University of East London

Pioneering Futures Since 1898

Apache HIVE

Data Warehousing & Analytics on Hadoop

Dr Amin Karami

a.karami@uel.ac.uk

www.aminkarami.com

CN7022 – Week 5
1 November 2019

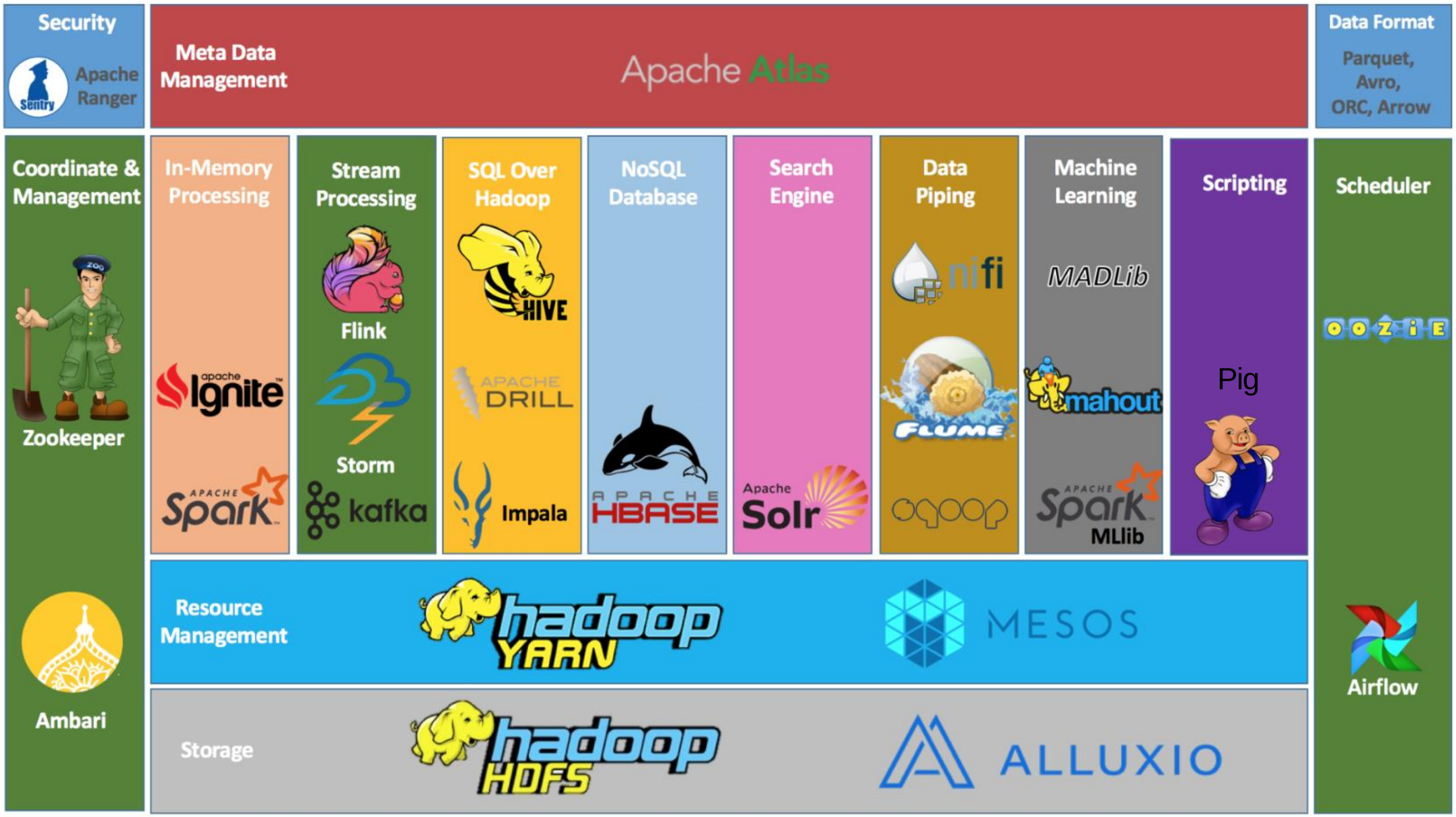
Outline

- A Review on Hadoop Ecosystem
- ZooKeeper and Oozie
- What is Apache HIVE?
- HIVE Architecture
- Partitioning in HIVE for Querying massive data
- HIVE Queries

