Big Data Storage and Processing

MSc in Data Analytics

CCT College Dublin

Apache Hive

Week 10

Lecturer: Dr. Muhammad Iqbal*

Email: miqbal@cct.ie

Agenda

- Introduction to HIVE
- Data Hierarchy and HiveQL
- Primitive and Complex Data Types
- HiveQL Limitations
- Hive Metastore and Compiler
- Hive Warehouse and Schema
- Partitioning and Bucketing
- Inserting Data During Table Creation
- Relational Operators

Introduction to HIVE

- Developed by two Engineers at Facebook and a top-level Apache project because they would need to write quite complex MapReduce jobs.
- This technology was developed for already skilled staff using SQL to support their ability for further working on the Hadoop framework.
- Facebook manages the Hive-Hadoop cluster, which has more than 3PB of data in storage and loads about 16 TB of data each day. Multiple companies such as IBM, Amazon, Yahoo, Netflix are using hive and developing it continually.
- Immediately makes data on a cluster available to non-Java programmers via SQL like queries.
- Built on HiveQL (HQL), which is similar to a SQL-like query language.
- Enables easy data summarization, ad-hoc reporting and querying, and analysis of large volumes of data.

Introduction to HIVE

- Hadoop was first released by Apache in 2011 as Version 1.0.0, which only contained HDFS and MapReduce.

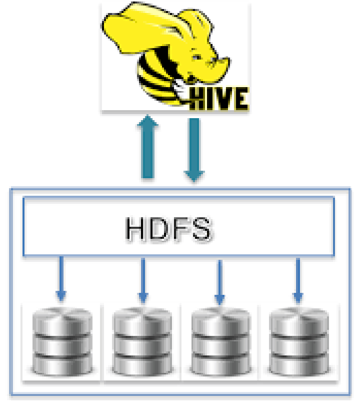
- Hadoop was designed as both a computing (MapReduce) and storage (HDFS) platform from the very beginning.

- With the increasing need for big data analysis, Hadoop attracts lots of other software to resolve big data questions and merges into a Hadoop-centric big data ecosystem.

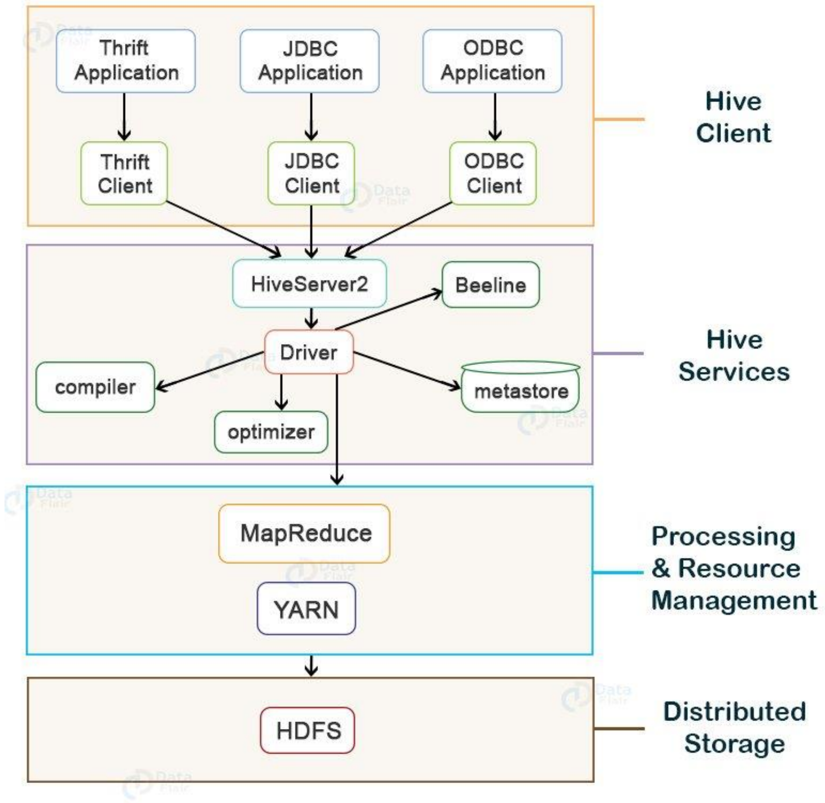
- In addition, Hive over Spark/Tez along with Live Long And Process (LLAP) offers users the ability to run a query in long-lived processes on different computing frameworks, rather than MapReduce, with in-memory data caching.

