Apache Hive

By-Shraddha Nikam

Introduction to Apache Hive

What is HIVE

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by Facebook.

Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive query language) which gets internally converted to MapReduce jobs.

Using Hive, we can skip the requirement of the traditional approach of writing complex MapReduce programs. Hive supports Data Definition Language (DDL), Data Manipulation Language (DML), and User Defined Functions (UDF).

Features of Hive

- Hive is fast and scalable.
- It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
- It is capable of analyzing large datasets stored in HDFS.
- It allows different storage types such as plain text, RCFile, and HBase.
- It uses indexing to accelerate queries.
- It can operate on compressed data stored in the Hadoop ecosystem.
- It supports user-defined functions (UDFs) where user can provide its functionality.

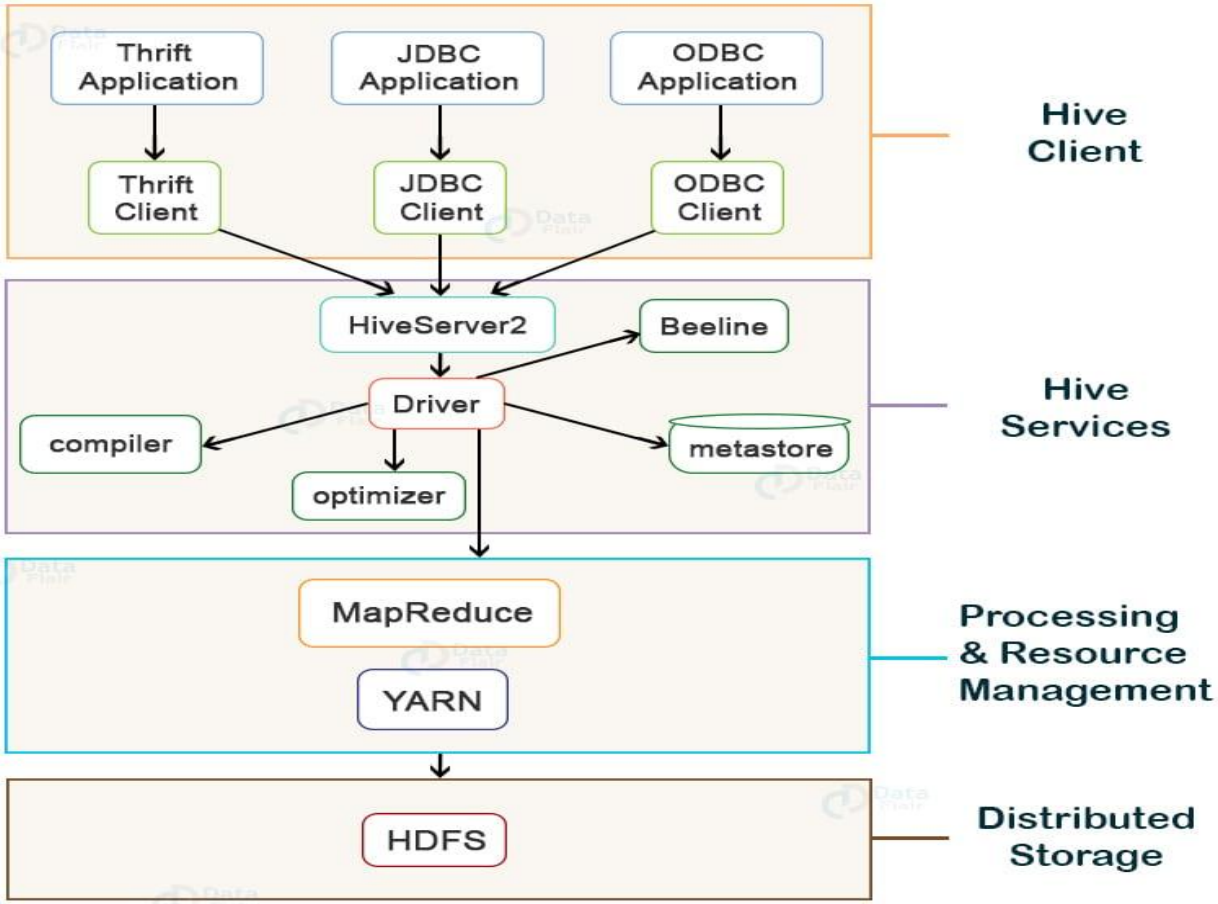
Limitations of Hive

- Hive is not capable of handling real-time data.
- It is not designed for online transaction processing.
- Hive queries contain high latency.

