1 **MAPREDUCE –PARTITIONER**

- A partitioner works like a condition in processing an input dataset.
- The partition phase takes place after the Map phase and before the Reduce phase.
- The number of partitioners is equal to the number of reducers.
- That means a partitioner will divide the data according to the number of reducers.
- Therefore, the data passed from a single partitioner is processed by a single Reducer.

Partitioner

- A partitioner partitions the key-value pairs of intermediate Map-outputs.
- It partitions the data using a user-defined condition, which works like a hash function.
- The total number of partitions is same as the number of Reducer tasks for the job.
- Let us take an example to understand how the partitioner works.

MapReduce Partitioner Implementation

- For the sake of convenience, let us assume we have a small table called Employee with the following data.
- We will use this sample data as our input dataset to demonstrate how the partitioner works.

Table 3.1 Employee Dataset

Id	Name	Age	Gender	Salary

1201	gopal	45	Male	50,000
1202	manisha	40	Female	50,000
1203	khalil	34	Male	30,000
1204	prasanth	30	Male	30,000
1205	kiran	20	Male	40,000
1206	laxmi	25	Female	35,000
1207	bhavya	20	Female	15,000
1208	reshma	19	Female	15,000
1209	kranthi	22	Male	22,000
1210	Satish	24	Male	25,000
1211	Krishna	25	Male	25,000
1212	Arshad	28	Male	20,000
1213	lavanya	18	Female	8,000

- We have to write an application to process the input dataset to find the highest salaried employee by gender in different age groups (for example, below 20, between 21 to 30, above 30).
- The above data is saved as input.txt in the "/home/hadoop/hadoopPartitioner" directory and given as input.
- Based on the given input, following is the algorithmic explanation of the program.

- **Partitioner Task**
    - The partitioner task accepts the key-value pairs from the map task as its input. Partition implies dividing the data into segments. According to the given conditional criteria of partitions, the input key-value paired data can be divided into three parts based on the age criteria.
    - **Input** – The whole data in a collection of key-value pairs.
        - key = Gender field value in the record.
        - value = Whole record data value of that gender.
    - **Method** – The process of partition logic runs as follows.
        - Read the age field value from the input key-value pair
    - Check the age value with the following conditions.
        - Age less than or equal to 20
        - Age Greater than 20 and Less than or equal to 30.
        - Age Greater than 30.

```
if(age<=20)
{
   return 0;
}
else if(age>20 && age<=30)
{
```



```
   return 1 % numReduceTasks;
}
else
{
   return 2 % numReduceTasks;
}
```

- **Output** – The whole data of key-value pairs are segmented into three collections of key-value pairs. The Reducer works individually on each collection.

Apache Hadoop is a collection of open-source software that solves problems involving massive amounts of data computation. 2

If you want to learn more about **Hadoop**, you can refer to this link Hadoop.

Hadoop Yarn is an **Apache Hadoop** module which is responsible for managing computing resources in clusters and using them for scheduling users.
Today, In this blog, we will discuss **Managing Applications and Resources with Hadoop Yarn.**

So, without further ado, let's start our discussion.

Source

Hadoop Yarn

Hadoop is an open-source framework used to efficiently store and process large datasets ranging in size from gigabytes of data to petabytes of data.

