Hive vs Shark - Performance Evaluation

Report

Overview

This report depicts a performance evaluation between AMP Lab Shark and Apache Hive. This is with the objective of seeing the possibility to move the WSO2 BAM data processing which currently uses Hive, to Shark (and Apache Spark) for improved performance.

AMP Lab Shark

Shark is a large-scale data warehouse system for Spark designed to be compatible with Apache Hive. It can execute Hive QL queries up to 100 times faster than Hive without any modification to the existing data or queries. Shark supports Hive's query language, metastore, serialization formats, and user-defined functions, providing seamless integration with existing Hive deployments and a familiar, more powerful option for new ones. [1]

Apache Spark

Apache Spark is an open-source data analytics cluster computing framework originally developed in the AMPLab at UC Berkeley. Spark fits into the Hadoop open-source community, building on top of the Hadoop Distributed File System(HDFS). However, Spark is not tied to the two-stage MapReduce paradigm, and promises performance up to 100 times faster than Hadoop MapReduce for certain applications. [2]

Official documentation: [3]

Current Evaluation - Hive vs Shark

Current performance evaluation is done on the following software releases.

Query Language Engine	Apache Hive 0.11 [4]	Shark Shark 0.9.1 [5] (Latest release) which uses, • Scala 2.10.3 • Spark 0.9.1 [7] • AMPLab's Hive 0.9.0
Analytics framework	Hadoop 1.0.4 [6]	
File system	HDFS	

NOTE:

- Shark 0.9.1 is directly compatible with Hive 0.11 [5]. Therefore Hive 0.11 is used. (Even though there are newer Hive versions available)
- Shark 0.9.1 recommends Hadoop 1.0.4. for 1.x.x versions, there for it was used in both instances.
- Both engines uses HDFS file system.
- Both the implementations used JDBC to communicate with the servers.
- Both systems are used under the 'standalone mode' (Local implementation).
- Data sets were taken from a WSO2 CEP resource. [8]

Evaluation Procedure

Simple SQL (Hive Queries) were run on both systems and implementation times were taken from the IDE.

The number of rows of the input was increased from 10 to 1 billion on a logarithmic scale.

Queries run:

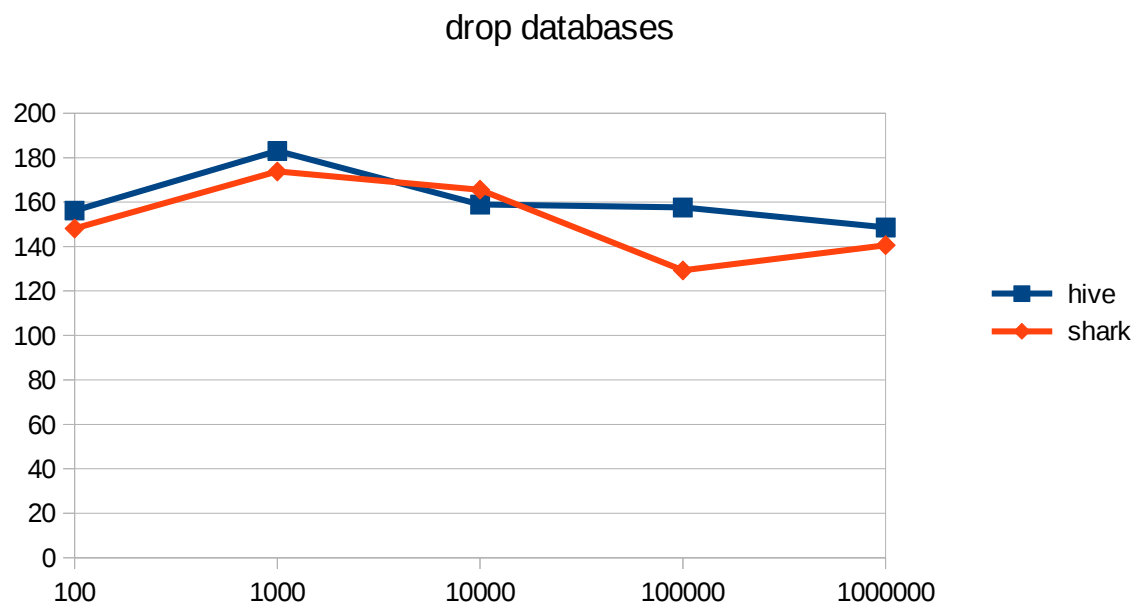
- DROP DATABASE IF EXISTS <DB> CASCADE
- CREATE DATABASE IF NOT EXISTS <DB>
- CREATE TABLE IF NOT EXISTS <TBL>