Differences between Hive and Pig

Hive	Pig
Hive is commonly used by Data Analysts.	Pig is commonly used by programmers.
It follows SQL-like queries.	It follows the data-flow language.
It can handle structured data.	It can handle semi-structured data.
It works on server-side of HDFS cluster.	It works on client-side of HDFS cluster.
Hive is slower than Pig.	Pig is comparatively faster than Hive.

Hive Architecture and Components:



Hive Architecture & Its Components

The above figure shows the architecture of Apache Hive and its major components. The major components of Apache Hive are:

1. **Hive Client**
2. **Hive Services**
3. **Processing and Resource Management**
4. **Distributed Storage**

1. Hive Client:

Hive drivers support applications written in any language like Python, Java, C++, and Ruby, among others, using JDBC, ODBC, and Thrift drivers, to perform queries on the Hive. Therefore, one may design a hive client in any language of their choice.

The three types of Hive clients are referred to as Hive clients:

1. **Thrift Clients**: The Hive server can handle requests from a thrift client by using Apache Thrift.
2. **JDBC client**: A JDBC driver connects to Hive using the Thrift framework. Hive Server communicates with the Java applications using the JDBC driver.
3. **ODBC client**: The Hive ODBC driver is similar to the JDBC driver in that it uses Thrift to connect to Hive. However, the ODBC driver uses the Hive Server to communicate with it instead of the Hive Server.

2. Hive Services:

Hive provides numerous services, including the Hive server2, Beeline, etc. The services offered by Hive are:

1. **Beeline**: HiveServer2 supports the Beeline, a command shell that which the user can submit commands and queries to. It is a JDBC client that utilises SQLLINE CLI (a pure Java console utility for connecting with relational databases and executing SQL queries). The Beeline is based on JDBC.
2. **Hive Server 2**: HiveServer2 is the successor to HiveServer1. It provides clients with the ability to execute queries against the Hive. Multiple clients may submit queries to Hive and obtain the desired results. Open API clients such as JDBC and ODBC are supported by HiveServer2.

Note: Hive server1, which is also known as a Thrift server, is used to communicate with Hive across platforms. Different client applications can submit requests to Hive and receive the results using this server.

HiveServer2 handled concurrent requests from more than one client, so it was replaced by HiveServer1.

Hive Driver: The Hive driver receives the HiveQL statements submitted by the user through the command shell and creates session handles for the query.

Hive Compiler: Metastore and hive compiler both store metadata in order to support the semantic analysis and type checking performed on the different query blocks and query expressions by the hive compiler. The execution plan generated by the hive compiler is based on the parse results.

The DAG (Directed Acyclic Graph) is a DAG structure created by the compiler. Each step is a map/reduce job on HDFS, an operation on file metadata, and a data manipulation step.

Optimizer: The optimizer splits the execution plan before performing the transformation operations so that efficiency and scalability are improved.

Execution Engine: After the compilation and optimization steps, the execution engine uses Hadoop to execute the prepared execution plan, which is dependent on the compiler's execution plan.

Metastore: Metastore stores metadata information about tables and partitions, including column and column type information, in order to improve search engine indexing.

The metastore also stores information about the serializer and deserializer as well as HDFS files where data is stored and provides data storage. It is usually a relational database. Hive metadata can be queried and modified through Metastore.

We can either configure the metastore in either of the two modes:

1. **Remote:** A metastore is not enabled in remote mode, and non-Java applications cannot benefit from Thrift services.
2. **Embedded:** A client in embedded mode can directly access the metastore via JDBC.

HCatalog: HCatalog is a Hadoop table and storage management layer that provides users with different data processing tools such as Pig, MapReduce, etc. with simple access to read and write data on the grid.