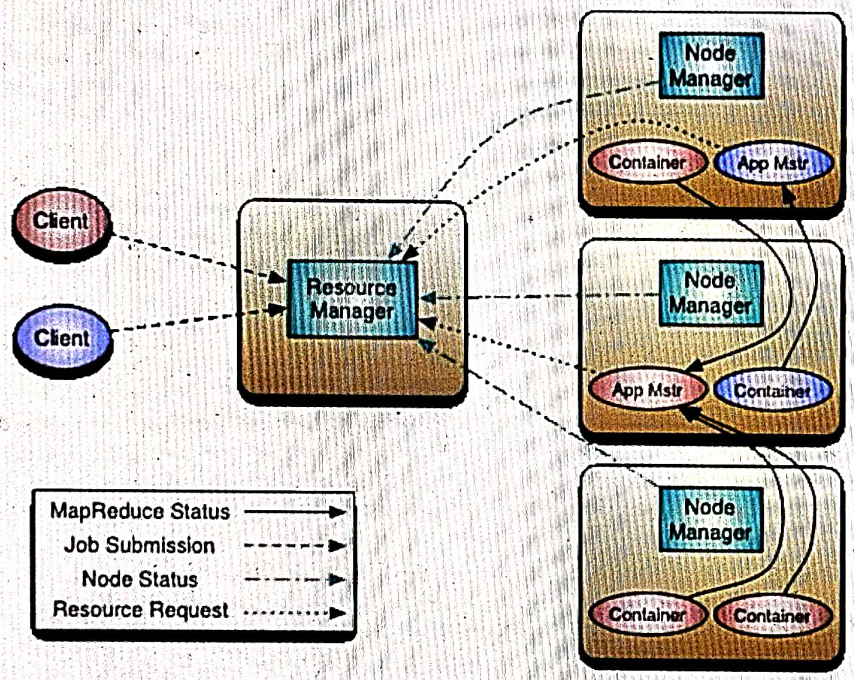
Yarn stands for **"Yet Another Resource Negotiator"**. It is the cluster resource management of Hadoop. It was introduced in **Hadoop 2.0**.

Yarn allows different data processing engines like interactive processing, graph processing, stream processing and batch processing to run and process data stored in **HDFS(Hadoop Distributed File System)**. Apart from this, it also allows job scheduling.

The fundamental idea of the YARN is to split up the functionalities of resource management and job scheduling. It provides two major services:

- **Global resource management (ResourceManager)**
- **Per-application management (ApplicationMaster)**

- Global resource management (ResourceManager)
- Per-application management (ApplicationMaster)



Resource Manager

The resource manager is the core component of **Hadoop Yarn**. It is the central controlling authority for resource management. The Resource Manager controls **NodeManager** in each of the nodes of a Hadoop cluster.

Resource manager has two main components **Scheduler** and **Application manager**.

- **Scheduler**: The Scheduler's task is to allocate system resources to specific running applications, but it does not track the status of the application.
- **Application Manager:** The task of the Application Manager is to accept the job submission and negotiate resources for executing the Application Master. If there is a failure, it restarts the Application Master.

Resource container stores all the required system information. It contains a detailed CPU, disk, network, and other important resource attributes necessary for running node and cluster applications.

Node Manager

Each node has a **Node Manager** slaved to the global Resource Manager in the cluster. It takes care of the individual compute nodes in a Hadoop cluster.

A few functionalities of the Node Manager are shown below:

- The Node Manager is responsible for launching the application containers for app execution.
- The Node Manager monitors the application's CPU, disk, network, and memory usage and reports back to the Resource Manager.
- All nodes of the cluster have a certain number of containers. Containers are computing units and wrappers for node resources to perform user application tasks.
- It is also responsible for tracking job status and progress within its node.

Application Master

For each application running on the node, there is a corresponding **Application Master**.

A few essential points regarding the Application Master are shown below:

- It is per application-specific entity.
- If more resources are necessary to support the running application, the Application Master notifies the Node Manager, and the Node Manager negotiates with the Resource Manager.
- It works with Node Manager for executing and monitoring component tasks.

