**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# Big Data Analytics

## UNIT- 1 (Introduction to Big Data)

### What is Big Data:

**Definition:** Big data refers to the technologies and initiatives that involve data that is too diverse, fast changing or massive for conventional technologies, skills and infra structure to address efficiently. Big Data is "data of a very large size, typically to the extent that its manipulation and management presents significant logistical challenges for an enterprise.

The emerging technologies and practices that enable the collection, processing, discovery, analysis and storage of large volumes and disparate types of data quickly and cost effectively.

**Characteristics of Big Data:** Following are the big data core characteristics. Understanding the characteristics of big data is vital to know how it works and how you can use it. There are primarily seven characteristics of big data analytics:

1. Volume:

Volume refers to the amount of data that you have. We measure the volume of our data in Gigabytes, Zettabytes (ZB), and Yottabytes (YB). According to the industry trends, the volume of data will rise substantially in the coming years.

2. Velocity: Velocity refers to the speed of data processing. High velocity is crucial for the performance of any big data process. It consists of the rate of change, activity bursts, and the linking of incoming data sets.

3. Value:

Value refers to the benefits that your organization derives from the data. Does it match your organization's goals? Does it help your organization enhance itself? It's among the most important big data core characteristics.

4. Variety:

Variety refers to the different types of big data. It is among the biggest issues faced by the big data industry as it affects performance. It's vital to manage the variety of your data properly by organizing it. Variety is the various types of data that you gather from different kinds of sources.

5. Veracity:

Veracity refers to the accuracy of your data. It is among the most important Big Data characteristics as low veracity can greatly damage the accuracy of your results.

6. Validity:

How valid and relevant is the data to be used for the intended purpose.

7. Volatility: Big data is constantly changing. The data you gathered from a source a day ago might be different from what you found today. This is called variability of data, and it affects your data homogenization.

8. Visualization

Visualization refers to showing your big data-generated insights through visual representations such as charts and graphs. It has become prevalent recently as big data professionals regularly share their insights with non-technical audiences.

Sources of Big Data: A significant part of big data is generated from three primary resources:

Machine data

Social data, and

Transactional data.


1. Machine Data

Machine data is automatically generated, either as a response to a specific event or a fixed schedule. It means all the information is developed from multiple sources such as smart sensors, SIEM logs, medical devices and wearables, road cameras, IoT devices, satellites, desktops, mobile phones, industrial machinery, etc. These sources enable companies to track consumer behaviour. Data extracted from machine sources grow exponentially along with the changing external environment of the market.

In a broader context, machine data also encompasses information churned by servers, user applications, websites, cloud programs, and so on.

## 2. Social Data

It is derived from social media platforms through tweets, retweets, likes, video uploads, and comments shared on Facebook, Instagram, Twitter, YouTube, Linked In etc. The extensive data generated through social media platforms and online channels offer qualitative and quantitative insights on each crucial facet of brand-customer interaction.

## 3. Transactional Data

As the name suggests, transactional data is information gathered via online and offline transactions during different points of sale. The data includes vital details like transaction time, location, products purchased, product prices, payment methods, discounts/coupons used, and other relevant quantifiable information related to transactions.

The sources of transactional data include:

Payment orders

Invoices

Storage records and

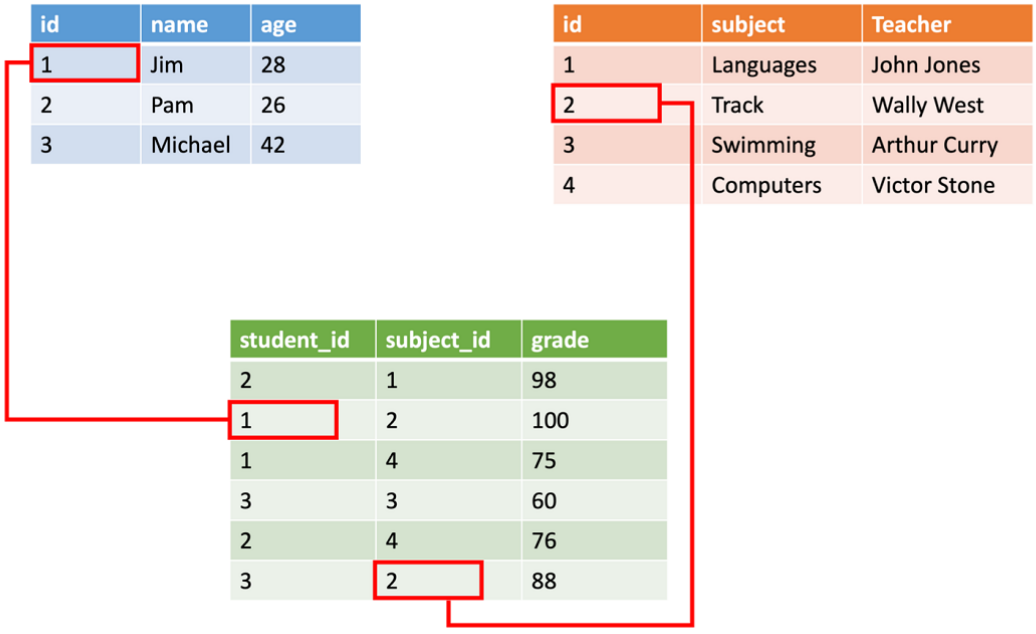
E-receipts

# Types of Big Data

There are primarily three types of data in big data:

## 1. Structured

Structured data refers to the data that you can process, store, and retrieve in a fixed format. It is highly organized information that you can readily and seamlessly store and access from a database by using simple algorithms. This is the easiest type of data to manage as you know what data format you are working with in advance. For example, the data that a company stores in its databases in the form of tables and spreadsheets is structured data.



| id | name | age |
|---|---|---|
| 1 | Jim | 28 |
| 2 | Pam | 26 |
| 3 | Michael | 42 |

| id | subject | Teacher |
|---|---|---|
| 1 | Languages | John Jones |
| 2 | Track | Wally West |
| 3 | Swimming | Arthur Curry |
| 4 | Computers | Victor Stone |

| student_id | subject_id | grade |
|---|---|---|
| 2 | 1 | 98 |
| 1 | 2 | 100 |
| 1 | 4 | 75 |
| 3 | 3 | 60 |
| 2 | 4 | 76 |
| 3 | 2 | 88 |

2. Unstructured

Data with an unknown structure is termed unstructured data. Its size is substantially bigger than structured data and is heterogeneous in nature. A great example of unstructured data includes the results you get when you perform a Google search. You get webpages, videos, images, text, and other data formats of varying sizes.

