APACHE HIVE OVERVIEW

Apache Hive Overview

- **Apache Hive Overview**
- Hive Architecture
- Hive Tables
- HiveQL
- Hive Data Types
- Hive Operators
- Built-in Functions

Apache Hive Overview

- Apache Hive is the Hadoop's data warehouse software
- On distributed datasets, Hive provides a mechanism for:
  - Projecting structure on data
  - Querying data with an SQL-like language (HiveQL)
- The HiveQL queries submitted by the clients are converted into MapReduce jobs, and run on Hadoop clusters

Apache Hive Overview

- Apache Hive greatly increases the accessibility to Apache Hadoop, since:
  - Non-developers can write SQL-like queries and analyze large datasets without writing MapReduce code
  - Traditional analytics software can connect to a Hadoop cluster via its ODBC/JDBC interface

Apache Hive Overview

- Apache Hive Overview
- **Hive Architecture**
- Hive Tables
- HiveQL
- Hive Data Types
- Hive Operators
- Built-in Functions

HiveServer2

- In Hive, the server component to which the clients connect and submit their queries to is called the HiveServer (**HiveServer2**, to be precise)
- It implements a Thrift-based RPC interface that handles concurrent requests
- It is also the container for the Hive Execution Engine, which compiles the submitted queries from the client into a series of MapReduce jobs, and runs the jobs

Table Metadata

- When we talk about a Hive table, we are referring to metadata for querying underlying distributed datasets
- To interpret and query a distributed dataset, while Hive is converting SQL statements into MapReduce jobs, it needs to know:
  - The table schema (table name, column names and types, …)
  - Table partition information (we will come to that later)
  - Storage format of the underlying data (*Data is in text files, each line representing a database row*, for example)
  - Row interpretation (*In each line, columns are seperated with commas*, for example)

