Apache Hive

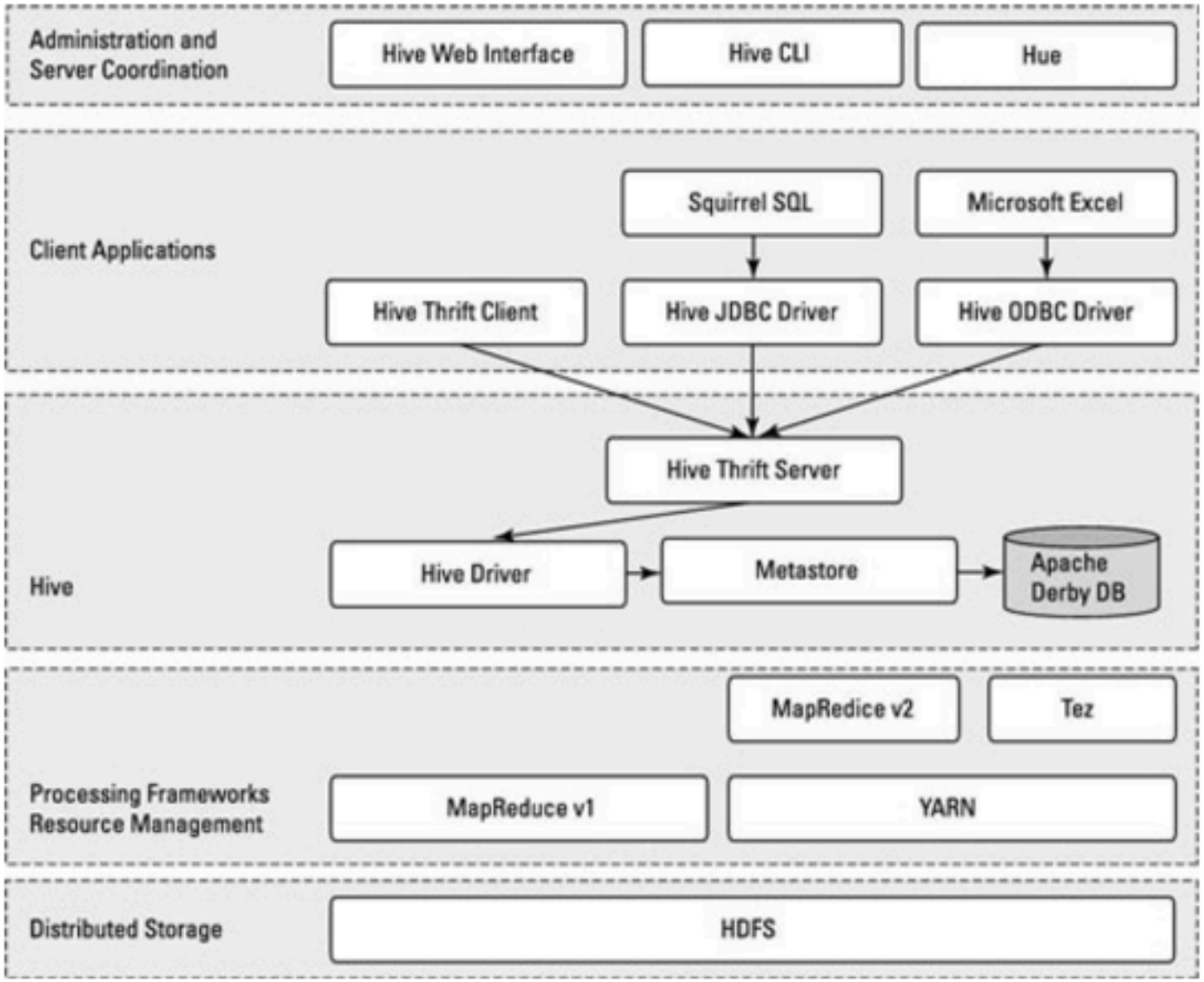
Hive

- Hive is a data warehousing infrastructure based on Hadoop.
- Hive is designed to enable easy data summarization, ad-hoc querying and analysis of large volumes of data.
- Provides a very simple query language called Hive QL, which is based on SQL.
- Supports map-reduce capabilities.

Hive

- Hive runs on top of HDFS
  => High Latency
  => Cannot perform OLTP (Online Transaction Processing)
  => No real time queries
  => Not 100% available and no ACID compliance.

Hive Architecture



Hive Driver compiles, optimizes, and executes the HiveQL.
- can be in local or MR mode

Hive Thrift Server - enables a rich set of clients to access the Hive subsystem.

The Hive Driver stores table metadata in the metastore and its database.