- CREATE TABLE IF NOT EXISTS <TBL> ROW FORMAT DELIMITED FIELDS TERMINATED BY <DLIMT>
- LOAD DATA LOCAL INPATH <PATH> OVERWRITE INTO TABLE <TBL>
- INSERT OVERWRITE TABLE <TBL> SELECT <PROP> FROM <TBL>
- SELECT <PROP> FROM <TBL> ORDER BY <CONDTION>
- SELECT <PROP> FROM <TBL> WHERE <CONDTION>
- SELECT COUNT (<PROP>) FROM <TBL>
- SELECT <PROPS> FROM <TBL> JOIN <TBL> ON (<CONDTION>)

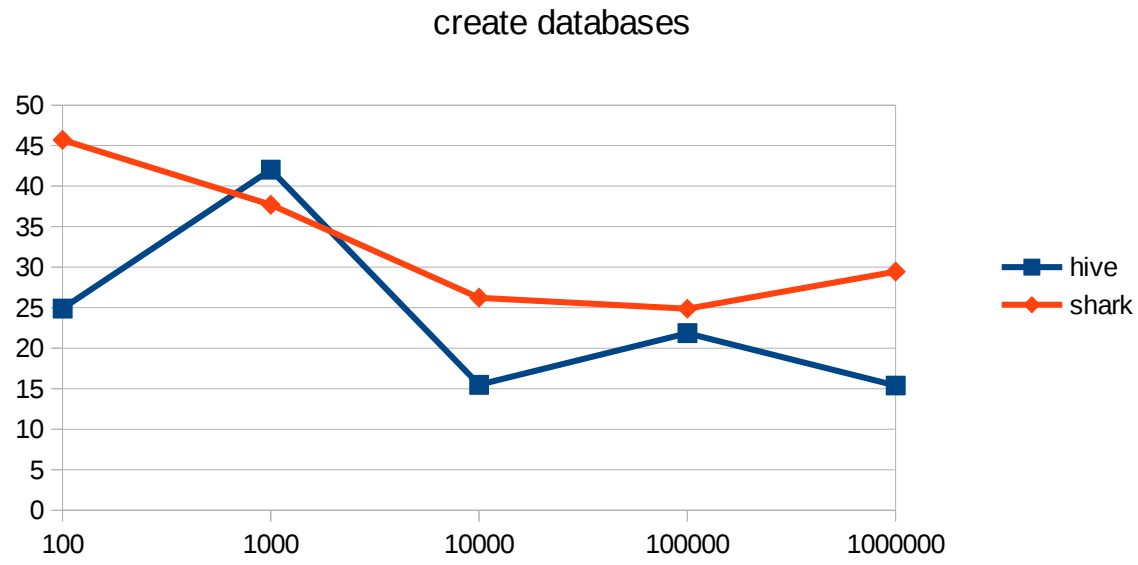
Results

Data Definition Language (DDL) Operations

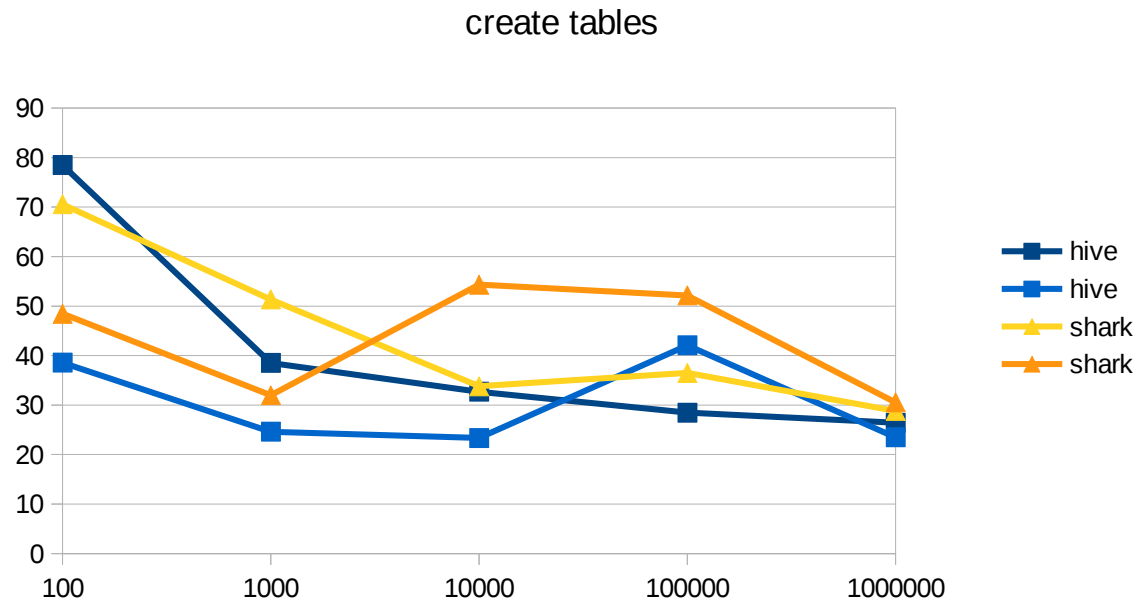
DROP DATABASE IF EXISTS <DB> CASCADE



CREATE DATABASE IF NOT EXISTS <DB>



CREATE TABLE IF NOT EXISTS <TBL>

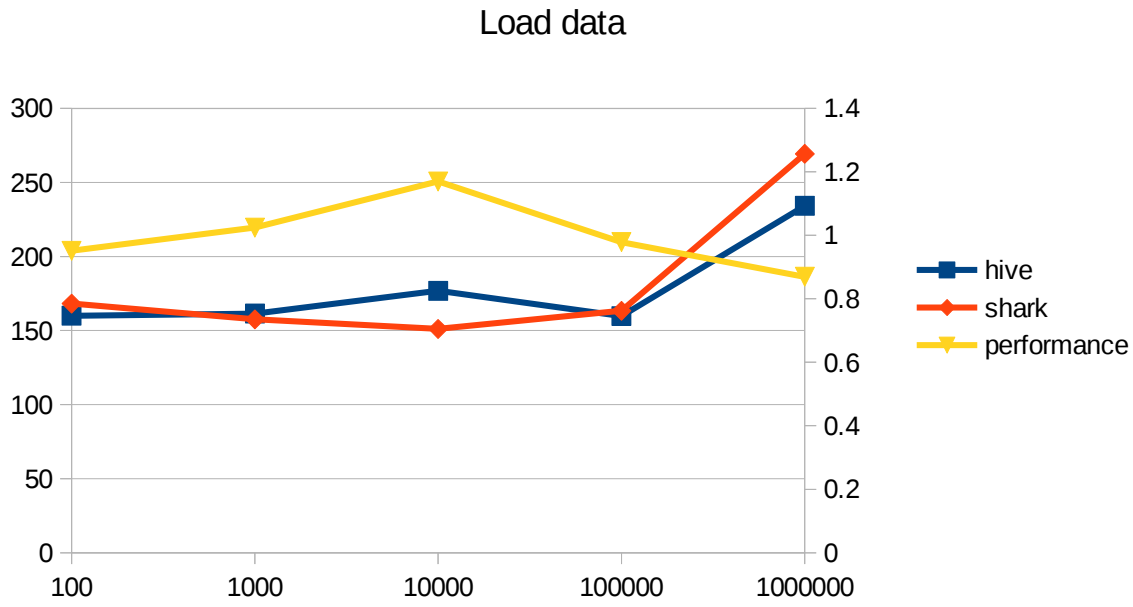


Considering the timing performance, there is no significant performance improvement in the DDL operations.