Examples of unstructured data include:

Media files, like photos, videos, and audio files

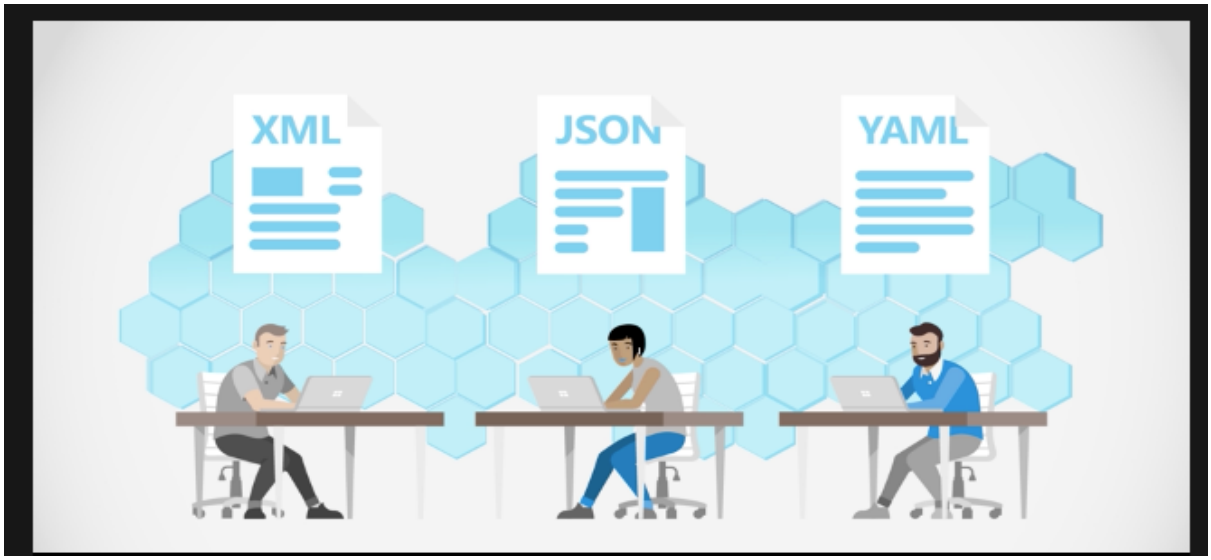
Microsoft 365 files, like Word documents

Text files

Log files

3. Semi-structured

As the name suggests, semi-structured data contains a combination of structured and unstructured data. Semi-structured data is less organized than structured data. Semi-structured data isn't stored in a relational format because the fields don't fit neatly into tables, rows, and columns. Semi-structured data contains tags that make the organization and hierarchy of the data apparent. One example is key/value pairs. Semi-structured data is also referred to as non-relational or not only SQL (NoSQL) data.




XML

```xml
<Person Age="23">
    <FirstName>Quinn</FirstName>
    <LastName>Anderson</LastName>
    <Hobbies>
        <Hobby Type="Sports">Golf</Hobby>
        <Hobby Type="Leisure">Reading</Hobby>
        <Hobby Type="Leisure">Guitar</Hobby>
   </Hobbies>
</Person>
```

JSON

```
{
    "firstName": "Quinn",
    "lastName": "Anderson",
    "age": "23",
    "hobbies": [
        { "type": "Sports", "value": "Golf" },
        { "type": "Leisure", "value": "Reading" },
        { "type": "Leisure", "value": "Guitar" }
    ]
}
```

**Challenges in Big Data:** These are the following challenges that come into the way while dealing with big data.

1. Lack of Knowledge professional: To run the modern technologies and large data tools, companies need skill data professionals. (data scientists, data analysts, data engineers).
2. Lack of proper understanding of massive data: Employees might not know what data is , its storage, processing, importance and sources.
3. Data Growth Issues: In RDBMS the data is kept in different-different tables to ensure normalization and eliminate redundancy. For Selection of information, the join queries will be fired on the respective tables in the dataset. So these join queries will take more time if the data is massive or keep

on increasing everyday. As these data grow exponentially with time it get more challenging to handle the data.

4. Confusion while Big data tool Selection: Companies often get confused while selecting the simplest tool for giant data analysis and storage.

   - Is HBase or Cassandra for data Storage.
   - Hadoop Map Reduce or Spark for Data Analysis.

5. Integrating data from a variety of Sources: Data comes from variety of resources such as social media pages, ERP applications, Customer logs, financial reports, email, presentations and various reports created by employees.

   So combining all this data into an organize reports might be challenging tasks for clients.

6. Securing Data: Securing these huge set of knowledge is one of the daunting challenges of massive data. Often companies are busy in understanding , storing and analyzing their datasets, that they push data security for later stages.

7. Storage and Transport Issues: Means Data transfer takes larger time then data processing.

8. Data Management Issues:

- Sources of data is varied by size, format, methods of collections
- What, Where, why, Who, When and How the data is collected and its provenance.

- Rigorous protocols are followed to ensure its accuracy and validity.

9. Processing Issues:

-Extensive processing and algorithms are required to provide timely and actionable information.

