Hive - Solution for Data Warehousing

Prof. Sachin Gupta[1], Prof. Yogesh Sharma[2]

[1,2] Sinhgad Institute of Management and Computer Application (SIMCA), Sinhgad Technical Campus, Narhe, Pune, Maharashtra, India - 411041

Abstract-
The size of data sets being collected and analyzed in the industry for business intelligence is growing rapidly, making traditional warehousing solutions prohibitively expensive. Hadoop is a popular open-source map-reduce implementation which is being used in companies like Yahoo, Facebook etc.to store and process extremely large data sets on commodity hardware. However, the map-reduce programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse. In this paper, we present Hive, an open-source data warehousing solution built on top of Hadoop. Hive supports queries expressed in a SQL-like declarative language -HiveQL, which are compiled into map-reduce jobs that are executed using Hadoop. In addition, HiveQL enables users to plug in custom map-reduce scripts into queries. The language includes a type system with support for tables containing primitive types, collections like arrays and maps, and nested compositions of the same. The underlying IO libraries can be extended to query data in custom formats. Hive also includes a system catalog-Metastore-that contains schemas and statistics, which are useful in data exploration, query optimization and query compilation.

Keywords - Big data, Hive, Date Warehousing.

I. INTRODUCTION

Big data is mainly collection of data sets so large and complex that it is very difficult to handle them using on-hand database management tools. The main challenges with Big databases include creation, curation, storage, sharing, search, analysis and visualization. So to manage these databases we need, "highly parallel software's". First of all, data is acquired from different sources such as social media, traditional enterprise data or sensor data etc. Flume can be used to acquire data from social media such as twitter. Then, this data can be organized using distributed file systems such as Google File System or Hadoop File System. These file systems are very efficient when number of reads are very high as compared to writes. At last, data is analyzed using mapreducer so that queries can be run on this data easily and efficiently.

Hadoop are the technologies that we have used to address these requirements at Facebook. The entire data processing infrastructure in Facebook prior to 2008 was built around a data warehouse built using a commercial RDBMS. The data that we were generating was growing very fast -as an example we grew from a 15TB data set in 2007 to a 700TB data set today. The infrastructure at that time was so inadequate that some daily data processing jobs were taking more than a day to process and the situation was just getting worse with every passing day. We had an data. As a result we started exploring Hadoop as a technology to address our scaling needs. The fact that Hadoop was already an open source project that was being used at petabyte scale and provided scalability using commodity hardware was a very compelling proposition for us. The same jobs that had taken more than a day to complete could now be completed within a few hours using Hadoop urgent need for infrastructure that could scale along within a few hours using Hadoop.

However, using Hadoop was not easy for end users, especially for those users who were not familiar with map-reduce. End users had to write map-reduce programs for simple tasks like getting raw counts or averages. Hadoop lacked the expressiveness of popular query languages like SQL and as a result users ended up spending hours (if not days) to write programs for even simple analysis. It was very clear to us that in order to really empower the company to analyze this data more productively, we had to improve the query capabilities of Hadoop. Bringing this data closer to users is what inspired us to build Hive in January 2007. Our vision was to bring 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. Hive was open sourced in August 2008 and since then has been used and explored by a number of Hadoop users for their data processing needs.

Right from the start, Hive was very popular with all users within Facebook. Today, we regularly run thousands of jobs on the Hadoop/Hive cluster with hundreds of users for a wide variety of applications starting from simple summarization jobs to business intelligence, machine learning applications and to also support Facebook product features.

II. DATA MODEL, TYPE SYSTEM AND QUERY LANGUAGE

Hive is designed to enable easy data summarization, ad-hoc querying and analysis of large volumes of data. It provides SQL which enables users to do ad-hoc querying, summarization and data analysis easily. At the same time, Hive's SQL gives users multiple places to integrate their own functionality to do custom analysis, such as User Defined Functions (UDFs).

A. Data Model and Type System

Primitive Types

- Types are associated with the columns in the tables.