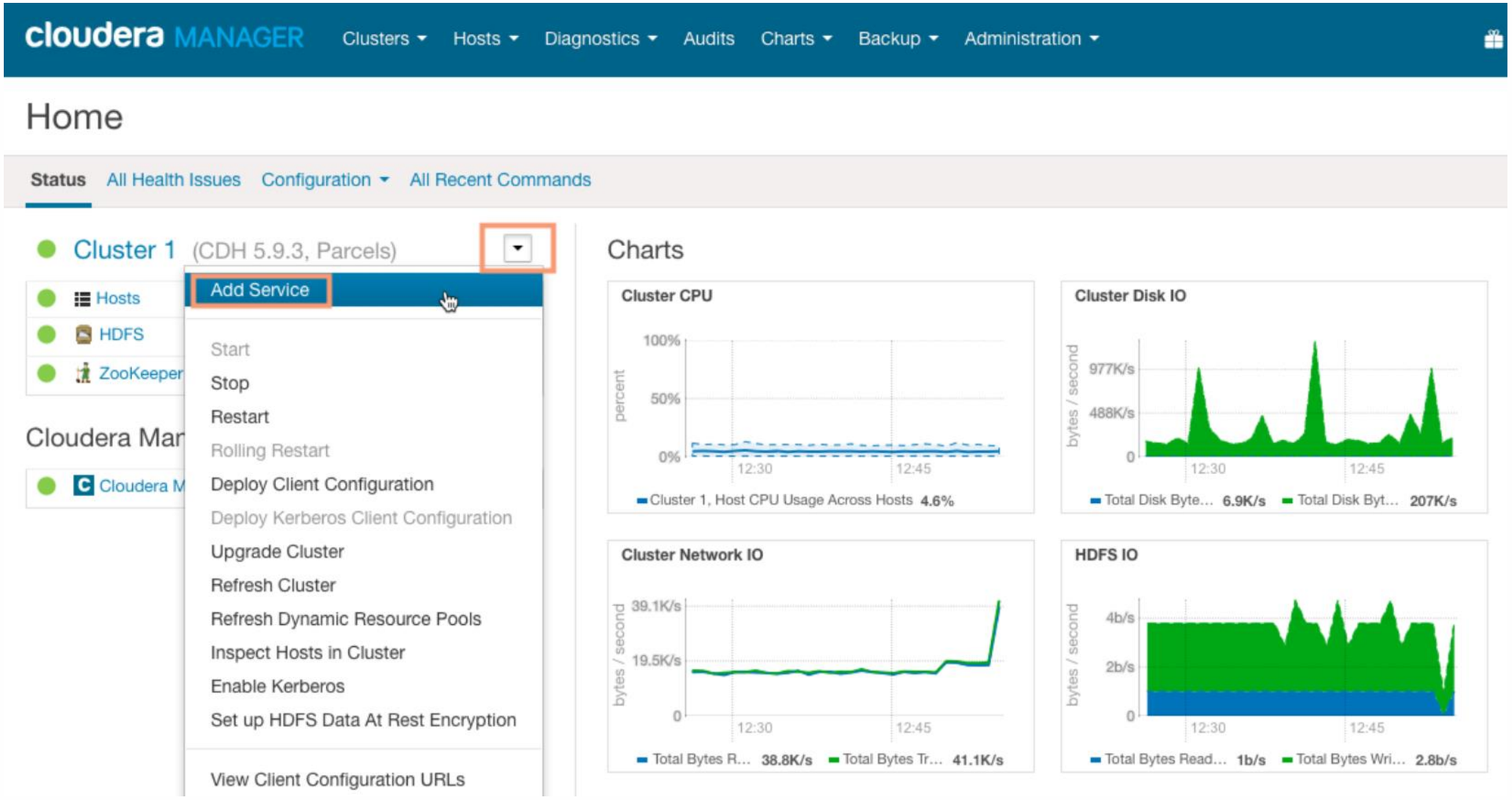
Learning Outcomes

- Be able to explain HIVE
- Understand the differences between HIVE and Impala
- Understand HIVE Architecture and its components
- Be able to write HIVE queries
- Understand optimization technique for HIVE Querying
- Understand the functionalities of Zookeeper and Oozie

Hadoop Ecosystem



Cloudera Manager



Add Service to Cluster 1

Select the type of service you want to add.