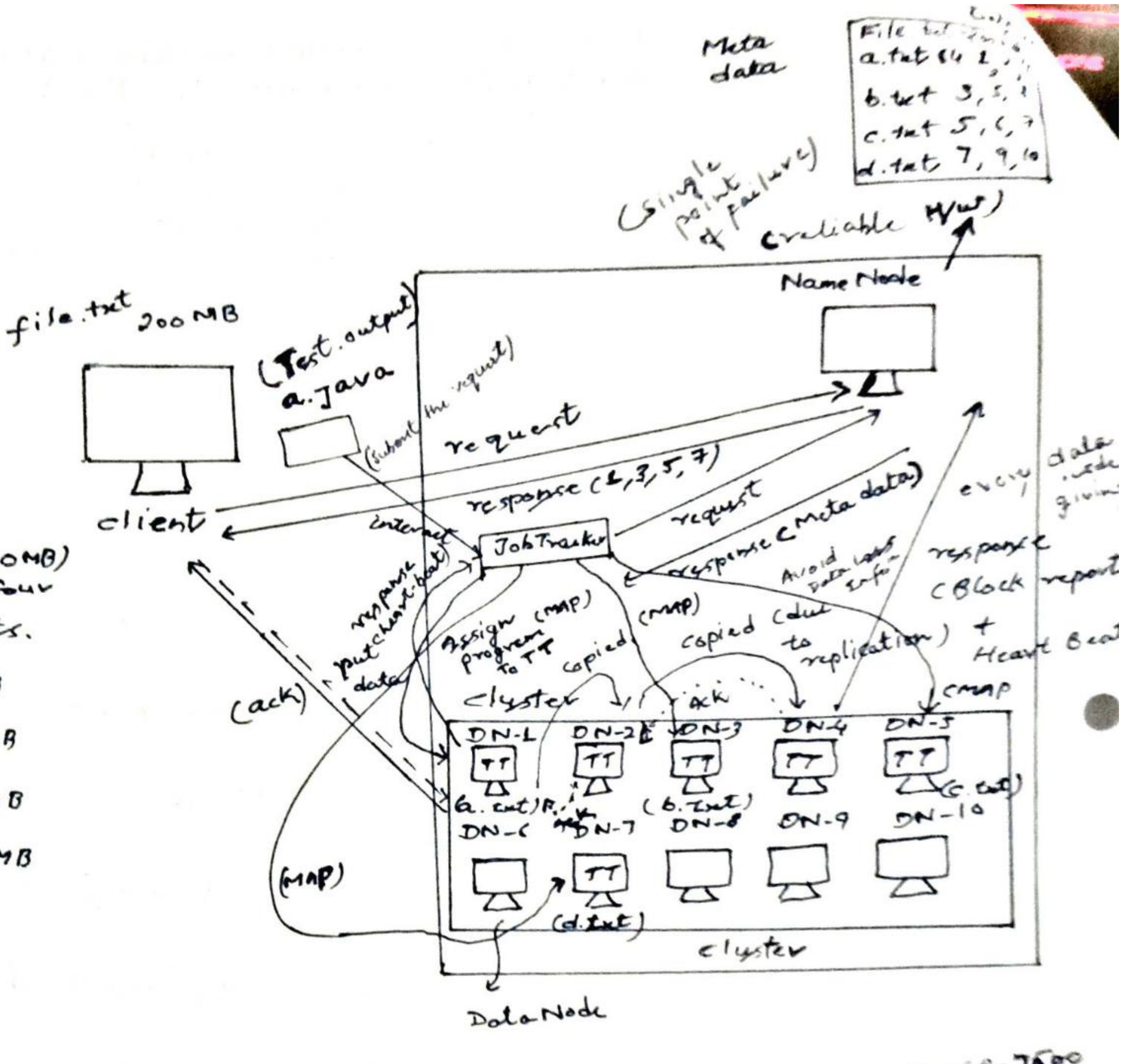
10. Characteristics Issues:

-Data Volume- Terabyte of data being produced everyday which is might be unmanageable using existing traditional systems.

- Data Velocity: Traditional System are not capable of performing analytics of data which is in motion. Ecommerce has increased the speed and richness of data used for business transactions.

- Data Variety-Since the data comes in various forms from various sources.

-Incompatible data formats, non-aligned data structure represents significant challenges that can lead to analytic spread.




file.txt - (200MB) spilts into four input spilts.

a.txt - 64MB

b.txt - 64MB

c.txt - 64MB

d.txt - 64MB

(HDFS Architecture)

# HDFS Architecture:

-- HDFS stands for Hadoop Distributed File Systems.

-- HDFS is used to store (HDFS) and processing (Map-Reduce) a large data sets.

-- HDFS is a specially designed file systems for storing a huge datasets with a cluster of commodity hardware with streaming access patterns. (Means write once and read any number of times but don't try to change the contents of a file on HDFS.)

--Block size of HDFS is 64 MB in Hadoop 1.0 . (By Default) You can also set the block size up to 128 MB. (Hadoop 2.0 ) (Job of Hadoop Administrator Profile).

--Hadoop Distributed File System is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a Master/Slave Architecture, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes).

-- Some of the services that are running on the Hadoop Distributed File Systems are on the top of current operating files systems are as-

-- NameNode
-- Secondary NameNode
-- Job Tracker

} These are called as Master Services

-- DataNode
-- Task Tracker

} These services are called as slave Services or Daemons.

-- Master Services and Slave Services can talk to each other to coordinate their works.( Means **NameNode** can interact to **DataNode** and **JobTracker** can interact to **Tasktracker** to coordinate their work on HDFS).

1. NameNode : NameNode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes).

--NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients. The HDFS architecture is built in such a way that the user data never resides on the NameNode. The data resides on DataNodes only.

--NameNode contains metadata about blocks allocated to data, block replica information of the data over to the clusters of data node, Number of splits of datasets. For example if a dataset file.txt is of 200 MB. If it is deployed over to HDFS then that dataset is divided in four data block of size 64MB, 64 MB, 64 MB, 8 MB block respectively over to HDFS.

| | SIZE. | Data Node and Replica information about block |
|---|---|---|
| a.txt. | 64 MB | 1, 2, 4 |
| b.txt. | 64 MB. | 3, 5, 8 |
| c.txt. | 64 MB. | 5, 6, 7 |
| d.txt. | 64 MB. | 7, 9, 10 |

- HDFS maintains the 3 replica of the data on the Data Node of the cluster.

## Functions of NameNode:

--It is the master daemon that maintains and manages the DataNodes (slave nodes)

--It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata:

-- **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode.

-- **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage.

- It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- It regularly receives a **Heartbeat** and **a block report** from all the DataNodes in the cluster to ensure that the DataNodes are live.
- It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
- The NameNode is also responsible to take care of the **replication factor** of all the blocks.

- In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

Important Point: If a NameNode is fails then entire concept of HDFS is fail so NameNode is a single point of failure since it contains all the information about the metadata of the blocks and its replicas.

2. Data Node: DataNodes are the slave nodes in HDFS. Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability. The DataNode is a block server that stores the data in the local file ext3 or ext4.

## Functions of DataNode:

--These are slave daemons or process which runs on each slave machine.

--The actual data is stored on DataNodes.

--The DataNodes perform the low-level read and write requests from the file system's clients.

--They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# Job Tracker and Task Tracker in Hadoop:

- JobTracker and TaskTracker are 2 essential process involved in MapReduce execution in MRv1 (or Hadoop version 1). Both processes are now deprecated in MRv2 (or Hadoop version 2) and replaced by Resource Manager, Application Master and Node Manager Daemons.