In fact, the performance of Hive is better than that of Shark in DDL operations.

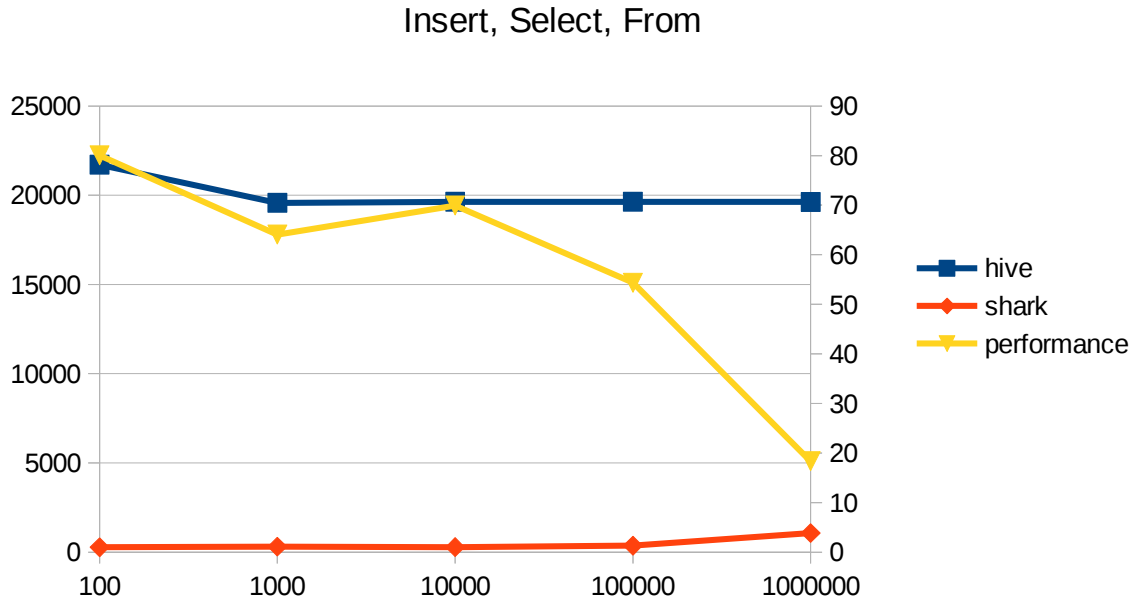
Hive Data Manipulation Language (DML) Operations

LOAD DATA LOCAL INPATH <PATH> OVERWRITE INTO TABLE <TBL>



The performance of Shark in LOAD operation is almost similar to Hive. There is no significant difference.