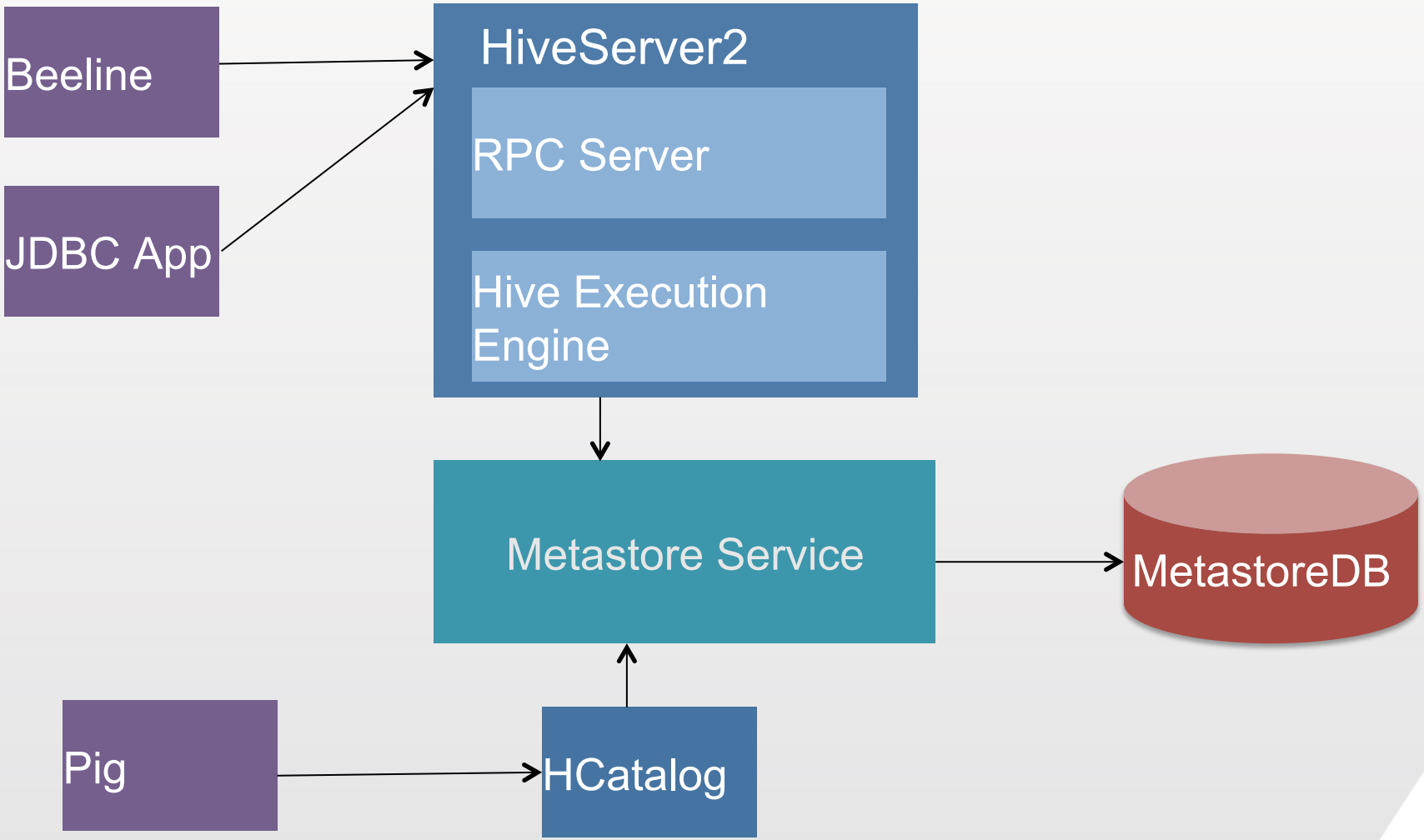
Metastore

- Such metadata is stored in an actual relational database, and its called the **Metastore DB**.
- The component of the Hive that serves metadata, to both the other components (the execution engine), and other systems (*to MepReduce and Pig via* ***HCatalog,*** for example) is called the **Metastore**

Clients

- Clients connect to the HiveServer2's Thrift RPC, via:
  - The JDBC driver (built-in in Hive)
  - The ODBC driver (not included in Hive, but many third party drivers are available)
- The default commandline client to Hive is the **Beeline CLI,** which itself is a JDBC client

Hive Architecture



Hive Architecture

- The MetastoreDB, Metastore Service, and HiveServer2 can be run on the same host, but the recommended practice is that the HiveServer2 component lives in a different host
- These services may or may not be installed in a node in the Hadoop cluster, but certainly, the HiveServer should be able to submit M/R jobs

Apache Hive Overview

- Apache Hive Overview
- Hive Architecture
- **Hive Tables**
- HiveQL
- Hive Data Types
- Hive Operators
- Built-in Functions

Hive Tables

- A Hive table is created by a **CREATE TABLE** statement
- When it is being queried, MapReduce jobs are run on the dataset that is located in HDFS, denoted in the **LOCATION** element of the table
- The **LOCATION**, if not specified, is by default set in the **`hive.metastore.warehouse.dir`** configuration property, which is by default the HDFS path /user/hive/warehouse

Hive Tables

- A CREATE TABLE statement looks like the following:

```
$ hive

hive> CREATE TABLE users (
        id INT,
        name STRING,
        email STRING)
        ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ‘,’;

hive> OK
```

Hive Tables

- We need to tell Hive
  - The storage format of the underlying data (*Data is in text files, each line representing a database row*, for example)
  - And the row interpretation (*In each line, columns are seperated with commas*, for example)
- This is what the

```
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,’
```

  statement is for
- The storage format ‘TEXTFILE’ is by default

Hive Tables

- Loading data into a Hive table can be done in several ways:
  - By manually copying or moving data (in the same format we described while creating the table) into the directory denoted in the table's LOCATION property
  - By using the *LOAD DATA INPATH <hdfs-path> INTO TABLE <table-name>* statement
    - This is equivalent to manually moving/copying data.
    - Either move or copy operation is performed depending on how table is created, *CREATE TABLE* or *CREATE EXTERNAL TABLE*, respectively
  - By using the *LOAD DATA LOCAL INPATH <local-path> INTO TABLE <table-name>* statement

