# BIG PAPER SUMMARY

- **HIVE--A PETABYTE SCALE DATA WAREHOUSE USING HADOOP**
  **BY: THUSOO, SARMA, JAIN, SHAO, CHAKKA, ZHANG, ANTONY, LIU & MURTHY**

- **A COMPARISION OF APPROACHES TO LARGE-SCALE DATA ANALYSIS**
  **BY: PAVLO, PAULSON, RASIN, ABADI, DEWITT, MADDEN & STONEBRAKER**

- **ONE SIZE FITS ALL--AN IDEA WHOSE TIME HAS COME AND GONE**
  **BY: STONEBRAKER & CETINTEMEL**

---

By: Federica Bruno

March 7, 2017

# The Main Idea of Hive

- Prior to 2008, Facebook was built using DBMS, but the increasing amount of data generating was not suitable for the slow-paced traditional DBMS infrastructure.
- Facebook turned to Hadoop, an open source MapReducing implementation that is used at petabyte scale, which provides more scalability, flexibility, and executed tasks faster -- more suitable for Facebook's needs.
- However, Hadoop was not easy because it is completely customized; it lacked the expressiveness of query languages like SQL; so as result, programmers had to spend hours writing programs for simplest tasks.
- Hive, an open source warehousing program, is the easier and more productive version of Hadoop, which provided the solution for what Facebook end users needed.

# How Hive Is Implemented

- Hive, built on Hadoop, uses a MapReduce implementation
  - The Map task indicates how the input columns can be transformed into output columns with a user program
  - The Reduce task specifies the user program to invoke on the output columns
- Hive's Data Model is very similar to the traditional DBMS infrastructure
  - Hive Data is stored in tables composed of columns and rows
  - Supports all major primitive types: integers, lots, doubles, and strings
  - Support complex types: associated arrays, list, and structs
  - Metadata stored in "Metastore" is used by the query compiler to generate the execution plans
- HiveQL – a subset of SQL combining both familiar concepts and limitations
  - Subqueries, joins (inner, left outer, right outer and outer joins), group by(s), and aggregations remain the same
  - Only equality predicates are supported in a join predicates
  - Inserts are not possible due to the side effects of overwriting existing data

# How Hive Is Implemented (continued)

- Hive’s data storage located in HDFS directories
  - Tables: stored in a directory in HDFS
  - Partitions: stored in a subdirectory within a table's directory
  - Buckets: stored in a file within the partition's or table's directory if it is partitioned
- Hive’s main components:
  - Metastore: system catalog that stores the metadata
  - Driver: manages the lifecycle and maintains a session handle of a HiveQL statement
  - Query Compiler: compiles HiveQL into a directed acyclic graph of map and reduce tasks
  - Execution Engine: executes tasks produced by the compiler in dependency order
  - HiveServer: provides a way of integrating Hive with other applications

# My Analysis of Hive's Implementation

- I think it's smart that Facebook is catering to its needs of a growing application. A growing world comes with growing data, and it's awesome that Facebook has and has been continually restructuring their data processing infrastructure to create a better user experience.
- I think it's also great that a solution has been created to Hadoop's downfall in lacking an expressive query language by implementing a subset of familiar SQL functions called HiveQL. This alleviates the burden of end users constantly implementing repetitive tasks, and enhances user productivity by making the program easier to write, modify, and for the new programmer to understand.
- I also appreciate how weaknesses are acknowledged and are promoted as a working progress.

# The Main Idea of the Comparison Paper

- “A Comparison of Approaches to Large-Scale Data Analysis“ compares and contrasts Hadoop, the MapReduce model, from two Parallel DBMS models, DBMS-X and Vertica, on a cluster of 100 nodes.
- DBMS proved to be faster and required less code to implement each task, but took longer to tune and load data. Therefore, although the process to load data and execute data in the DBMSs took much longer than MapReduce, the performance was much better.
- The paper undermines the hype of MapReduce as the Best Data Software by providing evidence that the nature of the MapReduce model is just not well suited for long-term or larger-sized projects, but works better in a development environment with a small number of programmers and a limited application domain.

# Implementation of MapReduce & DBMSs

- MapReduce consists of two functions, map and reduce, used to process key/value data pairs. The input data set is stored in a collection of partitions, distributed on nodes in a cluster.
- Parallel DBMSs have two key aspects that enable parallel execution: 1) tables are partitioned over nodes in a cluster and 2) the system uses an optimizer that translates SQL commands into a query plan whose execution is divided among nodes.
- The main difference between DBMSs and MapReduce is that DBMSs require that data conform to a well defined schema whereas MapReduce allows data to be in any format.
- Other differences include how each system provides indexing and compression optimizations, programming models, the way data is distributed, and how queries are executed.
- To compare and contrast trade-offs between DBMSs and MapReduce, the four tests that were performed were: Load Time, Aggregation Task, Selection Task, & Join Task

# My Analysis of MapReduce & DBMSs

- I thought it was very interesting how it sheds light on elements that are thought to be negative such as “rigid data infrastructure” or “challenging query language” but proved to work better with processing large data than MapReduce did.
- For example, although there are built-in functionalities in MapReduce that handle key/value formats, a programmer would have to write support for more complex data structures like compound keys. If data sharing exists, a second programmer would have to decipher the code written by the first programmer. It also lacks constraint rules to ensure data referential integrity installed in system catalogs.
- This paper did a great job at providing graphic representations in comparing and contrasting MapReduce and DBMSs as well as providing sufficient information to argue that Hadoop and Hive will probably not be long-run winners in handling big data in the future.

# Comparison of Both Papers

- I liked how the comparison paper put in perspective how people often think that the absence of a rigid schema and a flexible query language makes MapReduce the preferable option for large-scale data processing, but concludes that there are drawbacks in not using a schema and a ruled-based language.
- After reading the comparison paper, it made sense how Hive was a working progress to solve the problems that came with the nature of MapReduce, and how their solutions mirrored DBMS functions. I also enjoyed reading about Hive in a real-world context such as Facebook.
- Hive's promising future suggests to be an combination MapReduce's flexibility and scalability with DBMS's schema, index implementation, and high-level query language.

# The Main Idea of Stonebraker's Talk

- Stonebraker's main message is "One size fits none".
- He explains how many existing data software will eventually be phased out by newer software, and will no longer have its place in the modern markets.
- For example, numerous markets such as data warehouses and complex and graphic analytics are now using column stores, which Stonebraker predicts to be a major hype in the future.
- New software technology like OLTP use such little memory that it will not longer need row storage.

# The Advantages & Disadvantages of Hive in Context to Comparison Paper and Stonebraker

## ADVANTAGES

- Hybrid of pros from both DBMS and MapReduce
- HiveQL enhances user productivity by making its easier for end users to write programs
- Has a more fault tolerate system than DBMS due to its simplicity and flexibility
- Installation, configuration, and usage is easier than DBMS
- Cost effective manner to satisfy the needs of Facebook

## DISADVANTAGES

- Loading data takes more time than DBM
- Does not support scheming or build-in indexing
- Only equality predicates are supported in a join predicates and inserts are not possible due to the side effects of overwriting existing data
- As stated in Stonebraker's talk, would not have been helpful to markets that use column stores
- As stated in Stonebraker's talk, will not be long-run winners in handling big data in the future