	Service Type	Description
○	Accumulo	The Apache Accumulo sorted, distributed key/value store is a robust, scalable, high performance data storage and retrieval system. This service only works with releases based on Apache Accumulo 1.6 or later.
○	Flume	Flume collects and aggregates data from almost any source into a persistent store such as HDFS.
○	HBase	Apache HBase provides random, real-time, read/write access to large data sets (requires HDFS and ZooKeeper).
○	HDFS	Apache Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute hosts throughout a cluster to enable reliable, extremely rapid computations.

What is ZooKeeper?

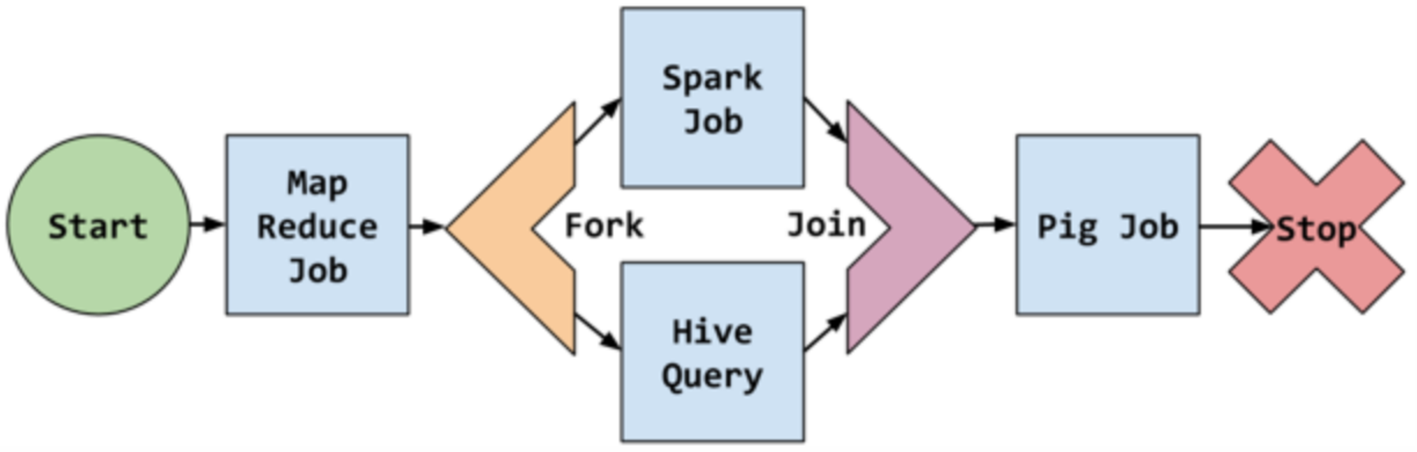
- It is a coordinator (written in Java) for your Hadoop cluster to enable highly reliable distributed coordination. (https://zookeeper.apache.org/)
- It is basically keep the track of information that must be synchronized across your cluster.
- The requirements in the distributed systems:
  1. **Networking**: Workstations can be either physically closed (LAN) or geographically distant (MAN or WAN)
  2. **Scalability**: Systems can be easily expanded by adding more machines
  3. **Redundancy & Reliability**: if a node is unavailable due to the system failure, the copy version of the machine should work
  4. **Availability**: the system must be up 100%

ZooKeeper roles

- **Maintaining Configuration** Information between nodes
- **Naming Service**: find a machine in a cluster of 1,000 nodes
- **Providing Distributed Synchronization**: deadlocks, Queues, …
- **Providing Group Services**: leader election and more
- **Timeliness**: the system availability is guaranteed up-to-date
- Example:
  - **HBase**: master selection, server lease management, bootstrapping, coordination between servers.
  - **Hadoop and MapReduce:** high availability of ResourceManager, Adaptive MapReduce, high availability of MapReduce jobs in case of failure.
  - **Flume:** configuration purposes in the recent releases

What is Oozie?

- Usually, we run several jobs for processing massive amount of data in the cluster. How do we can schedule them?
- **Oozie** is a workflow scheduler system to manage Apache Hadoop jobs (such as MapReduce, Streaming, Pig, Hive, Sqoop, HDFS operations).
- Oozie is a scalable, reliable and extensible system.
- It executes and monitors the workflows in Hadoop and performs a periodic scheduling of workflows.



Why another data warehousing?