Command Line Interface (CLI) - you can use a Linux terminal window to issue queries and administrative commands directly to the Hive Driver

Data Units in Hive

Data units in order of granularity in Hive:

- Databases: Namespaces that separate tables and other data units from naming confliction.
- Tables: Homogeneous units of data which have the same schema.
- Partitions: Each Table can have one or more partition Keys which determines how the data is stored
- 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

Hive Types

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
- String type
  - STRING - sequence of characters in a specified character set

Hive Types

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'.

Hive QL

- Very similar to SQL.
- Create Table:

```
CREATE TABLE IF NOT EXISTS mydb.employees (
  name         STRING COMMENT 'Employee name',
  salary       FLOAT  COMMENT 'Employee salary',
  subordinates ARRAY<STRING> COMMENT 'Names of subordinates',
  deductions   MAP<STRING, FLOAT>
               COMMENT 'Keys are deductions names, values are percentages',
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
               COMMENT 'Home address')
COMMENT 'Description of the table'
TBLPROPERTIES ('creator'='me', 'created_at'='2012-01-02 10:00:00', ...)
LOCATION '/user/hive/warehouse/mydb.db/employees';
```

Hive QL

- Create External Table (data located in external file)

```
CREATE EXTERNAL TABLE IF NOT EXISTS stocks (
  exchange        STRING,
  symbol          STRING,
  ymd             STRING,
  price_open      FLOAT,
  price_high      FLOAT,
  price_low       FLOAT,
  price_close     FLOAT,
  volume          INT,
  price_adj_close FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/data/stocks';
```

Creating Partitioned Tables

- Horizontal partitions based on key values.
- Example:

```
CREATE TABLE employees (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
PARTITIONED BY (country STRING, state STRING);
```

Creating Partitioned Tables

- There will be a directory for the table:

```
hdfs://master_server/user/hive/warehouse/mydb.db/employees
```

- There will be subdirectories for the partitions:

```
...
.../employees/country=CA/state=AB
.../employees/country=CA/state=BC
...
.../employees/country=US/state=AL
.../employees/country=US/state=AK
...
```

Hive QL

- Create Table:

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;
```

Hive QL

- How to specify field delimiter:

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY '1'
STORED AS SEQUENCEFILE;
```

Create Table and Store in external file

- Create table and store in an external file

```
CREATE EXTERNAL TABLE page_view_stg(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User',
                country STRING COMMENT 'country of origination')
COMMENT 'This is the staging page view table'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '44' LINES TERMINATED BY '12'
STORED AS TEXTFILE
LOCATION '/user/data/staging/page_view';

hadoop dfs -put /tmp/pv_2008-06-08.txt /user/data/staging/page_view

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, null, null, pvs.ip
WHERE pvs.country = 'US';
```

Load data from external file

- You can load data from local file or HDFS file:

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

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

Local signifies local file system, if it is omitted then it looks for file on HDFS

Load / Export data

- You can load data from local file or HDFS file:

```
LOAD DATA LOCAL INPATH '${env:HOME}/california-employees'
OVERWRITE INTO TABLE employees
PARTITION (country = 'US', state = 'CA');
```

- Export data from table to file:

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/ca_employees'
SELECT name, salary, address
FROM employees
WHERE se.state = 'CA';
```

Export data

- Can create local or HDFS files or tables.

```
hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a;
hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a WHERE a.key < 100;
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/reg_3' SELECT a.* FROM events a;
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_4' select a.invites, a.pokes FROM profiles a;
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT COUNT(*) FROM invites a WHERE a.ds='2008-
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT a.foo, a.bar FROM invites a;
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/sum' SELECT SUM(a.pc) FROM pc1 a;
```

Hive Queries

- Suppose table is:

```
CREATE TABLE employees (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
PARTITIONED BY (country STRING, state STRING);
```

Hive Queries

- Query involving arrays

```
hive> SELECT name, subordinates FROM employees;
John Doe     ["Mary Smith","Todd Jones"]
Mary Smith   ["Bill King"]
Todd Jones   []
Bill King    []
```

Hive Queries

- Query involving maps

```
hive> SELECT name, deductions FROM employees;
John Doe     {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
Mary Smith   {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
Todd Jones   {"Federal Taxes":0.15,"State Taxes":0.03,"Insurance":0.1}
Bill King    {"Federal Taxes":0.15,"State Taxes":0.03,"Insurance":0.1}
```

Hive Queries

- Query involving structs

```
hive> SELECT name, address FROM employees;
John Doe    {"street":"1 Michigan Ave.","city":"Chicago","state":"IL","zip":60600}
Mary Smith  {"street":"100 Ontario St.","city":"Chicago","state":"IL","zip":60601}
Todd Jones  {"street":"200 Chicago Ave.","city":"Oak Park","state":"IL","zip":60700}
Bill King   {"street":"300 Obscure Dr.","city":"Obscuria","state":"IL","zip":60100}
```

Group By Queries

- Just like SQL

