Matt Pineau
Database Management

December 12, 2014
Professor Labouseur

Big Data Paper Summary

Bibliography:

- Thusoo, Ashish, Joydeep Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. *Hive - A Petabyte Scale Data Warehouse Using Hadoop*. Proc. of ICDE Conference, California, Long Beach. N.p.: IEEE, 2010. Print.
- A comparison of approaches to large-scale data analysis. Pavlo et al., SIGMOD 2009

Hive: What's up with that?

- *Hive – A Petabyte Scale Data Warehouse Using Hadoop* is an article written by the Facebook Data Infrastructure Team (Facebook being the company that originally developed it) summarizing Hive's benefits over other petabyte-scale data processing systems, how it is implemented, and where it intends to go from here.
- Hive is essentially an open-source solution to the low level map-reduce programming model. They write, "end users had to write map-reduce programs for simple tasks like getting raw counts or averages (in regards to Hadoop)."
- The map-reduce programming model "requires developers to write custom programs which are hard to maintain and reuse." Hive is built on top of Hadoop and allows for queries to be expressed in a declarative language (HiveQL), which is a more intuitive format for users. These are "compiled into map-reduce jobs that are executed using Hadoop."
- Hive also allows for the customizability of the bare map-reduce programming model by enabling users to plug map-reduce scripts into their queries.
- In developing Hive, the Facebook team intended to merge the worlds of the traditional RDBMS and Hadoop, bringing the "familiar concepts of tables, columns, partitions , and a subset of SQL to the unstructured world of Hadoop, while still maintaining the extensibility and flexibility that Hadoop enjoyed."
- The end goal in developing Hive was to analyze data more productively by improving the query capabilities of Hadoop.

How do they make this work?

The Data Model/Type System

- Hive stores data in a manner similar to traditional databases (tables of rows and columns)
- Columns each have an associated type
- At the time this article was written, Hive supported the following primitive types: Integers, Floating Point Numbers, Strings; as well as the following complex types: Associative Arrays, Lists, and Structs

The Query Language

- HiveQL is the query language used which ”is comprised of a subset of SQL and some extensions.” This allows anyone familiar with SQL (who isn't), to fire up a hive command line interface and begin querying.
- Some limitations: only equality predicates supported by joins and joins must be specified by the ANSI join syntax (the latter not really a limitation I don't think), currently all inserts overwrite existing data.
- Extensions support analysis expressed as map-reduce programs by users in the language of their choice.

Data Storage

- Tables have metadata which associates the data in a table to hdfs directories (tables, partitions, buckets), despite being logical data units.

Serialization/Deserialization

- “Hive can take an implementation of the SerDe java interface provided by the user and associate it to a table or partition,” resulting in custom data formats that can be easily interpreted and queried from.
- Default implementation is LazySerDe. This results in the cost of deserialization of a columb only being incurred if the column is required in a query expression.
- More interesting SerDes are present in the hive_contrib.jar file.

File Formats

- Hadoop files can be stored in different formats, specifying how records are stored.
- No restrictions on any type of file input formats
- File format can be specified when a table is created using the STORED AS clause

System Architecture and Components (the following components/definitions are quoted directly from paper.

- Metastore – The component that stores the system catalog and metadata about tables, columns, partitions, etc.
- Driver – The component that manages the lifecycle of a HiveQL statement as it moves through Hive. Also maintains a session handle and session statistics.
- Query Compiler – The component that compiles HiveQL into a directed acyclic graph of map/reduce tasks.
- Execution Engine – The component that executes the tasks produced by the compiler in proper dependency order. Interacts with underlying Hadoop instance
- HiveServer – The component that provides a thrift interface and a JDBC/ODBC server and provides a way of integrating Hive with other applications.
- Client Components (Command Line Interface, the web UI, and JDBC/ODBC driver
- Extensibility Interfaces including the SerDe and ObjectInspector interfaces, and UDF and UDAF interfaces that enable users to define their own custom functions.