The following Primitive types are supported:

- Integers
  - TINYINT-1 byte integer
  - SMALLINT-2 byte integer
  - INT-4 byte integer
  - BIGINT-8 byte integer
- Boolean type
  - BOOLEAN-TRUE/FALSE
- Floating point numbers
  - FLOAT-single precision
  - DOUBLE-Double precision
- Fixed point numbers
  - DECIMAL-a fixed point value of user defined scale and precision
- String types
  - STRING-sequence of characters in a specified character set
  - VARCHAR-sequence of characters in a specified character set with a maximum length
  - CHAR-sequence of characters in a specified character set with a defined length
- Date and time types
  - TIMESTAMP- a specific point in time, up to nanosecond precision
  - DATE-a date
- Binary types
  - BINARY-a sequence of bytes

Complex Types

Complex Types can be built up from primitive types and other composite types using:

- Structs: the elements within the type can be accessed using the DOT (.) notation. For example, for a column c of type STRUCT {a INT; b INT}, the a field is accessed by the expression c.a
- Maps (key-value tuples): The elements are accessed using ['element name'] notation. For example in a map M comprising of a mapping from 'group' -> gid the gid value can be accessed using M['group']
- Arrays (indexable lists): The elements in the array have to be in the same type. Elements can be accessed using the [n] notation where n is an index (zero-based) into the array. For example, for an array A having the elements ['a', 'b', 'c'], A[1] retruns 'b'.

Using the primitive types and the constructs for creating complex types, types with arbitrary levels of nesting can be created. For example, a type User may comprise of the following fields:

- gender-which is a STRING.
- active-which is a BOOLEAN.

CREATE TABLE

There are 2 types of tables in Hive, Internal and External. This case study describes creation of internal table, loading data in it, creating views, indexes and dropping table on weather data.

Creating Internal Table

Internal table are like normal database table where data can be stored and queried on. On dropping these tables the data stored in them also gets deleted and data is lost forever. So one should be careful while using internal tables as one drop command can destroy the whole data. Open new terminal and fire up hive by just typing hive. Create table on weather data.

CREATE TABLE weather (wban INT, date STRING, precip INT)

ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,'
LOCATION ' /hive/data/weather';
Load the Data in Table
Data can be loaded in 2 ways in Hive either from local file or from HDFS to Hive. To load the data from local to Hive.
following command in NEW terminal:
hadoop fs -copyFromLocal /home/user/data/weather/2012.txt hdfs://hname:10001 /hive/data/weather

Drop table

On dropping the table loaded by second method that is from HDFS to Hive, the data gets deleted and there is no copy of data on HDFS. This means that on creating internal table the data gets moved from HDFS to Hive.
Table can be dropped using:
DROP TABLE weather;
Creating external table
Open new terminal and fire up hive by just typing hive.
Create table on weather data.
CREATE EXTERNAL TABLE weatherext (wban INT, date STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,'
LOCATION ' /hive/data/weatherext';
Load the data in table
Load the data from HDFS to Hive using the following command:
LOAD DATA INPATH 'hdfs:/data/2012.txt' INTO TABLE weatherext;

B. Query Language in Hive.

The Hive Query Language (HiveQL) is a query language for Hive to process and analyze structured data in a Metastore. This chapter explains how to use the SELECT statement with WHERE clause.

SELECT statement is used to retrieve the data from a table. WHERE clause works similar to a condition. It filters the data using the condition and gives you a finite result. The built-in operators and functions generate an expression, which fulfils the condition.
Syntax
Given below is the syntax of the SELECT query:
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]
[LIMIT number];

III. DATA STORAGE, SERDE AND FILE FORMATS

A. Data Storage

Hive data is organized into:

- Databases: Namespaces function to avoid naming conflicts for tables, views, partitions, columns, and so on. Databases can also be used to enforce security for a user or group of users.
- Tables: Homogeneous units of data which have the same schema. An example of a table could be page_views table, where each row could comprise of the following columns (schema):
  - timestamp-which is of INT type that corresponds to a UNIX timestamp of when the page was viewed.
  - userid -which is of BIGINT type that identifies the user who viewed the page.
  - page_url-which is of STRING type that captures the location of the page.
  - referer_url-which is of STRING that captures the location of the page from where the user arrived at the current page.
  - IP-which is of STRING type that captures the IP address from where the page request was made.
- Partitions: Each Table can have one or more partition Keys which determines how the data is stored. Partitions-apart from being storage units-also allow the user to efficiently identify the rows that satisfy a specified criteria; for example, a date_partition of type STRING and country_partition of type STRING. Each unique value of the partition keys defines a partition of the Table. For example, all "US" data from "2009-12-23" is a partition of the page_views table. Therefore, if you run analysis on only the "US" data for 2009-12-23, you can run that query only on the relevant partition of the table, thereby speeding up the analysis significantly. Note however, that just because a partition is named 2009-12-23 does not mean that it contains all or only data from that date; partitions are named after dates for convenience; it is the user's job to guarantee the relationship between partition name and data content! Partition columns are virtual columns, they are not part of the data itself but are derived on load.

- Buckets (or Clusters): Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of the Table. For example the page_views table may be bucketed by userid, which is one of the columns, other than the partitions columns, of the page_view table. These can be used to efficiently sample the data.

Note that it is not necessary for tables to be partitioned or bucketed, but these abstractions allow the system to prune large quantities of data during query processing, resulting in faster query execution.

B. Serialization/Deserialization (SerDe)

The SerDe interface allows you to instruct Hive as to how a record should be processed. A SerDe is a combination of a Serializer and a Deserializer (hence, Ser-De). The Deserializer interface takes a string or binary representation of a record, and translates it into a Java object that Hive can manipulate. The Serializer, however, will take a Java object that Hive has been working with, and turn it into something that Hive can write to HDFS or another supported system. Commonly, Deserializers are used at query time to execute SELECT statements, and Serializers are used when writing data, such as through an INSERT-SELECT statement.

Input Processing

- Hive's execution engine (referred to as just engine henceforth) first uses the configured InputFormat to read in a record of data (the value object returned by the RecordReader of the InputFormat).
- The engine then invokes Serde.deserialize() to perform deserialization of the record. There is no real binding that the deserialized object returned by this method indeed be a fully deserialized one. For instance, in Hive there is a LazyStruct object which is used by the LazySimpleSerDe to represent the deserialized object. This object does not have the bytes deserialized up front but does at the point of access of a field.
- The engine also gets hold of the ObjectInspector to use by invoking Serde.getObjectInspector(). This has to be a subclass of structObjectInspector since a record representing a row of input data is essentially a struct type.
- The engine passes the deserialized object and the object inspector to all operators for their use in order to get the needed data from the record. The object inspector knows how to construct individual fields out of a deserialized record. For example, StructObjectInspector has a method called getStructFieldData() which returns a certain field in the record. This is the mechanism to access individual fields. For instance ExprNodeColumnEvaluator class which can extract a column from the input row uses this mechanism to get the real column object from the serialized row object. This real column object in turn can be a complex type (like a struct). To access sub fields in such complex typed objects, an operator would use the object inspector associated with that field (The top level Struct Object Inspector for the row maintains a list of field level object inspectors which can be used to interpret individual fields).

For UDFs the new GenericUDF abstract class provides the ObjectInspector associated with the UDF arguments in the initialize() method. So the engine first initializes the UDF by calling this method. The UDF can then use these ObjectInspectors to interpret complex arguments (for simple arguments, the object handed to the udf is already the right primitive object like LongWritable/IntWritable etc).

Output Processing

Output is analogous to input. The engine passes the deserialized Object representing a record and the corresponding ObjectInspector to Serde.serialize(). In this context serialization means converting the record object to an object of the type expected by the OutputFormat which will be used to perform the write. To perform this conversion, the serialize() method can make use of the passed ObjectInspector to get the individual fields in the record in order to convert the record to the appropriate type.

C. File Format

A file format is the way in which information is stored or encoded in a computer file. In Hive it refers to how records are stored inside the file. As we are dealing with structured data, each record has to be its own structure. How records are encoded in a file defines a file format. These file formats mainly varies between data encoding, compression rate, usage of space and disk I/O. Most commonly used file formats are text file,sequence file,RC(RECORD-COLUMNAR) file and ORC(OPTIMIZED ROW-COLUMNAR) file.