```
hive> SELECT year(ymd), avg(price_close) FROM stocks
    > WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'
    > GROUP BY year(ymd);
1984    25.578625440597534
1985    20.193676221040867
1986    32.46102808021274
1987    53.88968399108163
1988    41.540079275138766
1989    41.65976212516664
1990    37.56268799823263
1991    52.49553383386182
1992    54.80338610251119
1993    41.026719564505 72
1994    34.0813495847914
```

Group By with Having

- Just like SQL

```
hive> SELECT year(ymd), avg(price_close) FROM stocks
    > WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'
    > GROUP BY year(ymd)
        > HAVING avg(price_close) > 50.0;
1987    53.88968399108163
1991    52.49553383386182
1992    54.80338610251119
1999    57.77071460844979
2000    71.74892876261757
2005    52.401745992993554
...
```

Joins

- Similar syntax to SQL – makes life easy ☺
- Join optimization:
  Hive also assumes that the last table in the query is the largest. It attempts to buffer the other tables and then stream the last table through, while performing joins on individual records

Joins

- Similar syntax to SQL – makes life easy ☺
- Inner Join:

```
hive> SELECT a.ymd, a.price_close, b.price_close
    > FROM stocks a JOIN stocks b ON a.ymd = b.ymd
    > WHERE a.symbol = 'AAPL' AND b.symbol = 'IBM';
2010-01-04      214.01  132.45
2010-01-05      214.38  130.85
2010-01-06      210.97  130.0
2010-01-07      210.58  129.55
2010-01-08      211.98  130.85
2010-01-11      210.11  129.48
```

Joins

- Similar syntax to SQL – makes life easy ☺
- Outer Join:

```
hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend
    > FROM stocks s LEFT OUTER JOIN dividends d ON s.ymd = d.ymd AND s.symbol = d.symbol
    > WHERE s.symbol = 'AAPL';
...
1987-05-01      AAPL    80.0    NULL
1987-05-04      AAPL    79.75   NULL
1987-05-05      AAPL    80.25   NULL
1987-05-06      AAPL    80.0    NULL
1987-05-07      AAPL    80.25   NULL
1987-05-08      AAPL    79.0    NULL
1987-05-11      AAPL    77.0    0.015
1987-05-12      AAPL    75.5    NULL
1987-05-13      AAPL    78.5    NULL
1987-05-14      AAPL    79.25   NULL
1987-05-15      AAPL    78.25   NULL
1987-05-18      AAPL    75.75   NULL
1987-05-19      AAPL    73.25   NULL
1987-05-20      AAPL    74.5    NULL
...
```

Joins

- Joins involve MapReduce jobs in background.
- You should optimize as much as possible.
- EXPLAIN {query} explains you how a query works.

SQL constructs still apply

- Good tutorial of commands: https://cwiki.apache.org/confluence/display/Hive/Tutorial
- https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-InsertingvaluesintotablesfromSQL

Some Exercises

- Create table for business, review, and user.
- Remember to use proper field delimiter. ('^')
- Load data from HDFS.

Some Exercises

- List the 'user id' and 'stars' of users that reviewed businesses located in Stanford.
- List the business_id , full address and categories of the Top 10 highest rated businesses using the average ratings.
- List the user_id , and name of the top 10 users who have written the most reviews.
- List the user_id , and name of the top 5^{th} user who has written the most reviews.
  ** This is the kth value retrieval problem **
  ** Group queries in Hive can be slow **

Beeline

- Original Hive:
  - HiveServer1 is the server
  - CLI is the client
- New change:
  - HiveServer2: additional concurrency and security
  see details:
  http://blog.cloudera.com/blog/2013/07/how-hiveserver2-brings-security-and-concurrency-to-apache-hive/
  - Client: Beeline
  https://blog.cloudera.com/blog/2014/02/migrating-from-hive-cli-to-beeline-a-primer/

Beeline

- Type beeline
- Connect as:

  !connect jdbc:hive2://
  OR
  !connect jdbc:hive2://localhost:10000 scott tiger
  org.apache.hive.jdbc.HiveDriver

- Same syntax as before.

Other features of HQL

- Supports Views, Indexes, Functions, Query Tuning, etc.

Hive vs Pig

Hive	Pig
Developed at Facebook	Developed at Yahoo!
Data warehousing language	Data flow modeling language
Popular with business analysts [resembles SQL]	Popular with scripting community
Works mostly with structured data	Can handle unstructured data as well
Schema is required	Optional Schema