- **Facebook's Problem 1**: Data, Data and more Data (200GB per day in March 2008, now it is **1-10TB per day**)
- **Problem 2**: MapReduce is great but every one is not a MapReduce expert. It is difficult to convert any code to key-value format.
- **Problem 3**: using Sqoop every day for importing 1-10TB data is very time consuming.
- **Problem 4**: The most people know SQL, solution?

- HIVE: A system for querying and managing structured data built on top of MapReduce and Hadoop

Why HIVE?

- **Yahoo** worked on Pig to facilitate application deployment on Hadoop, mainly on unstructured data.
- Simultaneously **Facebook** started working on deploying warehouse solutions on Hadoop that resulted in Hive.
- Facebook was thinking how to process about 1-10TB data without hardware limitations in every day? This size is rapidly growing in businesses that make traditional warehousing solution prohibitively expensive and not applicable.

What is HIVE?

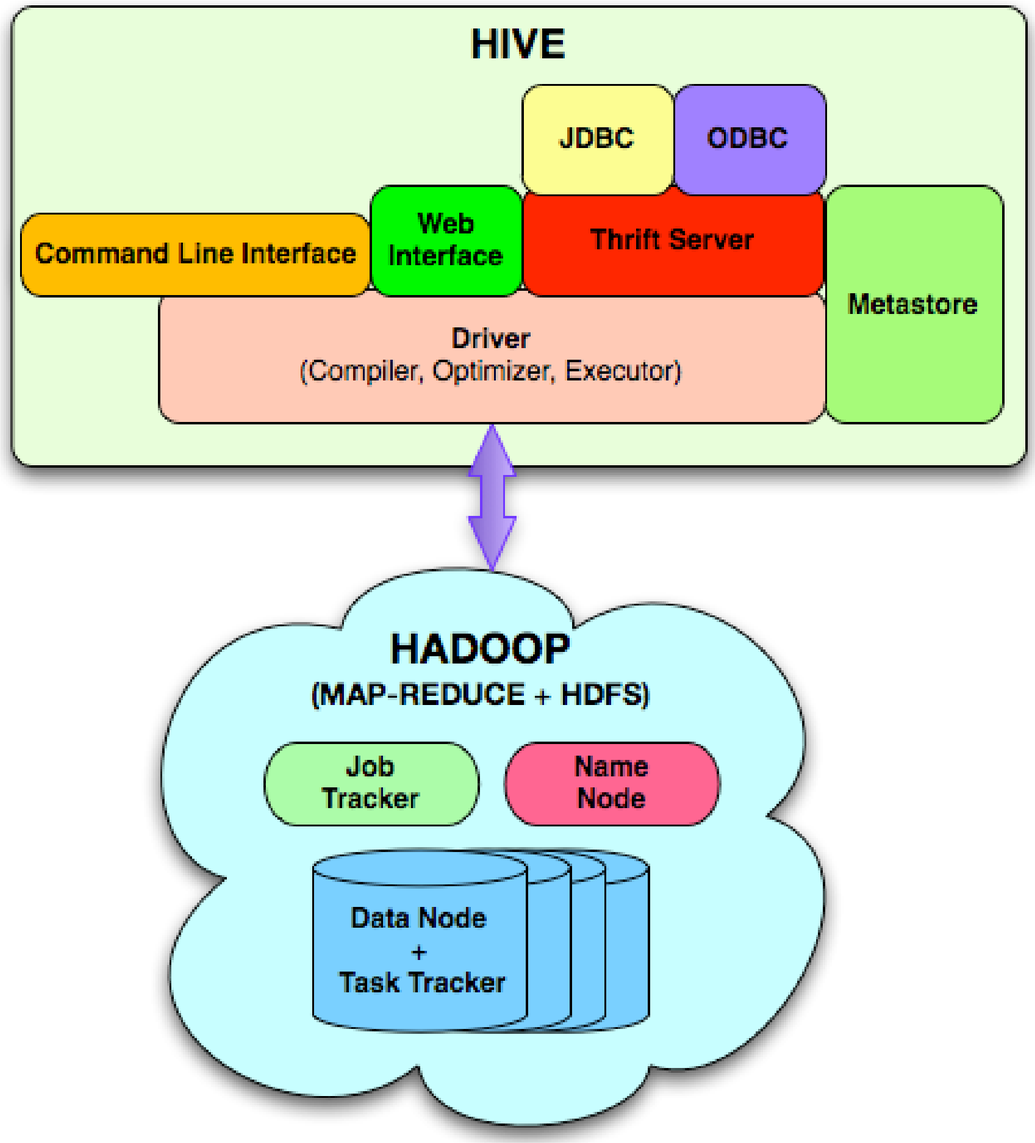
- A system for managing and querying structured data built on top of Hadoop:
  - ✓ Uses Map-Reduce for execution
  - ✓ HDFS for storage
  - ✓ Extensible to other Data Repositories
- What HIVE is **not**:
  - ✓ Not designed for OLTP, designed for data analysis to help in decision making (OLAP)
  - ✓ Does not offer real-time queries