3a Cassandra Architecture

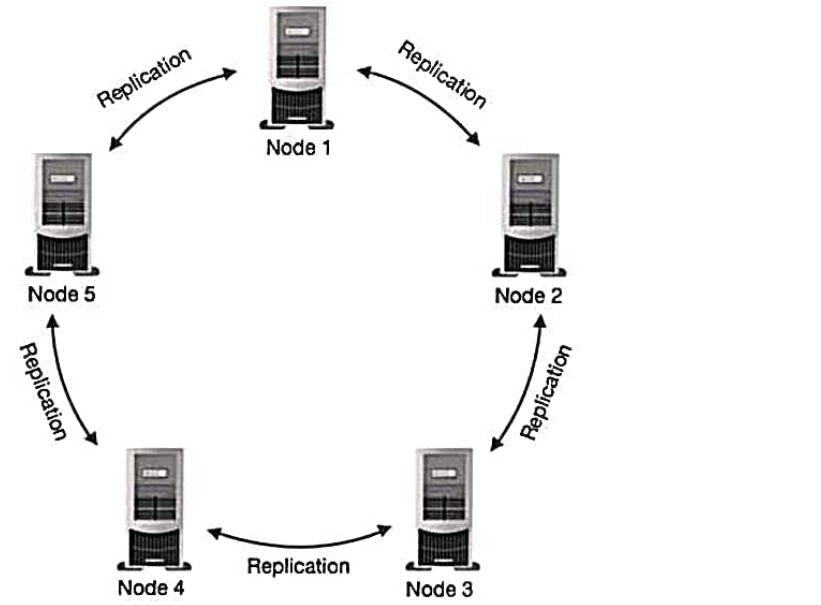
Cassandra was designed to handle big data workloads across multiple nodes without a single point of failure. It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- In Cassandra, each node is independent and at the same time interconnected to other nodes. All the nodes in a cluster play the same role.
- Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- In the case of failure of one node, Read/Write requests can be served from other nodes in the network.

Data Replication in Cassandra

In Cassandra, nodes in a cluster act as replicas for a given piece of data. If some of the nodes are responded with an out-of-date value, Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs a read repair in the background to update the stale values.

See the following image to understand the schematic view of how Cassandra uses data replication among the nodes in a cluster to ensure no single point of failure.



Components of Cassandra

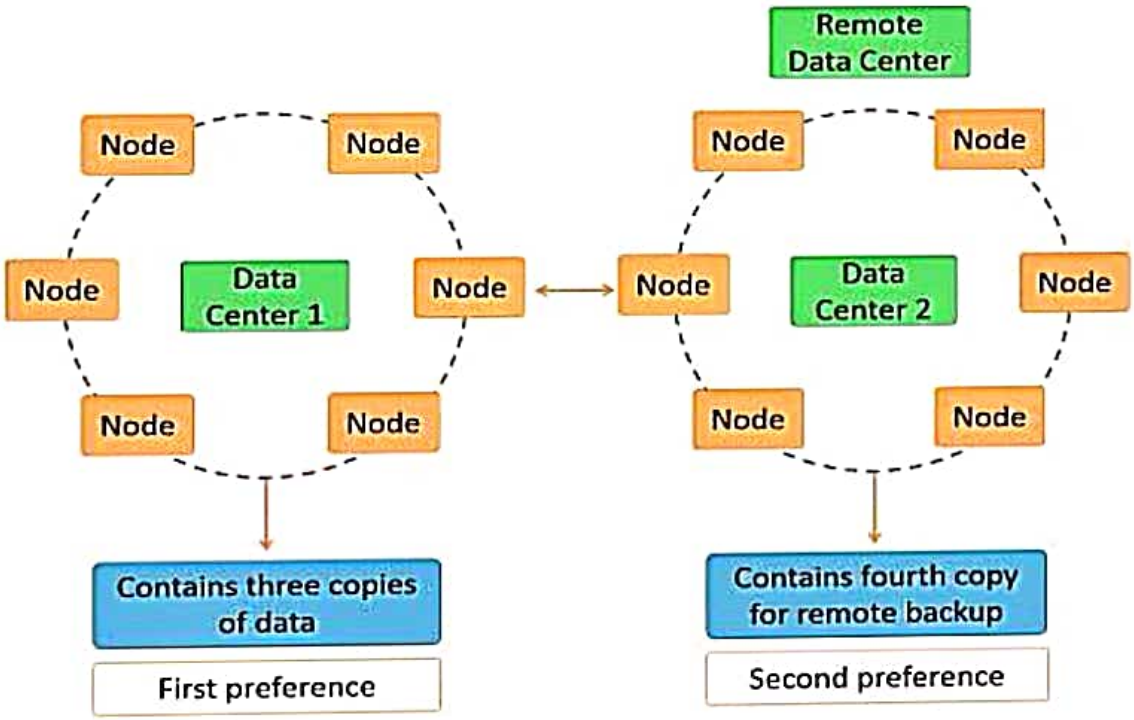
The main components of Cassandra are:

- **Node:** A Cassandra node is a place where data is stored.
- **Data center:** Data center is a collection of related nodes.
- **Cluster:** A cluster is a component which contains one or more data centers.
- **Commit log:** In Cassandra, the commit log is a crash-recovery mechanism. Every write operation is written to the commit log.
- **Mem-table:** A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable:** It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- **Bloom filter:** These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Additional features of Cassandra architecture are:

- Cassandra architecture supports multiple data centers.
- Data can be replicated across data centers.

You can keep three copies of data in one data center and the fourth copy in a remote data center for remote backup. Data reads prefer a local data center to a remote data center.



3b

❖ Cassandra CRUD operations on table(also called as column family)

- CRUD stands for create, read, update, delete
- Create
  - You can create a table using the command CREATE TABLE. Given below is the syntax for creating a table.

    Syntax
    CREATE (TABLE | COLUMNFAMILY) <tablename>
    ('<column-definition>' , '<column-definition>')
    (WITH <option> AND <option>)
  - Example

```
CREATE TABLE emp(
  emp_id int PRIMARY KEY,
  emp_name text,
  emp_city text,
  emp_sal varint,
  emp_phone varint
  );
```

- Read
  - SELECT clause is used to read data from a table in Cassandra. Using this clause, you can read a whole table, a single column, or a particular cell. Given below is the syntax of SELECT clause.

    SELECT FROM <tablename>
  - Example

```
Select * from emp;
Select * from emp where emp_id=123;
Select emp_sal from emp where emp_city='Hyd';
```

- Update
  - UPDATE is the command used to update data in a table. The following keywords are used while updating data in a table −

    Where − This clause is used to select the row to be updated.
    Set − Set the value using this keyword.
    Must − Includes all the columns composing the primary key.
  - While updating rows, if a given row is unavailable, then UPDATE creates a fresh row. Given below is the syntax of UPDATE command −

    UPDATE <tablename>
    SET <column name> = <new value>

    <column name> = <value>....
    WHERE <condition>
  - Example
    - UPDATE emp SET emp_city='Delhi',emp_sal=50000 WHERE emp_id=2;
- Delete
  - You can delete data from a table using the command DELETE. Its syntax is as follows −

    DELETE FROM <identifier> WHERE <condition>;
  - Example
    - DELETE emp_sal FROM emp WHERE emp_id=3;

❖ MongoDB CRUD Operations

4a

➢ CRUD operations create, read, update, and delete documents.

➢ Create Operations

- Create or insert operations and add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.
- MongoDB provides the following methods to insert documents into a collection:
  - db.collection.insertOne() New in version 3.2
  - db.collection.insertMany() New in version 3.2
- In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

```
db.users.insertOne(                    <-- collection
   {
      name: "sue",                     <-- field: value
      age: 26,                         <-- field: value      } document
      status: "pending"                <-- field: value
   }
)
```

➢ Read Operations

- Read operations retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection:
  - db.collection.find()
- You can specify query filters or criteria that identify the documents to return.

```
db.users.find(                         <-- collection
   { age: { $gt: 18 } },               <-- query criteria
   { name: 1, address: 1 }             <-- projection
).limit(5)                             <-- cursor modifier
```

➢ Update Operations

- Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:
  - db.collection.updateOne() New in version 3.2
  - db.collection.updateMany() New in version 3.2
  - db.collection.replaceOne() New in version 3.2
- In MongoDB, update operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.
- You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

```
db.users.updateMany(                   <-- collection
   { age: { $lt: 18 } },               <-- update filter
   { $set: { status: "reject" } }      <-- update action
)
```

➢ Delete Operations

- Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:
  - db.collection.deleteOne() New in version 3.2
  - db.collection.deleteMany() New in version 3.2
- In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.
- You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

```
db.users.deleteMany(                   <-- collection
   { status: "reject" }                <-- delete filter
)
```

4b In MongoDB, the `count()`, `limit()`, `sort()`, and `skip()` methods are commonly used to manipulate query results. Here's a brief explanation of each method along with examples:

1. **count()**: This method returns the count of documents that match the query criteria.