INSERT OVERWRITE TABLE <TBL> SELECT <PROP> FROM <TBL>

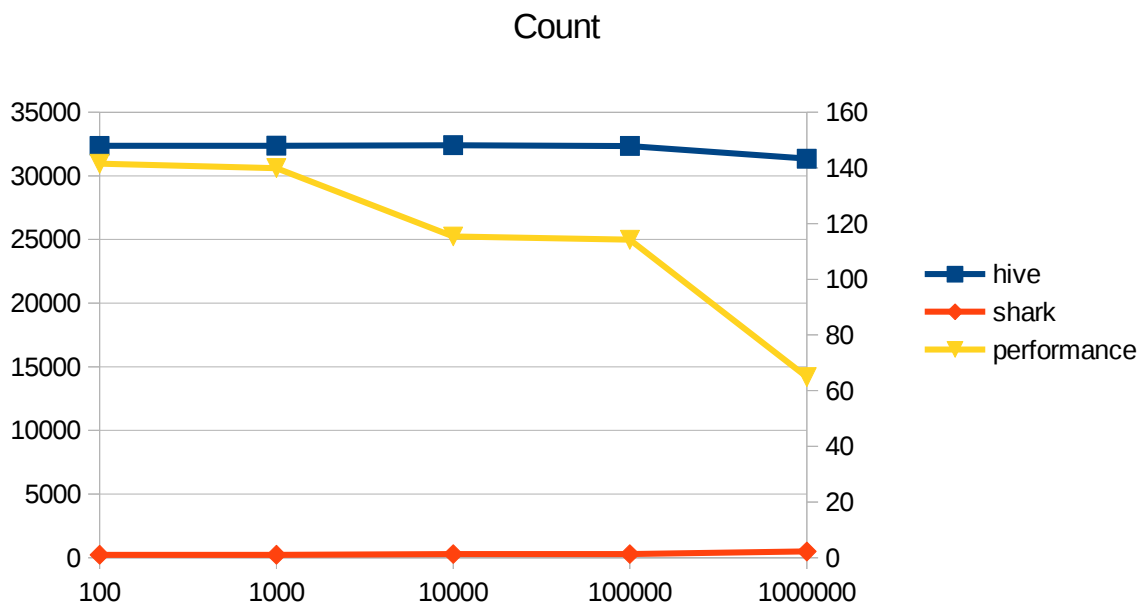


INSERT DML operation is significantly different from the LOAD operation, as Shark shows a significant performance improvement, giving a minimum 20x performance factor for 1 million

entries.