- Hive was created primarily to work with enormous data sets kept in distributed file systems like HDFS used by Hadoop. The scale and volume of big data, on the other hand, are often too much for SQL databases to handle.

What Hive is Not



- Hive, like Hadoop, is designed for batch processing of large datasets
- Interprets HiveQL and generates MapReduce jobs that run on the cluster.
- Not an OLTP or real-time system
- Latency and throughput are both high compared to a traditional RDBMS
  - Even when dealing with relatively small data (< 100 MB)



Hive Architecture & Its Components

Data Hierarchy

- Hive is organised hierarchically into
  - **Databases:** namespaces that separate tables and other objects
  - **Tables:** homogeneous units of data with the same schema
    - Analogous to tables in an RDBMS
  - **Partitions:** determine how the data is stored
    - Allow efficient access to subsets of the data
  - **Buckets/ Clusters**
    - For sub-sampling within a partition
    - Join optimization

Hive Query Language (HiveQL)

- **HiveQL / HQL** provides the basic **SQL-like** operations
  - Select columns using **SELECT**
  - Filter rows using **WHERE**
  - **JOIN** between tables
  - Evaluate aggregates using **GROUP BY**
  - Store query results into another table
  - Download results to a local directory (i.e., export from **HDFS**)
  - Manage tables and queries with **CREATE, DROP**, and **ALTER**

Primitive and Complex Data Types

Type	Comments
TINYINT, SMALLINT, INT, BIGINT	1, 2, 4 and 8-byte integers
BOOLEAN	TRUE/FALSE
FLOAT, DOUBLE	Single and double precision real numbers
STRING	Character string
TIMESTAMP	Unix-epoch offset *or* datetime string
DECIMAL	Arbitrary-precision decimal
BINARY	Opaque; ignore these bytes

