**Shark: SQL and Rich Analytics at Scale**

**Name: Teja devarapalli**

**ID: 16282634**

**Summary:**

* Data size and complexity of analysis are two challenges faced.
* Map Reduce and MPP analytic databases.

**Map Reduce:**

* Offers fine-grained fault tolerance model
* It can express many statistical and learning algorithms
* Results in high latency is the major drawback.

**MPP analytic databases:**

* Most of the features that makes databases efficient are present
* Low latency.
* Coarser-grained recovery model is the drawback.

**Shark:**

* Shark is a data analysis system that supports both SQL query processing and machine learning
* SQL + Spark and is built on RDD
* RDD extended to run SQL efficiently.
* Support properties of Spark engine like control over data partitioning which are not present in Map Reduce.

**Compatibility with Apache Hive:**

* Users can run Hive queries much faster without any changes to either the queries or the data.
* Using RDDs and above-mentioned optimizations, Shark can run SQL queries up to 100x faster than Hive and iterative machine learning algorithms more than 100x faster than Hadoop.

**Spark:**

* Spark is the MapReduce like cluster computing engine used by Shark.
* In-memory computation
* RDDs are unique to Spark and are essential to enabling mid-query fault tolerance
* Spark engine is modified for Shark to support partial DAG execution.

**RDD:**

* Resilient Distributed dataset.
* Lazy evaluation
* Fault tolerant distributed dataset
* In memory computation
* caching
* Immutability
* Partitioning

**Executing SQL over RDDs:**

* Given a query, Shark uses the Hive query compiler to parse the query and generate an abstract syntax tree. The tree is then turned into a logical plan
* Up to this point, Shark and Hive share an identical approach. Hive would then convert the operator into a physical plan consisting of multiple MapReduce stages
* Shark creates a physical plan consisting of transformations on RDDs rather than MapReduce jobs
* It uses a variety of operators already present in Spark, such as map and reduce, as well as new operators we implemented for Shark, such as broadcast joins

**Engine Extensions:**

* **Partial DAG Execution (PDE)**
* Shark frequently used to query fresh data that has not undergone a data loading process
* This precludes the use of static query optimization techniques that rely on accurate a prior data statistic
* The lack of statistics for fresh data, combined with the prevalent use of UDFs, requires dynamic approaches to query optimization.

Join Optimization:

* In map join, also known as broadcast join, a small input table is broadcast to all nodes, where it is joined with each partition of a large table.

**Skew handling and degree of parallelism:**

* The degree of parallelism for reduce tasks can have a large performance impact: launching too few reducers may overload reducers’ network connections and exhaust their memories, while launching too many may prolong the job due to task scheduling overhead
* Using partial DAG execution, Shark can use individual partitions’ sizes to determine the number of reducers at run-time by coalescing many small, fine-grained partitions into fewer coarse partitions that are used by reduce tasks

**Distributed data loading:**

* Shark can load data into memory at the aggregated throughput of the CPU’s processing incoming data.
* During loading, a table is split into small partitions, each of which is loaded by a Spark task.

**Data co-partitioning:**

* A technique commonly used by MPP databases is to co-partition the two tables based on their join key in the data loading process.

**Partitioning statistics and Map Pruning:**

* Data is stored using some logical clustering on one or more columns.
* For example: entries in a website’s traffic

**Implementation:**

* Memory-based Shuffle
* Temporary Object Creation
* Bytecode Compilation of Expression Evaluators
* Specialized Data Structures
* Shark is evaluated using four datasets:
* Pavlo et al. Benchmark
* TPC-H Dataset
* Real Hive Warehouse
* Machine Learning Dataset