The data processing tools can access the tabular data of Hive metastore through It is built on the top of Hive metastore and exposes the tabular data to other data processing tools.

WebHCat: The REST API for HCatalog provides an HTTP interface to perform Hive metadata operations. WebHCat is a service provided by the user to run Hadoop MapReduce (or YARN), Pig, and Hive jobs.

3. Processing and Resource Management:

Hive uses a MapReduce framework as a default engine for performing the queries, because of that fact.

MapReduce frameworks are used to write large-scale applications that process a huge quantity of data in parallel on large clusters of commodity hardware. MapReduce tasks can split data into chunks, which are processed by map-reduce jobs.

4. Distributed Storage:

Hive is based on Hadoop, which means that it uses the Hadoop Distributed File System for distributed storage.

Hive Metastore

Hive Metastore is a component of Apache Hive, a data warehouse infrastructure built on top of Hadoop. It acts as a central metadata repository that stores and manages metadata information related to tables, partitions, and schemas. The Hive Metastore not only stores the structure of the data but also tracks the location of the data stored in Hadoop Distributed File System (HDFS) or other storage systems.

Hive Metastore service is responsible for managing and persisting metadata, which helps us in creating a tabular representation of files. HMS uses a relational database to store these metadata. In embedded mode, it uses Derby databases to store the metadata. But it can use any JDBC-supported database as the backend.

Supported DBs — MySQL | Postgres | Oracle | MS SQL Server

How Hive Metastore Works

Hive Metastore allows users to define and manage tables and schemas using a rich metadata model. It provides a schema-on-read approach, allowing users to define and evolve the structure of their data independently of the physical data files. The metadata in Hive Metastore can be accessed using the Hive Metastore service or through APIs, enabling integration with various data processing systems and tools.

Why Hive Metastore is Important

Hive Metastore plays a crucial role in enabling efficient data processing and analytics in a Hive environment. It provides several benefits, including:

- **Metadata Management:** Hive Metastore stores and manages metadata information, making it easier to organize, discover, and analyze data.
- **Data Abstraction:** Hive Metastore enables schema-on-read, allowing users to work with logical tables and schemas without worrying about the underlying physical data format.
- **Data Catalog:** The central metadata repository in Hive Metastore serves as a data catalog, providing information about available tables, partitions, and schemas.
- **Data Lineage:** Hive Metastore tracks the lineage of data, allowing users to trace the origin and transformation of data.
- **Integration:** Hive Metastore integrates with various data processing systems, enabling seamless data access and analytics across the ecosystem.

Comparing Hive with Traditional Databases

1. Schema Flexibility

Traditional Databases: Traditional databases, often relational, require a well-defined schema upfront. Schema changes can be complex and may disrupt ongoing operations.

Hive: Hive provides schema-on-read, allowing users to define the structure of data during query execution. This flexibility is advantageous when dealing with unstructured or semi-structured data.

2. Query Language

Traditional Databases: Relational databases use SQL as the standard query language, which is familiar to most data professionals.

Hive: Hive uses HQL, which closely resembles SQL. However, complex queries might perform slower due to the underlying MapReduce processing.

3. Performance

Traditional Databases: Traditional databases are optimized for transactional operations and perform well for small to medium-sized datasets.

Hive: While Hive is scalable and suitable for large datasets, it may not match the real-time performance of traditional databases for ad-hoc queries.

4. Data Processing Paradigm

Traditional Databases: Traditional databases are optimized for OLAP (Online Analytical Processing) or OLTP (Online Transaction Processing), depending on the use case.

Hive: Hive is well-suited for batch processing and data warehousing scenarios. Its performance shines in complex data transformations and analytics tasks.

5. Use Cases

Traditional Databases: Traditional databases excel in scenarios where data is well-structured, and real-time processing is critical, such as online banking applications.

Hive: Hive is ideal for scenarios involving large-scale data processing, log analysis, social media analytics, and other Big Data use cases.