Primitive Data Types

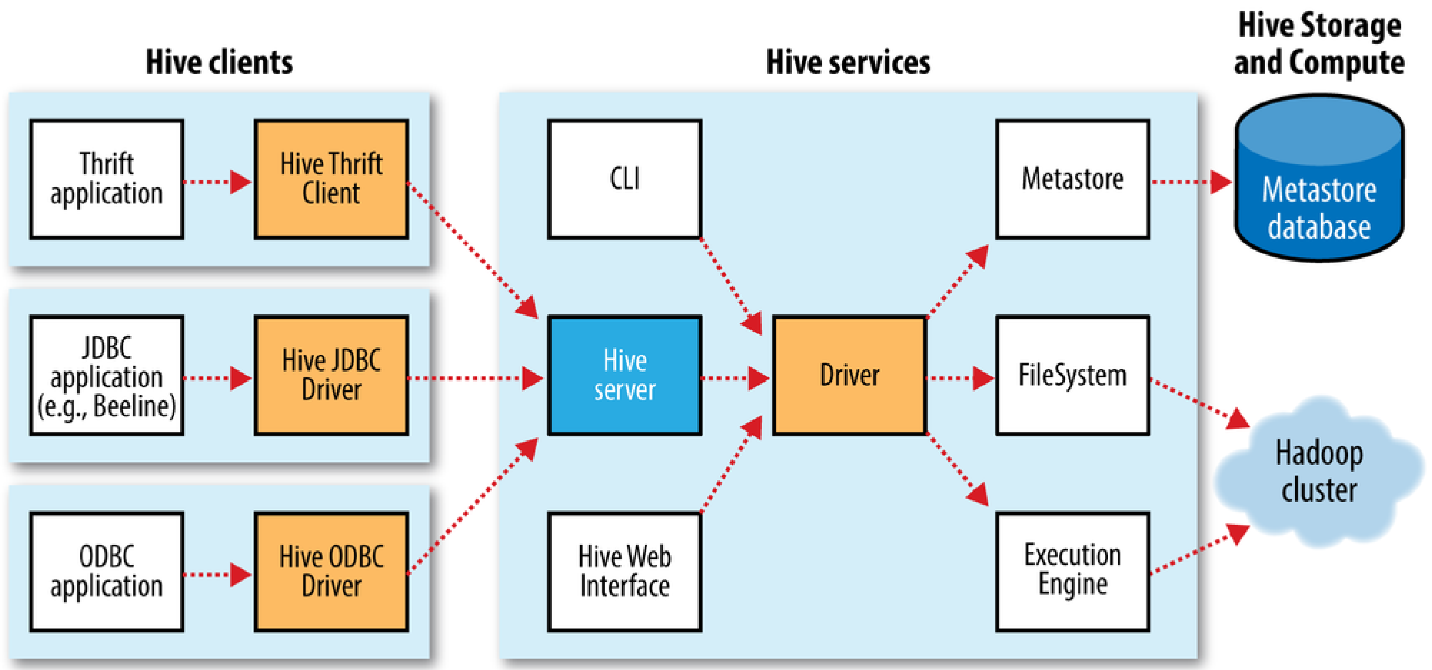
Complex Data Types

Type	Comments
STRUCT	A collection of elements If S is of type STRUCT {a INT, b INT}: S.a returns element a
MAP	Key-value tuple If M is a map from 'group' to GID: M['group'] returns value of GID
ARRAY	Indexed list If A is an array of elements ['a','b','c']: A[0] returns 'a'

HiveQL Limitations

- **HQL** only supports equi-joins, outer joins, left semi-joins
- Because it is only a ***shell for MapReduce***, complex queries can be hard to optimise
- Missing large parts of full **SQL** specification:
  - **HAVING** clause in **SELECT**
  - Correlated sub-queries
  - Sub-queries outside **FROM** clauses
  - Updatable or materialized views
  - Stored procedures

Hive Architecture



A **metastore** in Apache Hive is a central repository that contains metadata about the data kept in Hive tables. It is frequently referred to as the Hive Metastore. The metadata has details required for managing and querying the data, including as table schemas, column names, data types, file locations, and others.

Hive Architecture

- **Thrift Client**
- ***HiveServer*** is an optional service that allows a remote client to submit requests to Hive, using a variety of programming languages, and retrieve results. ***HiveServer*** is built on Apache ThriftTM (http://thrift.apache.org/), therefore it is called the ***Thrift server*** although this can lead to confusion because a newer service named HiveServer2 is also built on Thrift.
- **JDBC driver**
- Hive provides a **JDBC driver**, defined in the class org.apache.hadoop.hive.jdbc.HiveDriver. When configured with a **JDBC** URI of the form jdbc:hive2://host:port/dbname, a Java application will connect to a Hive server running in a separate process at the given host and port. In this mode, Hive runs in the same JVM as the application invoking it.
- The **Beeline CLI** uses the ***JDBC*** driver to communicate with **Hive**.
- **ODBC driver**
- An ***ODBC*** driver allows applications that support the ***ODBC*** protocol (such as business intelligence software) to connect to Hive. The Apache Hive distribution does not ship with an ***ODBC driver***, but several vendors make one freely available.

Hive Metastore

- Stores Hive metadata
- Default metastore database uses **Apache Derby** due to simplicity, no external dependencies and quick start.
- Hive provided the facility to use MySQL, PostgreSQL, Oracle, and Microsoft SQL Server
- **Various configurations:**
  - **Embedded** (in-process metastore, in-process database)
    - Mainly for unit tests
  - **Local** (in-process metastore, out-of-process database)
    - Each Hive client connects to the metastore directly
  - **Remote** (out-of-process metastore, out-of-process database)
    - Each Hive client connects to a metastore server, which connects to the metadata database itself

Hive Compiler

- The following task is performed during the compilation phase of a query.
- **Parser:** This stage transforms the query string into a parse tree.
- **Semantic Analyser:** Column names, type checking, implicit conversion type, and expression verification are all carried out in this complier stage. To determine whether a partition is required in the partition table scenario, the expression details are gathered.
- **Logical Plan Generator:** The query is converted into a logical plan which includes the relational algebra operators such as "join", "filter, and so on. In this plan, the optimizer transforms the plan to tune the performance.
- **Query Plan Generator:** The logical plan is converted into the Map-Reduce task that is further submitted on Hadoop HDFS.