- TextFile Format: TEXTFILE format is a famous input/output format used in Hadoop. In Hive if we define a table as TEXTFILE it can load data of form CSV (Comma Separated Values), delimited by Tabs, Spaces and JSON data. This means fields in each record should be separated by comma or space or tab or it may be JSON(Java Script Object Notation) data.

By default if we use TEXTFILE format then each line is considered as a record.
Create a text file by specifying STORED AS TEXTFILE in the end of a CREATE TABLE statement.
(e.g) create table text_file(id int,name string,age int,department string,location string) row format delimited fields terminated by ',' lines terminated by '\n' stored as textfile;

- Sequence File Format:Sequence files are flat files consisting of binary key-value pairs. When Hive converts queries to MapReduce jobs, it decides on the appropriate key-value pairs to be used for a given record. Sequence files are in binary format which are able to split and the main use of these files is to club two or more smaller files and make them as a one sequence file.Create a sequence file by specifying STORED AS SEQUENCEFILE in the end of a CREATE TABLE statement.

(e.g) create table sequence_file(id int,name string,age int,department string,location string) row format delimited fields terminated by ',' lines terminated by '\n' stored as sequencefile;

- Rc file: RCFILE stands of Record Columnar File which is another type of binary file format which offers high compression rate on the top of the rows. RCFILE is used when we want to perform operations on multiple rows at a time. RCFILEs are flat files consisting of binary key/value pairs, which shares much similarity with SEQUENCEFILE. RCFILE stores columns of a table in form of record in a columnar manner.

Create a Rc file by specifying STORED AS RCFILE in the end of a CREATE TABLE statement.

(e.g) create table rc_file(id int,name string,age int,department string,location string) row format delimited fields terminated by ',' lines terminated by '\n' stored as rcfile;

- Orc file: ORC stands for Optimized Row Columnar which means it can store data in an optimized way than the other file formats. ORC reduces the size of the original data up to 75%. As a result the speed of data processing also increases. ORC shows better performance than Text, Sequence and RC file formats. Create a Orc file by specifying STORED AS RCFILE in the end of a CREATE TABLE statement.
(e.g) create table orc_file(id int,name string,age int,department string,location string) row format delimited fields terminated by ',' lines terminated by '\n' stored as orcfile;

IV. SYSTEM ARCHITECTURE AND COMPONENTS

Fig1. Hadoop Hive Architecture

The above diagram shows the basic Hadoop Hive architecture. Primarily The diagram represents CLI (Command Line Interface),JDBC/ODBC and Web GUI (Web Graphical User Interface).This represents when user comes with CLI(Hive Terminal) it directly connected to Hive Drivers,When User comes with JDBC/ODBC(JDBC Program) at that time by using API(Thrift Server) it connected to Hive driver and when the user comes with Web GUI(Ambari server) it directly connected to Hive Driver.

The hive driver receives the tasks(Queries) from user and send to Hadoop architecture.The Hadoop architecture uses name node,data node,job tracker and task tracker for receiving and dividing the work what Hive sends to Hadoop

Major Component of Hive:
UI :- UI means User Interface, The user interface for users to submit queries and other operations to the system.
Driver :- The Driver is used for receives the quires from UI .This component implements the notion of session handles and provides execute and fetch APIs modeled on JDBC/ODBC interfaces.
Compiler :- The component that parses the query, does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition

metadata looked up from the metastore.

MetaStore :- The component that stores all the structure information of the various tables and partitions in the warehouse including column and column type information, the serializers and deserializers necessary to read and write data and the corresponding HDFS files where the data is stored.

Execution Engine :- The component which executes the execution plan created by the compiler. The plan is a DAG of stages. The execution engine manages the dependencies between these different stages of the plan and executes these stages on the appropriate system components.

The below diagram represents clear internal Hadoop Hive Architecture

Fig2. Hive_architecture

The above diagram shows how a typical query flows through the system

Step 1 :- The UI calls the execute interface to the Driver

Step 2 :- The Driver creates a session handle for the query and sends the query to the compiler to generate an execution plan

Step 3&4 :- The compiler needs the metadata so send a request for getMetaData and receives the sendMetaData request from MetaStore.