HIVE Querying with MapReduce

- SELECT ..... FROM ..... **Map**
- WHERE ..... **Map**
- GROUP BY ..... **Shuffle & Sort** + HAVING ..... **Reduce**
- JOIN ..... **Map/Reduce**
- ORDER BY / SORT BY ..... **Reduce**

HIVE architecture

1. **Hive (Thrift) Server**: It allows different clients to submit requests to Hive.
2. **Hive Deriver**: Driver is responsible for receiving the queries submitted Thrift, JDBC, ODBC, CLI, Web UL interface by a Hive client.
   - **Complier:** After that hive driver passes the query to the compiler. Where parsing, type checking, and semantic analysis takes place with the help of schema present in the metastore.
   - **Optimizer:** It generates the optimized logical plan in the form of a DAG (Directed Acyclic Graph) of MapReduce and HDFS tasks.
   - **Executor:** Once compilation and optimization complete, execution engine executes these tasks in the order of their dependencies using Hadoop.
3. **Metastore:** it is the central repository of Apache Hive metadata in the Hive Architecture. It stores metadata for Hive tables (like their schema and location) and partitions in a relational database. It provides client access to this information by using metastore service API.



WordCount() with HIVE

1.Create a table in Hive:

```
create table feedback(comments string);
```

2.Load Data:

```
load data local inpath '<feedback wordcount.txt>' into table
feedback;
```

3.Make a HIVE Query:

```
SELECT word, count(*) from (SELECT explode(split(comments,'
')) as word from feedback)tmp GROUP BY word ORDER BY word;
```

HIVE Syntax: Create a Database

HIVE Syntax for creating databases:

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

A database in Hive is a **namespace** or a collection of tables.

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

```
hive> CREATE SCHEMA userdb;
```

HIVE Syntax: Create a Table

Syntax:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

Example:

```
hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,
salary String, destination String)
COMMENT ‘Employee details’
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE;
```

HIVE Syntax: Load Data into a Table

Syntax:

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)]
```

- *LOCAL* is identifier to specify the local path. It is optional.
- *OVERWRITE* is optional to overwrite the data in the table.
- *PARTITION* is optional.

Example:

```
hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt'
OVERWRITE INTO TABLE employee;
```

HIVE Syntax: Alter a Table

Syntax:

```
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
```

Example:

```
hive> ALTER TABLE employee RENAME TO emp;
```

```
hive> ALTER TABLE employee CHANGE name ename String;
hive> ALTER TABLE employee CHANGE salary salary Double;
```

```
hive> ALTER TABLE employee ADD COLUMNS (
dept STRING COMMENT 'Department name');
```

HIVE Syntax: Alter a Table

Example:

```
hive> ALTER TABLE employee CHANGE name ename String;
hive> ALTER TABLE employee CHANGE salary salary Double;
```