Hive Warehouse and Schemas

- Hive tables are stored in the Hive "**warehouse**"
  - Default ***HDFS*** location: ***/user/hive/warehouse***
- Tables are stored as sub-directories in the warehouse directory
- Partitions are subdirectories of tables
- External tables are supported in **Hive**
- The actual data is stored in flat files

- Hive is ***schema-on-read***
  - Schema is only enforced when the data is read (at query time)
  - Allows greater flexibility: same data can be read using multiple schemas
- Contrast with an RDBMS, which is schema-on-write
  - Schema is enforced when the data is loaded
  - Speeds up queries at the expense of load times

Create Table Syntax

```
CREATE TABLE table_name
    (col1 data_type,
     col2 data_type,
     col3 data_type,
     col4 datatype )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   STORED AS format_type;
```

```
CREATE TABLE page_view
    (viewTime INT,
     userid BIGINT,
     page_url STRING,
     referrer_url STRING,
     ip STRING COMMENT 'IP Address of the User' )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   STORED AS TEXTFILE;
```

```
CREATE TABLE employees(
     (name STRING,
     salary FLOAT,
     subordinates ARRAY<STRING>,
     deductions MAP<STRING, FLOAT>,
     address STRUCT<street:STRING,
                    city:STRING,
                    state:STRING,
                    zip:INT>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   STORED AS TEXTFILE;
```

Partitioning and Bucketing

- Can make some queries faster
- Divide data based on partition column
- Use **PARTITION BY** clause when creating table
- Use **PARTITION** clause when loading data
- **SHOW PARTITIONS** will show a table's partitions
- Can speed up queries that involve sampling the data
  - Sampling works without bucketing, but Hive has to scan the entire dataset
- Use **CLUSTERED BY** when creating table
  - For sorted buckets, add **SORTED BY**
- To query a sample of your data, use **TABLESAMPLE**

https://www.cloudduggu.com/hive/partitioning/

Browsing Tables And Partitions

Command	Comments
`SHOW TABLES;`	Show all the tables in the database
`SHOW TABLES 'page.*';`	Show tables matching the specification (uses regex syntax)
`SHOW PARTITIONS page_view;`	Show the partitions of the **page_view** table
`DESCRIBE page_view;`	List columns of the table
`DESCRIBE EXTENDED page_view;`	More information on columns (useful only for debugging)
`DESCRIBE page_view PARTITION (ds='2008-10-31');`	List information about a partition

Loading Data

- Use **LOAD DATA** to load data from a file or directory
  - Will read from **HDFS** unless **LOCAL** keyword is specified
  - Will append data unless **OVERWRITE** specified
  - **PARTITION** required if destination table is partitioned

```
LOAD DATA LOCAL INPATH '/tmp/pv_2008-06-8_us.txt'
OVERWRITE INTO TABLE page_view
PARTITION (date = '2008-06-08', country = 'US')
```

Inserting Data

- Use **INSERT** to load data from a **Hive query**
  - Will append data unless **OVERWRITE** specified
  - **PARTITION** required if destination table is partitioned

```
FROM page_view_stg pvs

INSERT OVERWRITE TABLE page_view

PARTITION (dt = '2008-06-08', country = 'US′)

SELECT pvs.viewTime, pvs.userid, pvs.page_url,
pvs.referrer_url

WHERE pvs.country = 'US';
```

Inserting Data During Table Creation

- Use **AS SELECT** in the **CREATE TABLE** statement to populate a table as it is created

```
CREATE TABLE page_view AS
 SELECT pvs.viewTime, pvs.userid, pvs.page_url,
 pvs.referrer_url
 FROM page_view_stg pvs
 WHERE pvs.country = 'US';
```

Sample Select Clauses

- Select from a single table

```
    SELECT *
        FROM sales
        WHERE amount > 10 AND
              region = "US";
```

- Select from a partitioned table

```
SELECT page_views.*
 FROM page_views
 WHERE page_views.date >= '2008-03-01' AND
       page_views.date <= '2008-03-31';
```

Relational Operators

- **ALL and DISTINCT**
  - Specify whether duplicate rows should be returned
  - **ALL** is the default (all matching rows are returned)
  - **DISTINCT** removes duplicate rows from the result set
- **WHERE**
  - Filters by expression
  - Does not support **IN, EXISTS** or sub-queries in the **WHERE** clause
- **LIMIT**
  - Indicates the number of rows to be returned

Relational Operators

- **GROUP BY**
  - Group data by column values
  - Select statement can only include columns included in the **GROUP BY** clause
- **ORDER BY / SORT BY**
  - **ORDER BY** performs total ordering
    - Slow, poor performance
  - **SORT BY** performs partial ordering
    - Sorts output from each reducer

Resources/ References

- Lecture is prepared based on the slides provided by Hadoop-Based Distributed Computing, Adam Shook, 2016. https://www.csee.umbc.edu/~shadam1/491s16/lectures/07-Hive.pptx
- http://hive.apache.org
- https://www.cloudduggu.com/hive
- Some images are used from Google search repository (https://www.google.ie/search) to enhance the level of learning.