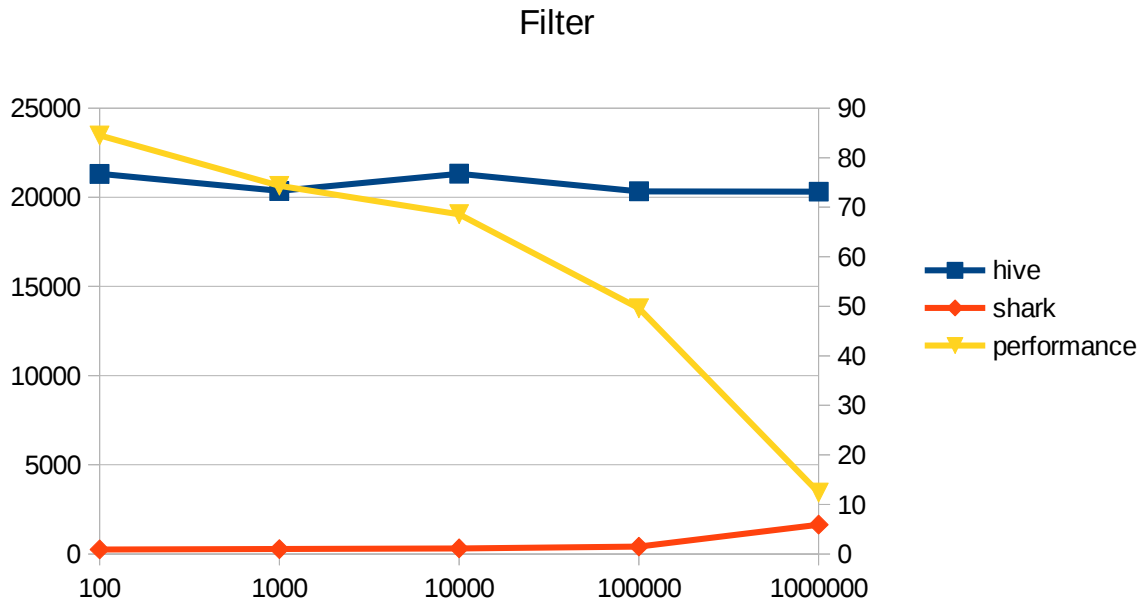
Data Retrieval Operations

COUNT



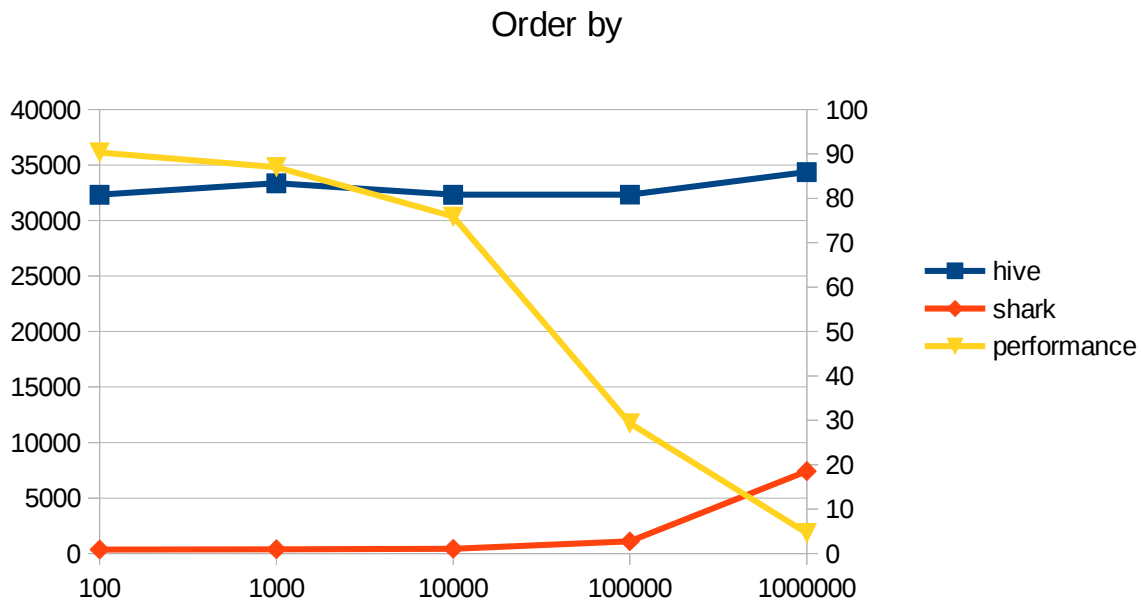
Performance factor is of 100x but it reduces while the number of rows of the input increases. For 1 million rows, the is around 60x.

SELECT and FILTER



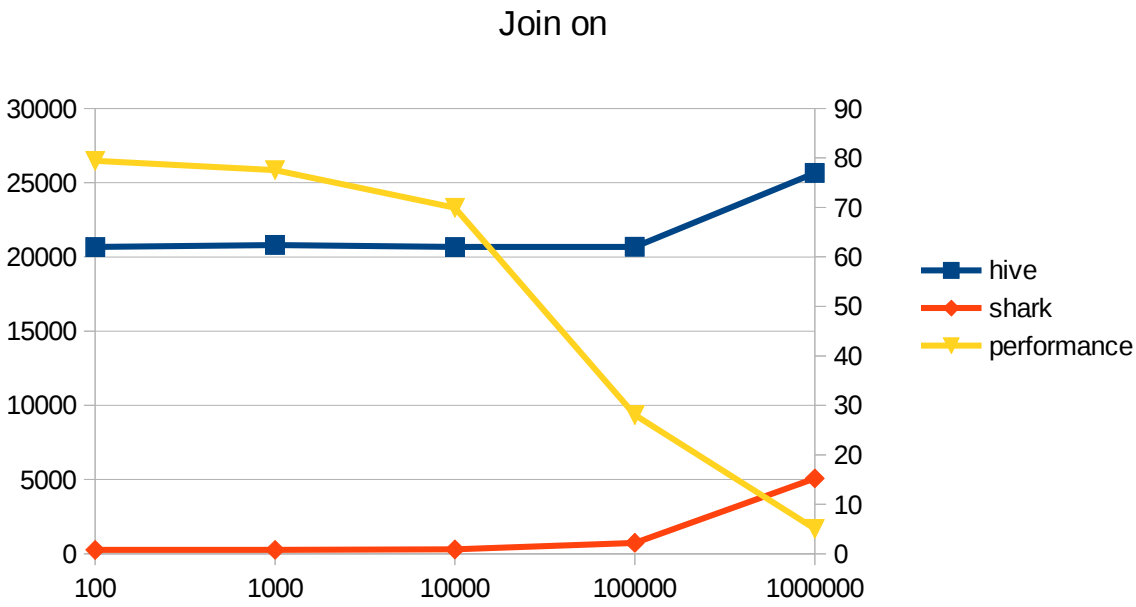
Performance of the filter operation of Hive has largely been fairly constant for all the inputs. Time taken in Shark has increased with the input size reaching a minimum performance factor of 10x for 1 million entries.