```
hive> ALTER TABLE employee ADD COLUMNS (
dept STRING COMMENT 'Department name');
```

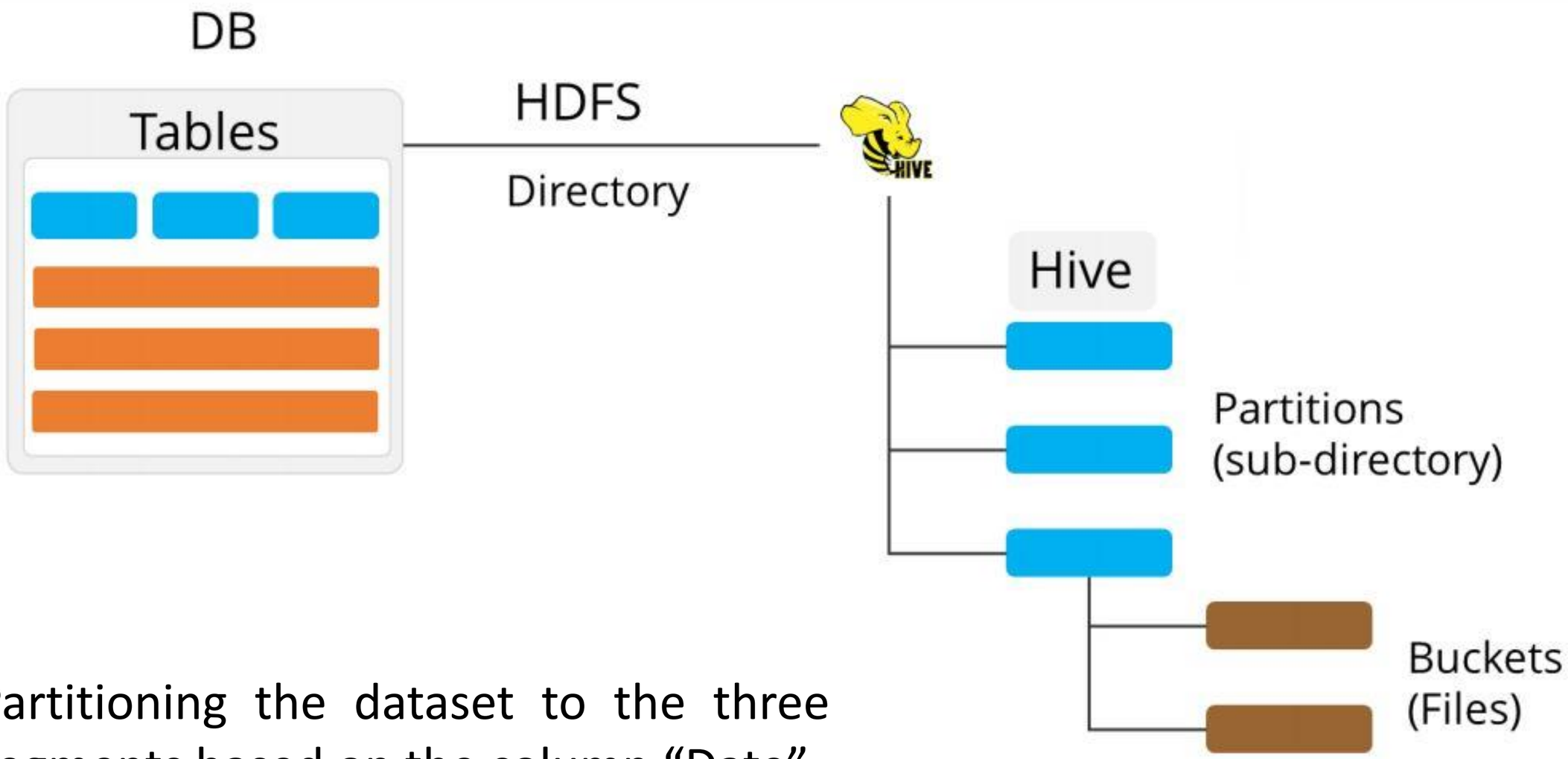
```
hive> ALTER TABLE employee REPLACE COLUMNS (
eid INT empid Int,
ename STRING name String);
```

Partitioning in HIVE

- Apache Hive converts the SQL queries into MapReduce jobs. When we submit a SQL query, Hive read the entire dataset. So, it becomes inefficient to run MapReduce jobs over a large table. Thus this is resolved by creating partitions in tables.
- Partitioning is a way of dividing a table into related parts based on the values of particular columns, such as date and city. Each table in the hive can have one or more partition keys to identify a particular partition.