## Job Tracker –

- JobTracker process runs on a separate node and not usually on a DataNode.
- JobTracker is an essential Daemon for MapReduce execution in MRv1. It is replaced by ResourceManager/ApplicationMaster in MRv2.

- JobTracker receives the requests for MapReduce execution from the client.
- JobTracker talks to the NameNode to determine the location of the data.
- JobTracker finds the best TaskTracker nodes to execute tasks based on the data locality (proximity of the data) and the available slots to execute a task on a given node.
- JobTracker monitors the individual TaskTrackers and the submits back the overall status of the job back to the client.
- JobTracker process is critical to the Hadoop cluster in terms of MapReduce execution.

When the JobTracker is down, HDFS will still be functional but the MapReduce execution can not be started and the existing MapReduce jobs will be halted.

**TaskTracker –**

- TaskTracker runs on DataNode. Mostly on all DataNodes.

TaskTracker is replaced by Node Manager in MRv2.

- Mapper and Reducer tasks are executed on DataNodes administered by TaskTrackers.
- TaskTrackers will be assigned Mapper and Reducer tasks to execute by JobTracker.

- TaskTracker will be in constant communication with the JobTracker signalling the progress of the task in execution.
- TaskTracker failure is not considered fatal. When a TaskTracker becomes unresponsive, JobTracker will assign the task executed by the TaskTracker to another node.

Secondary NameNode: The **Secondary namenode** is a helper node in hadoop, To understand the functionality of the secondary namenodelet's understand how the namenode works.

Name node stores metadata like file system namespace information, blockinformation etc in the memory. It also stores the persistent copy of the same on the disk. Name node stores information in two files.

**fsimage:** It's a snapshot of the file system, stores information like modification time access time, permission, replication.

**Edit logs:** It stores details of all the activities/transactions being performed on the HDFS..

When the namenode is in the active state the edit logs size grows continuously as the edit logs can only be applied to the fsimage at the time of namenode restart, to get the latest state of the HDFS. If edit logs grows significantly and namenode tries to apply it on fsimage at the time of namenode restart, the process can take very long, here secondary node come into the play.

Secondary namenode keeps the checkpoint on the namenode, It reads the edit logs from the namenode continuously after a specific interval and applies it to the fsimage copy of secondary namenode. In this way the fsimage file will have the most recent state of HDFS.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# Introducing and Configuring Hadoop cluster (Local, Pseudo distributed mode, Fully Distributed mode):

## Local Mode:

Standalone mode is the default mode in which Hadoop run. Standalone mode is mainly used for debugging where you don't really use HDFS.

You can use input and output both as a local file system in standalone mode.

You also don't need to do any custom configuration in the files- mapred-site.xml, core-site.xml, hdfs-site.xml.

Standalone mode is usually the fastest Hadoop modes as it uses the local file system for all the input and output. Here is the summarized view of the standalone mode-

- Used for debugging purpose
- HDFS is not being used
- Uses local file system for input and output

- No need to change any configuration files
- Default Hadoop Modes

## 2) Pseudo-distributed Mode

--The pseudo-distribute mode is also known as a single-node cluster where both NameNode and DataNode will reside on the same machine.

--In pseudo-distributed mode, all the Hadoop daemons will be running on a single node. Such configuration is mainly used while testing when we don't need to think about the resources and other users sharing the resource.

--In this architecture, a separate JVM is spawned for every Hadoop component as they could communicate across network sockets, effectively producing a fully functioning and optimized mini-cluster on a single host.

Here is the summarized view of pseudo distributed Mode-

- Single Node Hadoop deployment running on Hadoop is considered as pseudo distributed mode

- All the master & slave daemons will be running on the same node
- Mainly used for testing purpose
- Replication Factor will be ONE for Block
- Changes in configuration files will be required for all the three files- mapred-site.xml, core-site.xml, hdfs-site.xml.

### 3) Fully-Distributed Mode (Multi-Node Cluster)

This is the production mode of Hadoop where multiple nodes will be running. Here data will be distributed across several nodes and processing will be done on each node.

Master and Slave services will be running on the separate nodes in fully-distributed Hadoop Mode.

- Production phase of Hadoop
- Separate nodes for master and slave daemons
- Data are used and distributed across multiple nodes

In the Hadoop development, each Hadoop Modes have its own benefits and drawbacks. Definitely fully distributed mode is the one for which Hadoop is mainly known for but again there is no point in engaging the resource while in testing or debugging phase.

So standalone and pseudo-distributed Hadoop modes are also having their own significance.

Follow these steps for installing and configuring Hadoop on a single node:

Step-1. Install Java: we will use Java 1.6 therefore describing the installation of Java 1.6 in detail.

Use the below command to begin the installation of Java

$ sudo apt-get install openjdk-6-jdk

Step-2. Verify Java installation

You can verify java installation using the following command

$ java -version

On executing this command, you should see output similar to the following:

java version "1.6.0_27"

Step-3. SSH configuration

Install SSH using the command.

sudo apt-get install ssh

Generate ssh key

ssh -keygen -t rsa -P "" (press enter when asked for a file name; this will generate a passwordless ssh file)

Now copy the public key (id_rsa.pub) of current machine to authorized_keys. Below command copies the generated public key in the .ssh/authorized_keys file:

cat $HOME/.ssh/id_rsa.pub >>

$HOME/.ssh/authorized_keys

Verify ssh configuration using the command

ssh localhost

Pressing yes will add localhost to known hosts

Step-4. Download Hadoop

Download the latest stable release of Apache Hadoop from

http://hadoop.apache.org/ releases.html

Unpack the release tar – zxvf hadoop-1.0.3.tar.gz

Save the extracted folder to an appropriate location,

HADOOP_HOME will be pointing to this directory.

Step-5. Verify Hadoop

Check if the following directories exist under

HADOOP_HOME: bin, conf, lib, bin

Use the following command to create an environment variable that

points to the Hadoop installation directory

(HADOOP_HOME)

export HADOOP_HOME=/home/user/hadoop

Now place the Hadoop binary directory on your command-line path by executing the command

export PATH=$PATH:$HADOOP_HOME/bin

Use this command to verify your Hadoop installation:

hadoop version

The o/p should be similar to below one

Hadoop 1.1.2

Step-6. Configure JAVA_HOME:

Use the below command to set JAVA_HOME on Ubuntu

export JAVA_HOME=/usr/lib/jvm/java-6-sun

JAVA_HOME can be verified by command

echo $JAVA_HOME

Step-7. Create Data Directory for Hadoop

An advantage of using Hadoop is that with just a limited number of directories you can set it up to work correctly. Let us create a directory with the name hdfs and three sub-directories name, data and tmp.