HiveQL

Hive Query Language

- Hive QL is the HIVE QUERY LANGUAGE
- Hive offers no support for row-level inserts, updates, and deletes.
- Hive does not support transactions.
- Hive adds extensions to provide better performance in the context of Hadoop and to integrate with custom extensions and even external programs.
- DDL and DML are the parts of HIVE QL
- Data Definition Language (DDL) is used for creating, altering and dropping databases, tables, views, functions and indexes.
- Data manipulation language is used to put data into Hive tables and to extract data to the file system and also how to explore and manipulate data with queries, grouping, filtering, joining etc.

Databases in Hive:

- The Databases in the Hive is essentially just a catalog or namespace of tables.
- They are very useful for larger clusters with multiple teams and users, as a way of avoiding table name
- Hive provides commands such as
    - CREATE DATABASE db name -- to create a database in Hive
    - USE db name -- To use the database in Hive.
    - DROP db name -- To delete the database in Hive.
    - SHOW DATABASE -- to see the list of the DataBase

If no database is specified, tables belong to the default Data Base.

Tables in Hive:

Hive table is logically made up of the data being stored and the associated metadata describing the layout of the data in the table.

- The data typically resides in HDFS, although it may reside on any Hadoop file system including the local file system.
- Hive stores the metadata in a relational database and not in HDFS
- The command for creating a table in Hive is

```
have>CREATE TABLE EMP (empid int, ename string, esal double)
ROW FORMAT DELIMITED FIELDS TERMINATED By ‘t’ LINES TERMINATED
by ‘n’ STORED AS TEXT FILE;
```

To display the description of the table we use have>desc emp;

To have, we are having two types of tables

1. Managed tables
2. External tables

1. Managed tables

Managed tables are the one which will be managed in the Hive warehouse i.e. whenever we create a managed table definition, it will be stored under the default location of the Hive warehouse i.e./user/Hive/warehouse.

- When we drop a managed table, Hive deletes the data in the table
- Managed tables are less convenient for sharing with other tools.

Syntax for creating Hive managed table:-

Hive>create table manage- tab (empid, ename string, esal int) row format delimited fields terminated by ‘t’ lines terminated by ‘m’ stored as a text file;

- The table will be created under /user/Hive/warehouse/managed-tab by giving the command as

```
#hadoop fs –ls/user/Hive/warehouse.
```

- How to load the data into managed tables

We can load the data in two ways

1. Local Mode
2. HDFS Mode

In local mode, the syntax is

hive>load data local in path'/home/new Batch/input1.txt'

Into table managed-tab;

For HDFS mode, the syntax is

hive>load data in path'/user/ramesh/Hive/input2.txt'

Into table managed – tab;

Once the successful loading of the table and once the file is loaded, the file will be deleted in HDFS path and we can see in use/Hive/warehouse

2) External Tables:

Along with the managed tables, Hive also uses external tables.

Whenever the keyword ‘external’ comes in the table definition part. A hive will not bother about the table definition, i.e. the external table will not be managed by the Hive warehouse system.

Along with the external keyword, we can also mention the ‘location’ in the table definition, where exactly the table definition will get stored.

When you drop an external table, Hive leaves the data untouched and only delete the meta data.

Syntax:-

```
Hive>create external table external- tab(empid int, ename string, esal double)
row format delimited fields
Terminated by ‘f’ lines terminated by ‘n’ stored as text file location
‘userRameshHive-external’;
```

A location will be automatically created.

Loading data into External Tables:-

- Loading data from HDFS to

```
Hive>load data in path'/Ramesh/input data.txt' into table external-tab;
```

- Flow of Data in Hive process at the sample location
- If we delete the managed table, both the schema and the data file will be deleted.
- But, if we delete external tables, only the schema will be deleted and data file will be there in the specified location.

Difference between managed tables & External Tables:

One of the main differences between managed and external tables in Hive is that when an external table is dropped, the data associated with it does not get deleted from only the metadata (no. of cols, types of cols, terminators etc.) gets dropped form the Hive metastore

- When a managed table gets dropped, both the metadata and data get dropped.
- Making table external because if the schema of Hive table changes, we can just drop the external table and recreate another external table over the same HDFS data with the new schema

```
Hive>Create external table log in for tab(log id int, log Error string,Log error count int)
row format delimited fields
terminated by’f’ stored as text file location ‘user/external location’;
Hive>select*from log in for tab;
```

We get the result from the file which we specified in the location path

- For external tables, no need to load the data explicitly.
- However, most of the changes to the schema can now be made through ALTER TABLE or similar command
- So, the recommendation to use external tables over managed tables might be more of a legacy concern than a contemporary one.