ORDER BY



Similar to FILTER (SELECT) the time taken in Shark has increased with the input size reaching a minimum performance factor of 5x for 1 million entries.

JOIN



JOIN also shows a similar performance to SELECT and ORDER BY, and it reaches a minimum performance factor of 5x for 10 million entries.

SUBQUERIES

Subqueries were not considered in the comparison because Hive 0.11 only supports subqueries only in the FROM clause (through Hive 0.12) [9] and this was not sufficient to make a meaningful query out of the dataset used. (Hive 0.13 and above supports some types of subqueries are supported in the WHERE clause.)

NOTE:

Implementations for input size of 10 million to 1 billion entries were aborted, as Shark exited with memory limit reached exception.

Conclusion

From the above evaluation, the following conclusions can be derived.

- Shark is indifferent to Hive in DDL operations (CREATE, DROP .. TABLE, DATABASE). Both engines show a fairly constant performance as the input size increases.
- Shark is indifferent to Hive in DML operations (LOAD, INSERT) but when a DML operation is called in conjuncture of a data retrieval operation (ex. INSERT <TBL> SELECT <PROP> FROM <TBL>), Shark significantly over-performs Hive with a performance factor of 10x+ (minimum). Shark performance factor reduces with the input size increases, while HIVE performance is fairly indifferent.
- Shark clearly over-performs Hive in Data Retrieval operations (FILTER, ORDER BY, JOIN). Hive performance is fairly indifferent in the data retrieval operations while Shark performance reduces as the input size increases. But at every instance Shark over-performed Hive with a minimum performance factor of 5x+.

Recommendation

Considering the performance improvement it is recommended to use Shark over Hive in the upcoming releases of WSO2 BAM for data processing operations.

It requires the current storage handlers to be edited, in order to support the use of Cassandra and H2 databases in Shark implementaitons.

Future Work

- The main abstraction Apache Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it.

  The use of RDDs and its impact to the performance is to be analyzed.

- Apache Spark also introduces its own query language named, Spark SQL [10].

  Use of this Spark SQL over Shark on Spark framework is to be analyzed.

Repositories

- The performance evaluation code and logs:

https://github.com/nirandaperera/hiveToShark/tree/master/hiveVsShark

- Performance evaluation docs:

https://drive.google.com/a/wso2.com/?ddrp=1#folders/0B1GsnfycTl32QTZqUktKck1Ucjg

- Data sets:

http://people.wso2.com/~wso2/cep_datasets/debs2014/

Reference

[1] https://github.com/amplab/shark/wiki
[2] http://en.wikipedia.org/wiki/Apache_Spark
[3] http://spark.apache.org/docs/latest/
[4] https://archive.apache.org/dist/hive/hive-0.11.0/
[5] https://github.com/amplab/shark/releases/tag/v0.9.1
[6] https://archive.apache.org/dist/hadoop/core/hadoop-1.0.4/
[7] http://spark.apache.org/downloads.html
[8] http://people.wso2.com/~wso2/cep_datasets/debs2014/
[9] https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SubQueries
[10] http://spark.apache.org/docs/latest/sql-programming-guide.html