Since a Hadoop user would require to read-write to these directories you would need to change the permissions of above directories to 755 or 777 for Hadoop user.

Step-8. Configure Hadoop XML files

Next, we will configure Hadoop XML file. Hadoop configuration files are in the HADOOP_HOME/conf dir.

conf/core-site.xml

<! -- Putting site-specific property overrides the file. -->

fs.default.name

hdfs://localhost:9000

hadoop.temp.dir

/home/girish/hdfs/temp

conf/hdfs-site.xml

<! -- Putting site specific property overrides in the file. -->

dfs.name.dir

/home/girish/hdfs/name

dfs.data.dir

/home/girish/hdfs/data

dfs.replication

1

conf/mapred-site.xml

<! -- Putting site-specific property overrides this file. -->

mapred.job.tracker

localhost:9001

conf/masters

Not required in single node cluster.

conf/slaves

Not required in single node cluster.

Step-9. Format Hadoop Name Node-

Execute the below command from hadoop home directory

$ ~/hadoop/bin/hadoop namenode -format

Step-10. Start Hadoop daemons

$ ~/hadoop/bin/start-all.sh

Step-11. Verify the daemons are running

$ jps (if jps is not in path, try /usr/java/latest/bin/jps)

output will look similar to this

9316 SecondaryNameNode

9203 DataNode

9521 TaskTracker

9403 JobTracker

9089 NameNode

Now we have all the daemons running

Step-12. Verify UIs by namenode & job tracker

Open a browser window and type the following URLs:

namenode UI: http://machine_host_name:50070

job tracker UI: http://machine_host_name:50030

substitute 'machine host name' with the public IP of your node

e.g: http://localhost:50070

Now you have successfully installed and configured Hadoop on a single node

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

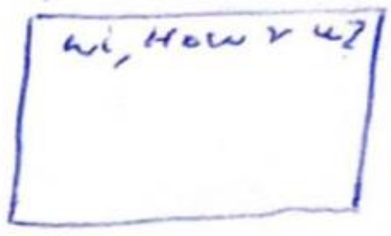
# UNIT- II (Writing MAP-REDUCE Programs)

## Helloworld Job:-

**step 1:** creating a file

```
$ cat > file.txt
```

file.txt




**step-2:** Loading file.txt from local file system to HDFS

```
$ hadoop fs -put file.txt file
```

**step-3:** writing programs

DriverCode.java → MapperCode.java → ReducerCode.java.

**step 4:** Compiling all above .java files.

```
$ javac -classpath $HADOOP-HOME/hadoop-core.jar
        *.java
```

**step 5:** creating jar file

```
$ jar cvf test.jar *.class
```

**step 6:** Running test.JAR on file (which is there in your HDFS).

```
$ hadoop JAR test.JAR DriverCode file TestOutput
```

↑ Same file or folder

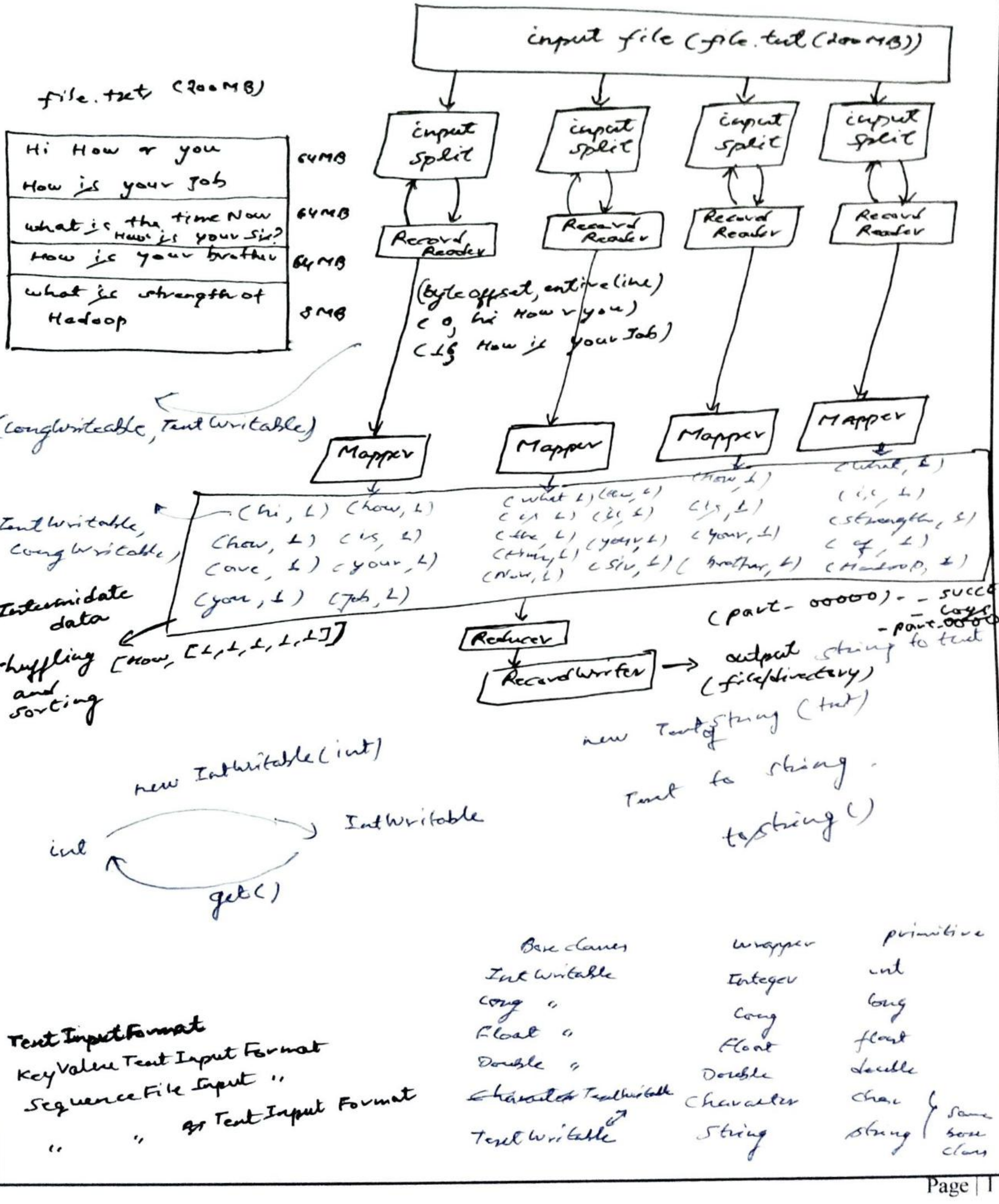
↓ (containing Main Method)