Altering Table:

- Most table properties can be altered with the ALTER TABLE statement, which change metadata about the table but not the table itself
- ALTER TABLE modifies table meta data on.
- Then statements can be used to fix mistakes in schema, move partition locations and do other operations.

Renaming a Table:

- This statement is used to rename the table Log_messages to log msgs

```
Cmd: ALTER TABLE log _ messages RENAME To logmsgs;
```

Changing columns

You can rename a column, change its position, type or comment.

Syntax:

ALTER TABLE log-messages CHANCE COLUMN hms hours-minutes-

Seconds INT COMMENT ’The hours, minutes and seconds are part of the times tamp’ AFTER Severity;

- You have to specify the old name, a new name and the type even if the name or type is not changed.
- If you are not moving the column, the AFTER other – column close is not necessary.
- In the example shown, we move the column after the column
- If you want to move the column to the first position, use FIRST instead of AFTER other – column.

Adding Columns

You can add new columns to the end of the existing columns, before any partition.

```
Example: ALTER TABLE Log-message ADD COLUMNS(app-name String
COMMENT”
Application Name” ,session-id long);
```

Deleting or replacing columns:

The replace statement can only be used with tables that use one of the native ser De modules are Dynamic Ser De or Metadata Type column set ser De.

- Ser De determines how records are parsed into columns i.e deserialization and how records columns are stored (serialization)

```
Ex:-ALTER TABLE log- messages REPLACE COLUMNS(
Hours-mins-sees INT
Severity STRING
Message String);
```

This statement effectively renames the original hms column and removes the server and process – id columns from the original schema definition.

As for all the ALTER Statements, only the table metadata is changed.

Partitioning and Bucketing:

Hive organizes tables into partitions, a way of dividing a table into course – grained parts based on the value of a partition column, such as date.

- Using partition can make it faster to do queries on slices of the data.
- Tables or partitions may further be sub divided into buckets, to give extra structure to the data that may be used for more efficient queries.
- For example, bucketing by user ID means we can quickly evaluate a user based query by running if on a randomized sample of the total set of users.

Partitions:

- A table may be partitioned in multiple dimensions.
- For example, in addition to partitioning logs by date, we might also subpartition each date partition by country to permit efficient queries by location.
- Partitioned are defined at table creation time using the PARTITIONED by the clause, which takes a list of column definitions.
- If we want to search a large amount of data, then we can divide the large data into partitions.

Example:

```
hive>create table party table(loaded int, log error string)
PARTITIONED BY (Logdt string, country string) row format delimited field
      terminated by ‘t’ lines terminated by ‘n’ stored as text file
```

Partition is one of the concepts in Hive where exactly certain things are grouped by the means of column combination.

Buckets:

There are two reasons why you might want to organize your tables (or partitions) into buckets.

1. The first is to enable more efficient queries.
2. The second reason to bucket a table is to make sampling more efficient.

Example:-

Tell Hive that a table should be bucketed. And we use the CLUSTERED By clause to specify the columns to a bucket and the number of buckets

```
hive>CREATE TABLE bucketed users(id INT, name STRINA)
CLUSTERED BY (id)INTO 4 BUCKETS;
```

Here we are using the user ID to determine the bucket the Hive does which is done by hashing the value and reducing module and the number of buckets, so any particular bucket will effectively have a random set of users in it.

- The data within a bucket may additionally be stored by one or more columns.
- This allows even more efficient map-side joins since the join of each bucket becomes an efficient merge sort.

Syntax for delving that a table has sorted buckets is:

```
have>CREATE TABLE bucketed users(id INT, name STRING)
CLUSTERED By(id)SORTED By(id ASC)INTO 4 BUCKETS;
```