Hive Tables

- Loading data into a Hive table can be done in several ways:
  - At the create time using the CREATE TABLE … AS SELECT … FROM … statement
    - Data is selected from another Hive table, and written into the directory denoted in the created table's LOCATION property
    - Data is written in the appropriate format defined in the CREATE statement in the new table
  - By inserting data from a query result, for example:
    - *INSERT OVERWRITE TABLE <t1> select_statement*
    - *INSERT INTO TABLE <t1> select_statement*
    - Again, data is written in the appropriate format

Demo

Creating and Populating a Hive Table

Apache Hive Overview

- Apache Hive Overview
- Hive Architecture
- Hive Tables
- **HiveQL**
- Hive Data Types
- Hive Operators
- Built-in Functions

HiveQL

- HiveQL is Hive's query language, very close to MySQL's dialect of ANSI SQL standard
- It doesn't allow row-level INSERTs, UPDATEs, and DELETEs (all of them are now supported [starting from Hive version 0.14], and they are useful to use in slow-changing dimension tables)
  - For UPDATE and DELETE to be performed, the table should be marked as 'transactional'

HiveQL: DDL Statements

- A tour of Hive DDL Statements:
  - **CREATE** DATABASE/SCHEMA, TABLE, VIEW, FUNCTION, INDEX
  - **DROP** DATABASE/SCHEMA, TABLE, VIEW, INDEX
  - **TRUNCATE** TABLE

HiveQL: DDL Statements

- A tour of Hive DDL Statements:
  - **ALTER** DATABASE/SCHEMA, TABLE, VIEW
  - **SHOW** DATABASES/SCHEMAS, TABLES, TBLPROPERTIES, PARTITIONS, FUNCTIONS, INDEX[ES], COLUMNS, CREATE TABLE
  - **DESCRIBE** DATABASE/SCHEMA, table_name, view_name

HiveQL: DML Statements

- A tour of Hive DML Statements:
  - **LOAD DATA [LOCAL] INPATH … INTO** TABLE …
  - **INSERT OVERWRITE** TABLE, DIRECTORY
  - **INSERT INTO** TABLE … **SELECT … FROM**
  - **FROM …**

    INSERT INTO TABLE … SELECT

    INSERT INTO TABLE … SELECT
  - **FROM …**

    INSERT OVERWRITE TABLE/DIRECTORY … SELECT

    INSERT OVERWRITE TABLE/DIRECTORY … SELECT

HiveQL: Data Retrieval

- A **`SELECT`** statement, from a table reference (which can be a table, a subquery, a view, or a join construct), is used for data retrieval

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[CLUSTER BY col_list
  | [DISTRIBUTE BY col_list] [SORT BY col_list]
]
[LIMIT number]
```

HiveQL: Data Retrieval

- A **`SELECT`** statement can be a part of a union query

```
select_statement UNION [ALL | DISTINCT] select_statement
UNION [ALL | DISTINCT] select_statement ...
```

HiveQL: Data Retrieval

- A **`SELECT`** statement can be a part of a subquery

```
SELECT ... FROM (SELECT ... FROM ...) [AS] name ...
```

```
SELECT *
FROM A
WHERE A.a IN (SELECT foo FROM B);
```

```
SELECT A
FROM T1
WHERE EXISTS (SELECT B FROM T2 WHERE T1.X = T2.Y)
```

HiveQL: Data Retrieval

- A **`SELECT`** statement can take regex-based column specification

```
SELECT `(ds|hr)?+.+` FROM sales
```

HiveQL: Data Retrieval

- Hive supports
  - **`GROUP BY`**
  - **`ORDER BY`**
  - **`JOIN`**
  - **`UNION`**
  - **`TABLESAMPLE`**
  - Subqueries
  - Windowing, **`OVER`**, and Analytics
  - **`LATERAL VIEW`**
  - UDFs

HiveQL: Data Retrieval

- Hive also supports passing custom (Hadoop Streaming) Map and Reduce functions, using the **`TRANSFORM`**, **`MAP`**, and **`REDUCE`** clauses:

```
FROM (
  FROM t1
  MAP t1.c1, t1.c2
  USING 'map_script'
  AS f1, f2
  CLUSTER BY f1) map_output
INSERT OVERWRITE TABLE reduce_output
  REDUCE map_output.f1, map_output.f2
  USING 'reduce_script'
  AS k1, k2;
```

Apache Hive Overview

- Apache Hive Overview
- Hive Architecture
- Hive Tables
- HiveQL
- **Hive Data Types**
- Hive Operators
- Built-in Functions

Hive Data Types

- Hive supports some primitive and complex types
- Integers can be represented as TINYINTs, SMALLINTs, INTs, BIGINTs
- BOOLEAN type is supported
- FLOAT and DOUBLE are supported
- STRING type is supported

Hive Data Types

- Hive supports some primitive and complex types
- Complex types are composite collections, and can be:
  - **Struct**s;
    - if a column c is a `STRUCT{a:INT, b:INT}`, `a` can be accessed with `c.a`
    - A Struct column can be constructed using the `struct(val1, val2, …)` or `named_struct(name1, val1, name2, val2, …)` built-in functions

Hive Data Types

- Hive supports some primitive and complex types
- Complex types are composite collections, and can be:
  - **Map**s;
    - Maps are collections of key-value tuples
    - If a map `M` contains the mapping `'gid'->gnum`, `gnum` can be retrieved with `M['gid']`
    - A map can be constructed using the `map(key1, value1, key2, value2, …)` built-in function

Hive Data Types

- Hive supports some primitive and complex types
- Complex types are composite collections, and can be:
  - **Array**s;
    - An array is a collection of elements of the same type
    - An element of an array `A` can be accessed using the `A[i]` notation, where `i` is an integer, denoting the index of interest
    - An array can be constructed using the `array(element1, element2, …)` built-in function

Apache Hive Overview

- Apache Hive Overview
- Hive Architecture
- Hive Tables
- HiveQL
- Hive Data Types
- **Hive Operators**
- Built-in Functions

Hive Operators

- Hive supports comparison, arithmetic and logical operators
  - Some comparison operators supported by Hive:
    - `A IS [NOT] NULL`
    - `A [NOT] LIKE B`
    - `A RLIKE B`
    - `A>B`
    - `A<>B or A!=B`
    - `A = B`
    - `A <=> B` (same as equal, but returns true if both A and B are NULL)
    - `A [NOT] BETWEEN B AND C`

Hive Operators

- Hive supports comparison, arithmetic and logical operators
  - Some arithmetic operators supported by Hive:
    - `A + B` (arithmetic)
    - `A % B` (modulo)
    - `A & B` (bitwise operations)

Hive Operators

- Hive supports comparison, arithmetic and logical operators
  - Some logical operators supported by Hive:
    - `A AND B, A&&B`
    - `A OR B, A||B`
    - `NOT A, !A`
    - `A [NOT] IN (val1, val2, …)`
    - `[NOT] EXISTS(subquery)`

Apache Hive Overview

- Apache Hive Overview
- Hive Architecture
- Hive Tables
- HiveQL
- Hive Data Types
- Hive Operators
- **Built-in Functions**

Built-in Functions

- Hive supports many **mathematical**, **date**, **boolean**, and **String** functions
- A complete list of functions can be listed using

```
$ hive

hive> SHOW FUNCTIONS;
```

- A function can be documented using

```
$ hive

hive> DESCRIBE FUNCTION [EXTENDED] <function_name>;
```

Built-in Functions

- A function can return a value with a different type than its argument
  - e.g. **`length(s)`** returns an int, where s is a string
- A function can return an aggregate of a collection of values (or a column) (built-in aggregate functions)
  - e.g. **`count(C), min(C), corr(c1, c2)`**

Built-in Functions

- A function can generate multiple rows (built-in table generating function)
  - e.g. **`explode(arr)`** returns N rows, where N is the number of elements in the array argument **`arr`**
  - e.g. **`stack(n, v_1, v_2, …, v_k)`** breaks up **`v_1, v_2, … v_k`** into n rows, each of which having k/n columns

Demo

Running Hive Queries

Apache Hive Overview

End of Chapter