↓ only dir

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

MAPReduce Flow chart:

hadoop jar test.jar DriverCode file.txt
Testoutputfile




TextInputFormat
KeyValue Text Input Format
SequenceFile Input "
" " or Text Input Format

| Base classes | wrapper | primitive | |
|---|---|---|---|
| IntWritable | Integer | int | |
| Long " | Long | long | |
| Float " | Float | float | |
| Double " | Double | double | |
| ~~Character~~ TextWritable | Character | char | Same base class |
| TextWritable | String | String | |

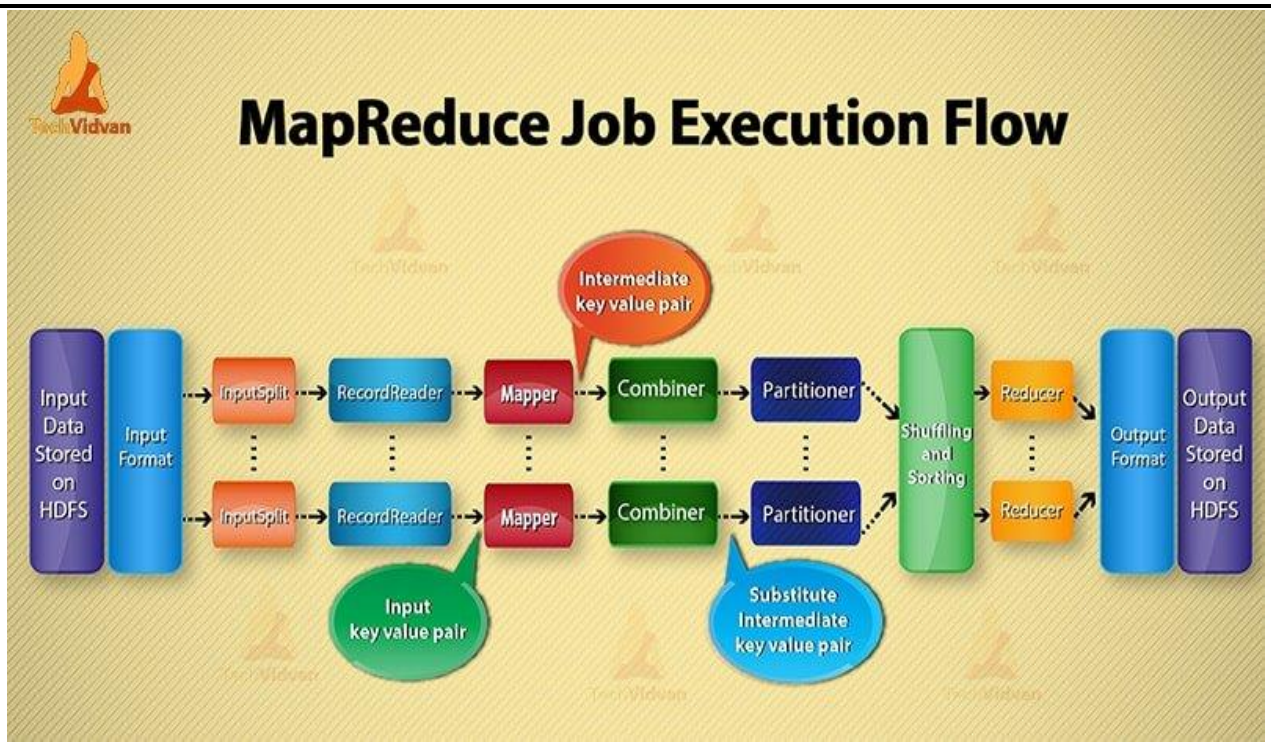
**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# Hadoop Map Reduce Job Execution Flow Chart:

Hadoop Map Reduce is the data processing layer. It processes the huge amount of structured and unstructured data stored in HDFS. Map Reduce processes data in parallel by dividing the job into the set of independent tasks. So, parallel processing improves speed and reliability.

Hadoop Map Reduce data processing takes place in 2 phases- Map and Reduce phase.

- Map phase- It is the first phase of data processing. In this phase, we specify all the complex logic/business rules/costly code.
- Reduce phase- It is the second phase of processing. In this phase, we specify light-weight processing like aggregation/summation.




**Steps of MAP Reduce Flow Chart:** These are the following steps that are performed in the MAP Reduce.

1. Input Files

In input files data for Map Reduce job is stored. In HDFS, input files reside. Input files format is arbitrary. Line-based log files and binary format can also be used.

2. InputFormat:

After that InputFormat defines how to split and read these input files. It selects the files or other objects for input. InputFormat creates InputSplit.

3. InputSplits

It represents the data which will be processed by an individual Mapper. For each split, one map task is created. Thus the number of map tasks is equal to the number of InputSplits. Framework divide split into records, which mapper process.

4. RecordReader

--It communicates with the inputSplit. And then converts the data into key-value pairs suitable for reading by the Mapper.

--RecordReader by default uses TextInputFormat to convert data into a key-value pair.

-- It assigns byte offset to each line present in the file. Then, these key-value pairs are further sent to the mapper for further processing.

5. Mapper

--It processes input record produced by the RecordReader and generates intermediate key-value pairs.

-- The output of the mapper is the full collection of key-value pairs.

6. Combiner

Combiner is Mini-reducer which performs local aggregation on the mapper's output. It minimizes the data transfer between mapper and reducer.

7. Partitioner:

Partitioner comes into the existence if we are working with more than one reducer. It takes the output of the combiner and performs partitioning.

8. Shuffling and Sorting: In this phase all common key-value pair are shuffled on the basis of key and their common key and values are merged.

Then all the mappers finish and shuffle the output on the reducer nodes. Then framework merges this intermediate output and sort.

