERE Transaction.price > 100

Abstract syntax tree

SELECT id, price
JOIN Customer.ID == Transaction.buyer_id
WHERE price > 100

TABLE: Customer
TABLE: Transaction
Example: SQL on RDD

Step 3: Turns syntax tree into logical plan (Basic logical optimization)

Abstract syntax tree

```
SELECT ID, price
JOIN Customer.ID == Transaction.buyer_id
WHERE price > 100
TABLE: Customer
TABLE: Transaction
```

Logical execution plan

```
Load Table: Customer
PROJECT ID
Join at (id, buyer_id)

Load Table: Transaction
PROJECT buyer_id, price
Filter for: price > 100
```
Example: SQL on RDD

Step 4: Create physical plan (i.e. RDD transformation) (Additional optimization)
RDD

RDD is a small partition of records

Each RDD is either created from external FS (e.g. HDFS) or derived from applying operators to other RDDs.

Each RDD is immutable, and deterministic

Lineage of the RDDs are used for fault recovery
What are the benefits of RDDs?
What Does Hive Do Instead?

Step 4: Create physical plan (MapReduce stages)

Logical execution plan

Load Table: Customer
- PROJECT ID
- PROJECT buyer_id, price
- Filter for: price > 100
- JOIN at (id, buyer_id)

Load Table: Transaction

Physical execution plan

Master

Part 1
- Map
- Reduce

Part 2
- Map
- Reduce

Part 3
- Map
- Reduce

Output
Diff(MapReduce, Spark RDD)
Partial DAG Execution

Map join

Shuffle join
Columnar Memory Store

What are the benefits?
Distributed Data Loading

Tracking metadata to customize data compression of each RDD
Data Co-partitioning

Each partition of the parent RDD is used by at most one partition of the child RDD partition.

Schema-aware for faster join operations
Partition Statistics & Map Pruning

Avoid scanning irrelevant part of the data

**Row Storage**

1. john 4.1
2. mike 3.5
3. sally 6.4

**Column Storage**

1 2 3

john mike sally

4.1 3.5 6.4
Experiments

Pavlo et al. Benchmark (2.1TB data)

TPC-H Dataset (100GB/1TB data)

Real Hive Warehouse (1.7TB data)

Machine Learning Dataset (100GB data)
Experiment Setup

100 m2.4xlarge nodes

8 virtual cores each node

68GB memory each node

1.6TB local storage each node
Result Summary

Up to 100X faster than Hive for SQL

More than 100X faster than Hadoop for ML

Comparable performance to MPP databases

Even faster than MPP in some cases
Aggregation Queries

SELECT sourceIP, SUM (adRevenue)
FROM uservisits GROUP BY sourceIP
Join Query

Effects of data co-partitioning

Effects of dynamic query optimization
Fault Tolerance

Measuring Shark performance in presence of node failures
Real Hive Warehouse

![Chart showing query times for different systems: Shark, Shark (disk), and Hive. Time is measured in seconds. Q1 has times of 1.1, Q2 has times of 0.8, Q3 has times of 0.7, and Q4 has times of 1.0.](chart.png)
Machine Learning

Logistic regression per iteration runtime (seconds)

K-means clustering per iteration runtime (seconds)
Next Steps?

Implement optimizations promised in the paper (e.g. bytecode compilation of expression evaluators)

Exploit specialized data structures for more compact data representations and better cache behavior

More benchmark experiment should be conducted after optimization for comparison