// Example: Count the number of documents in a collection

db.collection.count()

// Count documents that match certain criteria

db.collection.count({ field: value })

2. **limit()**: This method limits the number of documents returned in the result set.

```

// Example: Limit the number of documents returned to 5

db.collection.find().limit(5)

// Limit with query criteria

db.collection.find({ field: value }).limit(10)

```

3. **sort()**: This method sorts the documents in the result set based on the specified field(s) and sorting order.

```

// Example: Sort documents by a field in ascending order

db.collection.find().sort({ field: 1 })

// Sort in descending order

db.collection.find().sort({ field: -1 })

// Sort with multiple fields

db.collection.find().sort({ field1: 1, field2: -1 })

```

4. **skip()**: This method skips a specified number of documents and returns the rest.

```

// Example: Skip the first 10 documents and return the rest

db.collection.find().skip(10)

// Skip with query criteria

db.collection.find({ field: value }).skip(20)

```

Here's a combined example demonstrating the use of these methods together:

```

// Example: Find documents, apply sorting, limit, and skip

db.collection.find({ status: "active" }) // Filter documents with status "active"

  .sort({ createdAt: -1 }) // Sort by createdAt field in descending order

  .skip(10) // Skip the first 10 documents

  .limit(5); // Limit the result set to 5 documents

```

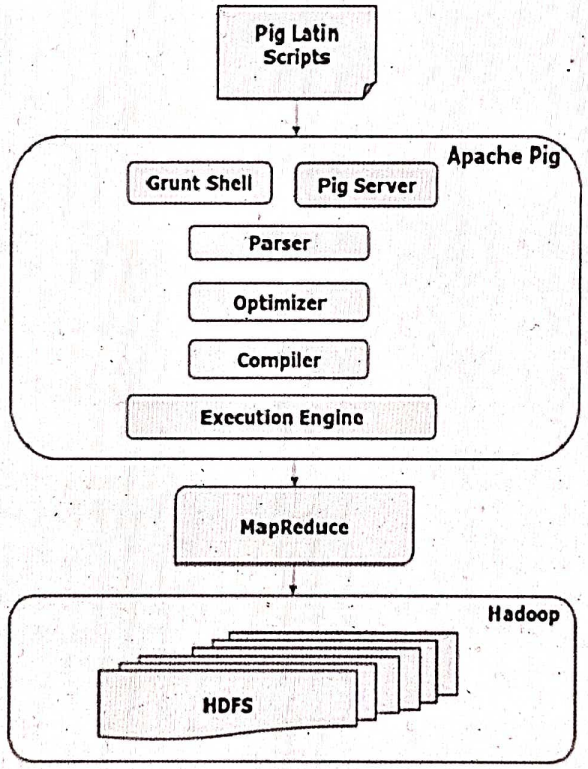
In this example, MongoDB will first filter documents with the `status` field equal to "active", then sort them based on the `createdAt` field in descending order, skip the first 10 documents, and finally return the next 5 documents.

The language used to analyze data in Hadoop using Pig is known as **Pig Latin**. It is a highlevel data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.

5a



Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

Compiler

The compiler compiles the optimized logical plan into a series of MapReduce jobs.

Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

5b Pig Latin – Relational Operations

The following table describes the relational operators of Pig Latin.