9. Reducer: Reducer then takes set of intermediate key-value pairs produced by the mappers as the input. After that runs a reducer function on each of them to generate the output.

The output of the reducer is the final output. Then framework stores the output on Output Directory over to HDFS.

10. Record-Writer: It writes these output key-value pair from the Reducer phase to the output files.

11. OutputFormat:

--OutputFormat defines the way how RecordReader writes these output key-value pairs in output files.

--Thus OutputFormat instances write the final output of reducer on HDFS.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# Map Reduce (Word Count Problem code in Java):

Here, the role of Mapper is to map the keys to the existing values and the role of Reducer is to aggregate the keys of common values. So, everything is represented in the form of Key-value pair.

MapReduce consists of two steps:

- **Map Function** – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

## Example – (Map function in Word Count)

| | | |
|---|---|---|
| **Input** | Set of data | Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN |
| **Output** | Convert into another set of data<br>(Key,Value) | (Bus,1), (Car,1), (bus,1), (car,1), (train,1),<br>(car,1), (bus,1), (car,1), (train,1), (bus,1),<br>(TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1),<br>(car,1), (BUS,1), (TRAIN,1) |

## WordMapper.java:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordMapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{
```

```
    public void map(LongWritable key,Text value,
OutputCollector<Text,IntWritable> output, Reporter r)
    throws IOException{
        String s=value.toString();
        for (String word:s.split(" ")){
            if(word.length()>0){
                output.collect(new Text(word),new IntWritable(1));
            }
        }
    }
}
```

WordReducer.java:

- Reduce Function – Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

Example – (Reduce function in Word Count)

| | | |
|---|---|---|
| **Input (output of Map function)** | Set of Tuples | (Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1) |
| **Output** | Converts into smaller set of tuples | (BUS,7), (CAR,7), (TRAIN,4) |

```
public class WordReducer extends MapReduceBase implements
Reducer<Text,IntWritable,Text,IntWritable>{
    public void reduce(Text key,Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output,
    Reporter r) throws IOException{
        int count=0;
        while(values.hasNext()){
            IntWritable i=values.next();
            count+=i.get();
        }
        output.collect(key,new IntWritable(count));
    }
}
```

The Map class which extends the public class Mapper<KEYIN,VALUEIN,KEYOUT, VALUEOUT> and implements the Map function.

The Reduce class which extends the public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Reduce function.

## Driver Code in Map Reduce:

```
public class MyDriver{
    public static void main(String args[]) throws Exception{
        Configuration conf= new Configuration();
        Job job = new Job(conf,"My Word Count Program");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        Path outputPath = new Path(args[1]);
        //Configuring the input/output path from the filesystem into
the job
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //deleting the output path automatically from hdfs so that
we don't have to delete it explicitly
        outputPath.getFileSystem(conf).delete(outputPath);
        //exiting the job only if the flag value becomes false
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

## Run the MapReduce code:

The command for running a MapReduce code is:

```
hadoop jar hadoop-mapreduce-example.jar WordCount
/sample/input /sample/output
```

- In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
- We specify the name of the job, the data type of input/output of the mapper and reducer.
- We also specify the names of the mapper and reducer classes.
- The path of the input and output folder is also specified.
- The method setInputFormatClass () is used for specifying how a Mapper will read the input data or what will be the unit of work. Here, we have chosen TextInputFormat so that a single line is read by the mapper at a time from the input text file.

- The main() method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

**Record Reader:** An InputFormat is also responsible for creating the **Input Splits** and dividing them into records. The data is divided into the number of splits (typically 64/128mb) in HDFS. This is called as inputsplit, which is the input that is processed by a single map.

InputFormat class calls the **getSplits()** function and computes splits for each file and then sends them to the **JobTracker**, which uses their storage locations to schedule map tasks to process them on the **TaskTrackers**. Map task then passes the split to the **createRecordReader()** method on InputFormat in task tracker to obtain a RecordReader for that split. The RecordReader load's data from its source and converts into key-value pairs suitable for reading by the **mapper.**

**How the Record Reader Works:** A RecordReader is more than iterator is over records, and map task which uses one record to generate key-value pair, which is passed to the map function.

Then nextKeyValue() will repeat on the context, to populate the key and value objects for the mapper. The key and value is retrieved from the record reader by way of context and passed to the map () method to do its work.

An input to the map function, which is a key-value pair (K, V), gets processed as per the logic mentioned in the map code. When the record gets to the end of the record, the nextKeyValue () method returns false.

# Types of Record Reader:

1. **LineRecordReader:** LineRecordReader in Hadoop is the default RecordReader that textInputFormat provides, it treats each line of the input file as the new value, and associated key is byte offset.

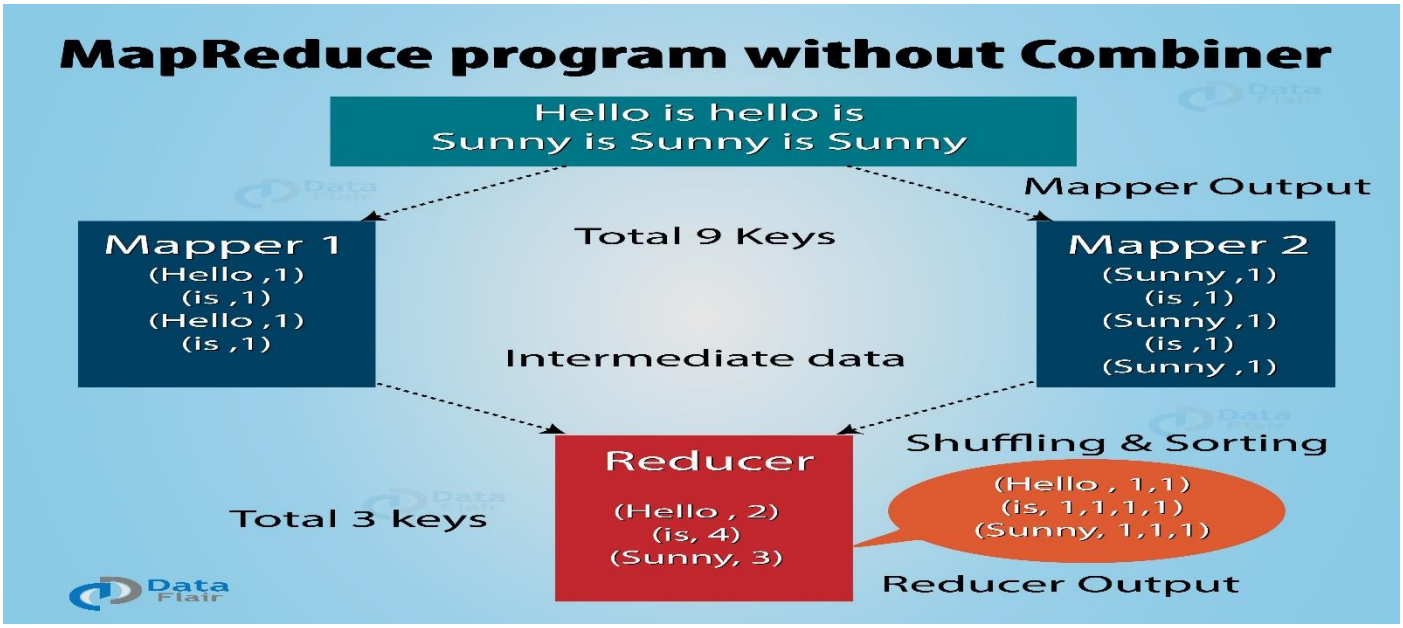
2. **Sequence File Record Reader:** It reads data specified by the header of a sequence file.