```
CREATE TABLE table_name (column1 type, column2 type, …)
PARTITIONED BY (partition1 type, …);
```

Partitioning in HIVE



Partitioning the dataset to the three segments based on the column “Date”

Partitioning in HIVE: example

salesperson_id	product_id	date_of_sale
12	101	10-27-2017
10	10010	10-27-2017
111	2010	10-27-2017
13	222	10-28-2017
15	235	10-28-2017

```
CREATE TABLE Sale (personID int, ProductID int, DateofSale string)
PARTITIONED BY (DateofSale string);
```

Further studies on partitioning:

- ✓ Static/Dynamic Partitioning
- ✓ External Partitioned tables
- ✓ Alter Partitions

Partitioning in HIVE

Advantages:

Partitioning in Hive distributes execution load horizontally.

In partition faster execution of queries with the low volume of data takes place. For example, search population from Vatican City returns very fast instead of searching entire world population.

Disadvantages:

There is the possibility of too many small partition creations- too many directories.

But there some queries like group by on high volume of data take a long time to execute. For example, grouping population of China will take a long time as compared to a grouping of the population in Vatican City.

HIVE Syntax: SELECT … WHERE

Syntax:

```
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];
```

Example:

```
hive> SELECT * FROM employee WHERE salary>30000;
```

HIVE Syntax: SELECT … GROUP BY

Syntax:

```
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];
```

Example:

```
hive> SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

HIVE Syntax: Retrieving Information

Function	Hive
Retrieving Information (General)	`SELECT from_columns FROM table WHERE conditions;`
Retrieving All Values	`SELECT * FROM table;`
Retrieving Some Values	`SELECT * FROM table WHERE rec_name = "value";`
Retrieving With Multiple Criteria	`SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2";`
Retrieving Specific Columns	`SELECT column_name FROM table;`
Retrieving Unique Output	`SELECT DISTINCT column_name FROM table;`
Sorting	`SELECT col1, col2 FROM table ORDER BY col2;`
Sorting Reverse	`SELECT col1, col2 FROM table ORDER BY col2 DESC;`
Counting Rows	`SELECT COUNT(*) FROM table;`
Grouping With Counting	`SELECT owner, COUNT(*) FROM table GROUP BY owner;`
Maximum Value	`SELECT MAX(col_name) AS label FROM table;`

Built-in Functions with HIVE

Data type	Function	Description
BIGINT	round(double a)	Returns the rounded BIGINT value of the double.
BIGINT	floor(double a)	Returns the maximum BIGINT value that is equal or less than the double.
BIGINT	ceil(double a)	Returns the minimum BIGINT value that is equal or greater than the double.
double	rand(), rand(int seed)	Returns a random number (that changes from row to row). Specifying the seed will make sure the generated random number sequence is deterministic.
string	concat(string A, string B,...)	Returns the string resulting from concatenating B after A. For example, concat('foo', 'bar') results in 'foobar'. This function accepts an arbitrary number of arguments and returns the concatenation of all of them.
string	substr(string A, int start)	Returns the substring of A starting from start position till the end of string A. For example, substr('foobar', 4) results in 'bar'.
string	substr(string A, int start, int length)	Returns the substring of A starting from start position with the given length, for example, substr('foobar', 4, 2) results in 'ba'.
string	upper(string A)	Returns the string resulting from converting all characters of A to uppercase, for example, upper('fOoBaR') results in 'FOOBAR'.
string	ucase(string A)	Same as upper.
string	lower(string A)	Returns the string resulting from converting all characters of B to lowercase, for example, lower('fOoBaR') results in 'foobar'.
string	lcase(string A)	Same as lower.
string	trim(string A)	Returns the string resulting from trimming spaces from both ends of A, for example, trim('foobar ') results in 'foobar'.
string	ltrim(string A)	Returns the string resulting from trimming spaces from the beginning (left hand side) of A. For example, ltrim(' foobar ') results in 'foobar '.
string	rtrim(string A)	Returns the string resulting from trimming spaces from the end (right hand side) of A. For example, rtrim(' foobar') results in 'foobar'.
string	regexp_replace(string A, string B, string C)	Returns the string resulting from replacing all substrings in B that match the Java regular expression syntax (See Java regular expressions syntax) with C. For example, regexp_replace('foobar', 'oo|ar',) returns 'fb'.
int	size(Map<K.V>)	Returns the number of elements in the map type.
int	size(Array<T>)	Returns the number of elements in the array type.

HIVE Syntax: Create a VIEW

Views are in the virtual tables and display only selected data. It is good for security purposes.