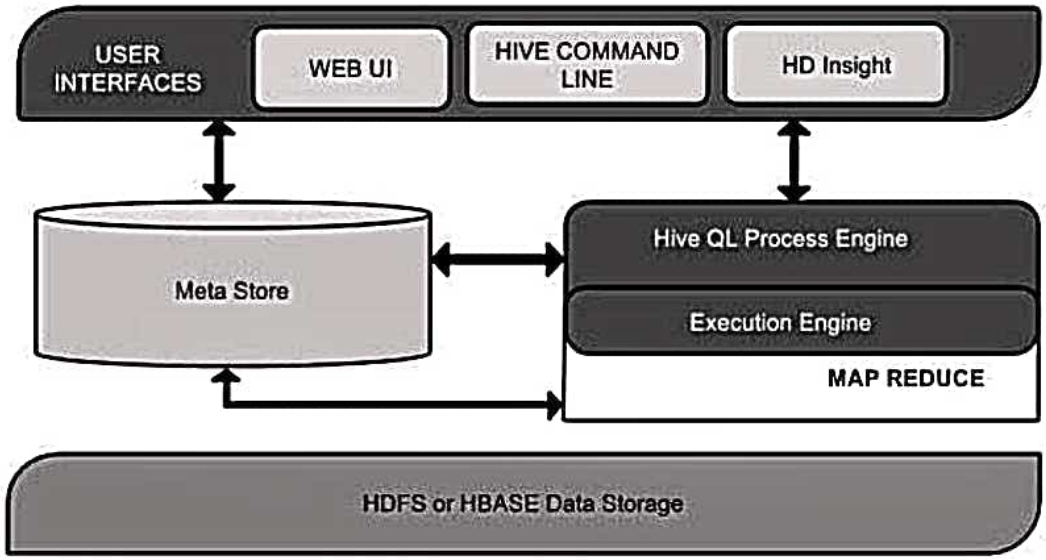
Operator	Description
Loading and Storing	
LOAD	To Load the data from the file system (local/HDFS) into a relation.
STORE	To save a relation to the file system (local/HDFS).
Filtering	

FILTER	To remove unwanted rows from a relation.
DISTINCT	To remove duplicate rows from a relation.
FOREACH, GENERATE	To generate data transformations based on columns of data.
STREAM	To transform a relation using an external program.
Grouping and Joining	
JOIN	To join two or more relations.
COGROUP	To group the data in two or more relations.
GROUP	To group the data in a single relation.
CROSS	To create the cross product of two or more relations.
Sorting	
ORDER	To arrange a relation in a sorted order based on one or more fields (ascending or descending).
LIMIT	To get a limited number of tuples from a relation.
Combining and Splitting	
UNION	To combine two or more relations into a single relation.
SPLIT	To split a single relation into two or more relations.

Diagnostic Operators	
DUMP	To print the contents of a relation on the console.
DESCRIBE	To describe the schema of a relation.
EXPLAIN	To view the logical, physical, or MapReduce execution plans to compute a relation.
ILLUSTRATE	To view the step-by-step execution of a series of statements.

6a **Architecture of Hive**

The following component diagram depicts the architecture of Hive:



This component diagram contains different units. The following table describes each unit:

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

What is Hive

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not

- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

6b

S.No.	Pig	Hive
1.	Pig operates on the client side of a cluster.	Hive operates on the server side of a cluster.
2.	Pig uses pig-latin language.	Hive uses HiveQL language.
3.	Pig is a Procedural Data Flow Language.	Hive is a Declarative SQLish Language.
4.	It was developed by Yahoo.	It was developed by Facebook.
5.	It is used by Researchers and Programmers.	It is mainly used by Data Analysts.
6.	It is used to handle structured and semi-structured data.	It is mainly used to handle structured data.
7.	It is used for programming.	It is used for creating reports.
8.	Pig scripts end with .pig extension.	In HIve, all extensions are supported.
9.	It does not support partitioning.	It supports partitioning.
10.	It loads data quickly.	It loads data slowly.
11.	It does not support JDBC.	It supports JDBC.
12.	It does not support ODBC.	It supports ODBC.
13.	Pig does not have a dedicated metadata database.	Hive makes use of the exact variation of dedicated SQL-DDL language by defining tables beforehand.
14.	It supports Avro file format.	It does not support Avro file format.
15.	Pig is suitable for complex and nested data structures.	Hive is suitable for batch-processing OLAP systems.
16.	Pig does not support schema to store data.	Hive supports schema for data insertion in tables.
17.	It is very easy to write UDFs to calculate matrices.	It does support UDFs but is much hard to debug.