Step 5 :- This metadata is used to typecheck the expressions in the query tree as well as to prune partitions based on query predicates. The plan generated by the compiler is a DAG of stages with each stage being either a map/reduce job, a metadata operation or an operation on HDFS. For map/reduce stages, the plan contains map operator trees (operator trees that are executed on the mappers) and a reduce operator tree (for operations that need reducers).

Step 6 :- The execution engine submits these stages to appropriate components (steps 6, 6.1, 6.2 and 6.3). In each task (mapper/reducer) the deserializer associated with the table or intermediate outputs is used to read the rows from HDFS files and these are passed through the associated operator tree.Once the output generate it is written to a temporary HDFS file though the serializer. The temporary files are used to provide the to subsequent map/reduce stages of the plan.For DML operations the final temporary file is moved to the table's location

Step 7, 8&9 :- For queries, the contents of the temporary file are read by the execution engine directly from HDFS as part of the fetch call from the Driver.

V. Data Warehouses

Building a true data warehouse can be a massive project. There are different appliances, methodologies, and theories. What is the lowest common value? What are the facts, and what subjects relate back to those facts? And how do you mix, match, merge, and integrate systems that might have been around for decades with systems that only came to fruition a few months ago? This was before big data and Hadoop. Add unstructured, data, NoSQL, and Hadoop to the mix, and suddenly you have a massive data-integration project on your hands.

The simplest way to describe a data warehouse is to realize that it comes down to star schemas, facts, and dimensions. How you go about creating those elements is really up to you - whether it's through staging databases; on-the-fly extract, transform, load processes; or integrating secondary indices. Certainly, you can build a data warehouse with star schemas, facts, and dimensions, using Hive as the core technology, but it won't be easy. Outside the Hadoop world, it becomes an even bigger challenge. Hive is far more an integration, transformation, quick lookup tool compared to a legitimate data warehouse.

VI. Build a data warehouse with Hive

Three objects approach an enterprise. The first - the data warehouse - is massive: It brings history and experience, and it talks a good game. And most of it is true. But it's also bloated in many ways, expensive in a lot of other ways, and people are tired of the cost for varied results. Apache Hadoop walks into the same building and throws out claims of taking over the market. It preaches big data, velocity, volume, variety, and a bunch of other v words that don't mean much outside of a marketing plan. It throws out analytics, predictions, and much more. And it's cheap. People stop and listen. Apache Hive steps outside of the box, but does not attempt to beat the other objects. It wants to work with Hadoop, but unlike Hadoop, it doesn't want to throw the data warehouse to the curb. Hive has data warehouse capabilities, but with business intelligence (BI) and analytic limitations. It has database possibilities, but relational database management system (RDBMS) and Structured Query Language (SQL) limitations. It is more open and honest. It relates to the data warehouse. It relates to the RDBMS. But it never comes out and claims that it's more than meets the eye. Hadoop interrupts and proclaims it is the data warehouse for the Hadoop world. Hadoop seems to have sent its best marketing public relations rep, and what went from a simple conversation turned into Hive and Hadoop saving the world. It's intriguing. It's interesting.

VII.Conclusion

Obviously, a lot of design work goes into creating a simple star schema. You can go back and create a fact table based on teams, for example. The benefit of this data warehouse schema is that you won't have to join a lot of tables. And for this example, there are few updates besides yearly stats, and, in that case, overwriting the data warehouse tables or adding another year and recalculating shouldn't be a problem. Hive certainly has its limitations, but if you're working on a budget or the tools have been mandated from on high, with a bit of work, Hive can give you the data warehouse you need.

VIII. References.

I. https://cwiki.apache.org/confluence/display/Hive/Tutorial

II. http://blog.cloudera.com/blog/2012/12/how-to-use-a-serde-in-apache-hive/

III. http://www.geoinsyssoft.com/hive-file-format-examples/

IV. https://cwiki.apache.org/confluence/display/Hive/SerDe

V. https://www.ibm.com/developerworks/library/bd-hivewarehouse/index.html

VI. http://www.hadooptpoint.org/hadoop-hive-architecture/