An Undergrad's Perspective

- The first time I read through this article, I must admit that I did not understand much more than the basic idea of Hive (the second time around I picked up more).
- The first page of *Hive – A Petabyte Scale Data Warehouse Using Hadoop,* gives a fairly simple overview of the purpose of Hive: to create another layer on top of Hadoop in order to create a more simplified and productive user experience (Great!)
- The idea of introducing tables of rows and columns, with the ability to be queried by a subset of SQL commands is an interesting one. While I do not know that much about Hadoop, I do know that what makes it useful in the realm of big data is that it strays from the traditional RDBMS model, trading accuracy for performance.
- My first thought after reading the introduction, is that Hive will just present another compromise. By crossing the worlds of the relational model and Hadoop, you would get slightly more accurate results, while trading some performance (this is ignoring the user experience factor for a moment). I figured that Hadoop sacrifices some accuracy of the information it retrieves for the ability to process extremely large amounts of data. What perhaps saves Hadoop is something like the law of large numbers. Since so much data is processed and retrieved, as long as it is simply used for data analysis and doesn't need to be perfectly precise, the information will converge to a fairly accurate result. I thought about Hive as just reintroducing the concept of tables to Hadoop, and therefore slowing down the process and still retrieving inaccurate results.
- However, it seems to me after reading this article that the main purpose of Hive is to reintroduce a declarative language to Hadoop. Perhaps this performance tradeoff pays off in the long run because developers no longer have to spend so much time writing complicated functions to do something as simple as averaging a set of data, or even counting.

MapReduce vs. Parallel DBMSs

- The entirety of the paper, *A Comparison of Approaches to Large-Scale Data Analysis*, is spent discussing the differences between MapReduce and Parallel DBMSs and their advantages and disadvantages.
- In this paper, it seems as though Parallel DBMSs are used as more of a basis of comparison for MapReduce, which makes sense because MapReduce is a new approach to analyzing big data and parallel DBMSs have been around since the 80's. For this reason I will mostly discuss MapReduce and where it exceeds and falls short relative to Parallel DBMSs.
- This paper puts MapReduces approach to handling big data into fairly simple terms. MapReduce seems to be very barebones (only functions are map and reduce) and relies heavily on programmers writing custom code for their implementation of the database. For example, they may need to write a custom parser, explicitly write support for more complex data structures (e.g. composite keys), assure that data added or modified does not violate integrity or other high-level constraints, implement indexes desired to speed up access to data, etc. This drawback of MapReduce is expressed again in its comparison to Codasyl (a data model that the relational model beat out back in the 70's) since "one is forced to write algorithms in a low-level language in order to perform record-level manipulation. However, work is being done to remove some of this burden from programmers by introducing high-level languages to run on top of MapReduce, such as Pig and Hive.
- Despite the almost overwhelming amount of negative information about MapReduce presented in the article, there were a few redeeming qualities noted. One is the structure of MapReduce, or the lack thereof. A MapReduce programmer is free to structure his data any way he chooses, whereas "parallel DBMSs require data to fit into the relational paradigm of rows and columns." MapReduce does tend to beat parallel DBMSs in the field of flexibility. The authors say that "despite its widespread adoption, SQL is routinely criticized for its insufficient expressive prowess." MapReduce is also much better at handling node failures during the execution of a MapReduce computation. This is due to the MR schedulers ability to automatically restart a unit of work on an alternate node. This is because "the output files of the Map phase are materialized locally instead of being streamed to the nodes running the Reduce tasks. This differs from parallel DBMSs that have larger granules of work which, in the case of a failure are rolled back to avoid saving intermediate results.
- The experiment results showed that on average, parallel DBMSs (DBMS-X and Vertica) out-performed the MapReduce system (Hadoop). On average DBMS-X was 3.2 times faster and Vertica was 2.3 times faster. However Hadoop was much easier to set up and use in comparison to the other DBMSs. Of the five tasks tested (data loading, selection, aggregation, join, and UDF aggregation) Hadoop only out-performed in dataloading.

Hive in Comparison

- One of the biggest issues with MapReduce is the amount of programming that has to go into the simplest of tasks. It seemed like it was trying to reinvent the wheel for each implementation of it, but Hive seems like a solution (or at least the start of a solution) to this.
- One of the best things about the relational model is the fact that it's grounded in mathematics, and therefore, if implemented properly, it guarantees accuracy. To me (although I may be wrong), it seems as though Hive is an attempt to implement queries that rely on these mathematical guarantees to a system that cannot provide them.
- Hive, being built on top of Hadoop, should provide the same ease of installation and configuration found in the experiment in the second paper, relative to parallel DBMSs.
- There are other issues with Hadoop other than an excessive amount of programming, such its "cold start" nature. According to the paper, "it takes 10 seconds from the moment that a job is submitted to the JobTracker before the first Map task begins to execute and 25 seconds until all the nodes in the cluster are executing the job." The authors did express though that this is not a flaw in the MapReduce model, simply Hadoop's version of it. Regardless, Hive runs on top of Hadoop.
- Hive essentially only provides more ease-of-use to Hadoop, offering users a SQL-like language, which while very useful, will not affect the performance of the data analysis. Hive will still yield the same performance results relative to parallel DBMSs (significantly slower in most aspects) as found in the experiment in the second paper.