Syntax:

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...) ]
[COMMENT table_comment]
AS SELECT ...
```

Example:

```
hive> CREATE VIEW emp_30000 AS
SELECT * FROM employee
WHERE salary>30000;
```

HIVE Syntax: DROP

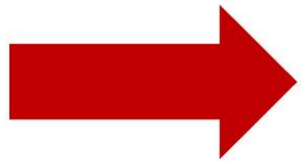
```
DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] database_name
```

```
hive> DROP DATABASE IF EXISTS userdb;
```

```
DROP TABLE [IF EXISTS] table_name;
```

```
hive> DROP TABLE IF EXISTS employee;
```

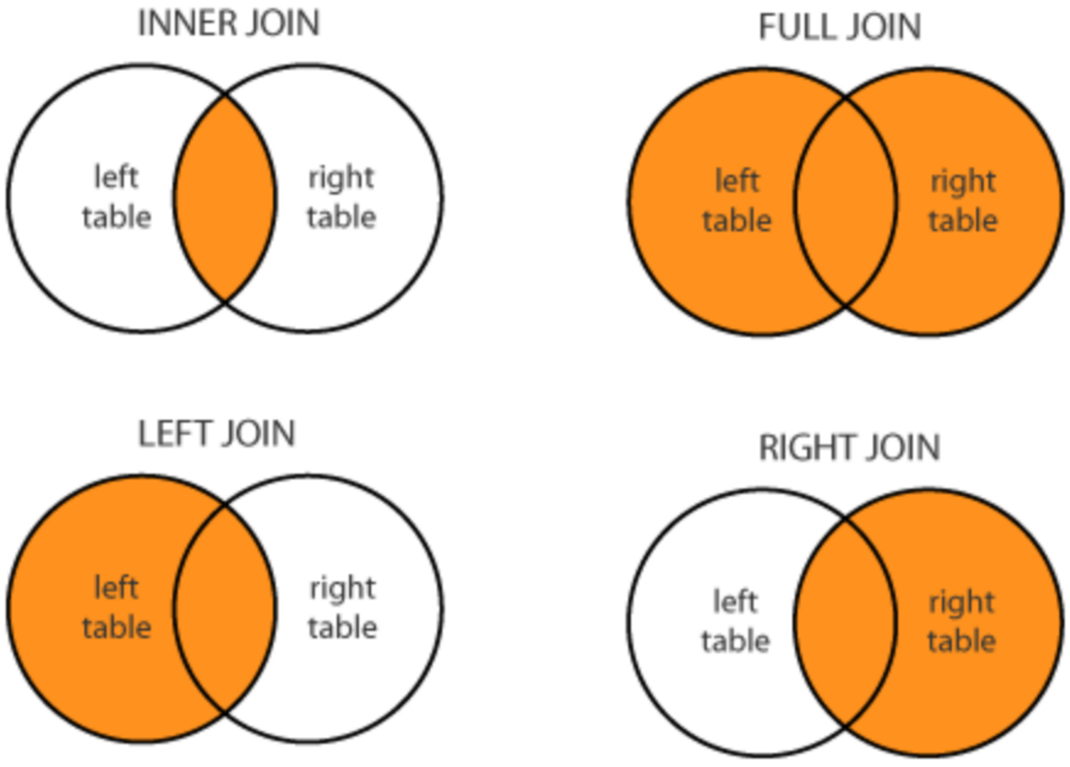
```
DROP VIEW view_name
```



```
hive> DROP VIEW emp_30000;
```

HIVE Syntax: JOIN

- (INNER) JOIN: Select records that have matching values in both tables.
- LEFT (OUTER) JOIN: Select records from the first (left-most) table with matching right table records.
- RIGHT (OUTER) JOIN: Select records from the second (right-most) table with matching left table records.
- FULL (OUTER) JOIN: Selects all records that match either left or right table records.



Source: https://www.dofactory.com/sql/join
Source: https://www.w3schools.com/sql/sql_join.asp

Future Topics for HIVE

- **HiveServer2:** It is a server interface that enables remote clients to submit queries to Hive and retrieve the results. HiveServer2 supports multi-client concurrency, capacity planning controls, Sentry authorization, Kerberos authentication, LDAP, SSL, and provides better support for JDBC and ODBC clients.
- **Hive on Spark:** Hive can use Spark as the underlying computation and parallelization engine.
- **Hive and HBase:** Apache HBase is a NoSQL database that supports real-time read/write access to large datasets in HDFS. You can configure Hive to use HBase.

Summary

- Discussed Hadoop Ecosystem
- Discussed Zookeeper and Oozie
- Introduced Apache HIVE
- Discussed HIVE Architecture and HIVEServer2
- Learned Partitioning in HIVE for Querying massive data
- Practiced HIVE Queries

University of East London