**Combiner in Hadoop:** Combiner is also known as "Mini-Reducer" that summarizes the Mapper output record with the same Key before passing to the Reducer.

On a large dataset when we run MapReduce, job. Therefore, Mapper generates large chunks of intermediate data. Then the framework passes this intermediate data on the Reducer for further processing.

This leads to enormous network congestion. The Hadoop framework provides a function known as Combiner that plays a key role in reducing network congestion.

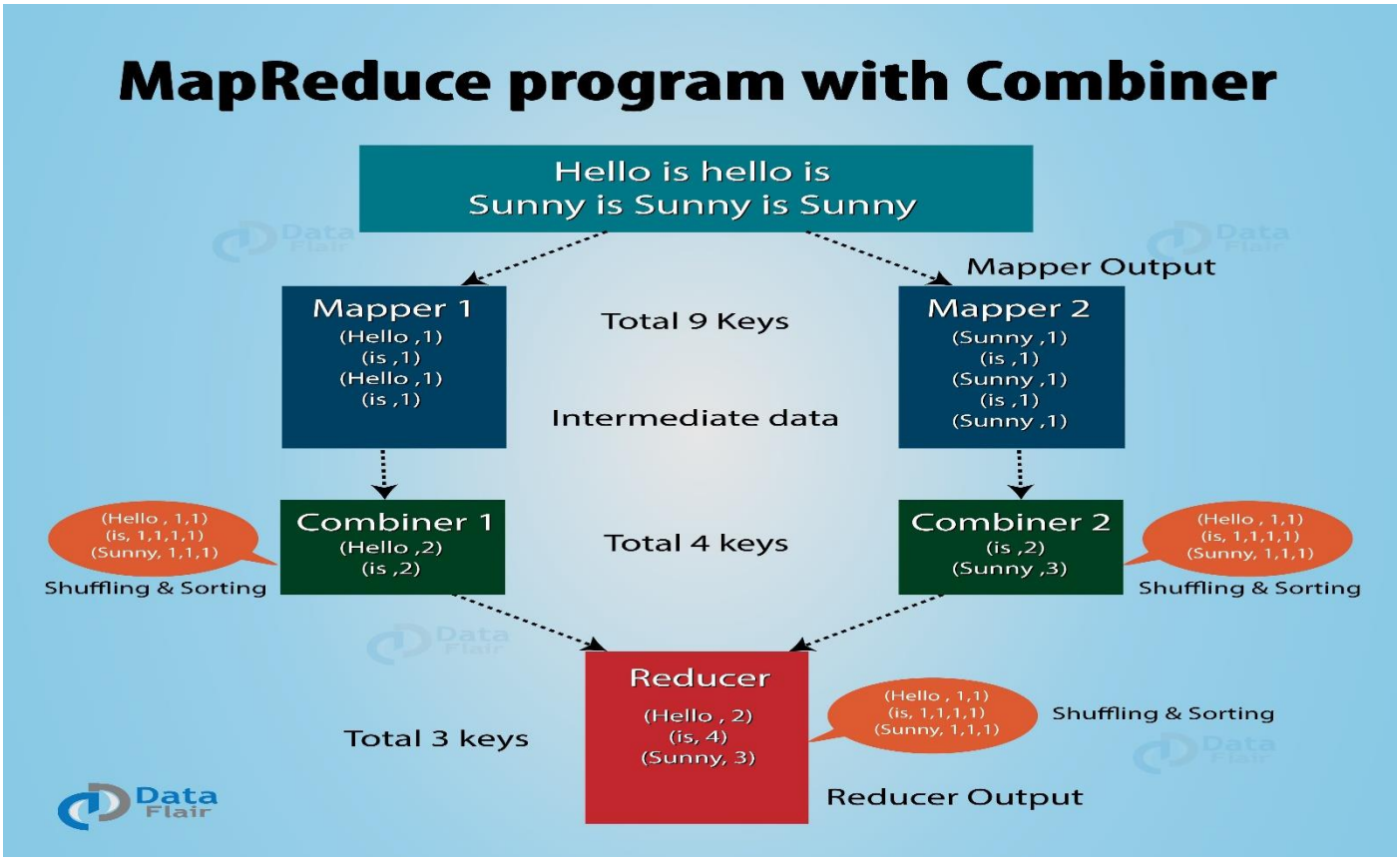
The primary job of Combiner a "Mini-Reducer is to process the output data from the Mapper, before passing it to Reducer. It runs after the mapper and before the Reducer. Its usage is optional.




Here Input is split into two mappers. The framework generates 9 keys from the mappers. Therefore, now we have (9 key/value) intermediate data. Further mapper sends this **key-value** directly

to the reducer. While sending data to the reducer, it consumes some network bandwidth. It takes more time to transfer data to reducer if the size of data is big.

Now in between mapper and reducer if we use a Hadoop combiner, then combiner shuffles intermediate data (9 key/value) before sending it to the reducer and generates 4 key/value pair as an output.




Reducer now needs to process only 4 key/value pair data which is generated from 2 combiners. Thus, reducer gets executed only 4 times to produce final output, which increases the overall performance.

**Partitioner in Hadoop:** Partitioning of the keys of the intermediate map output is controlled by the Partitioner. By hash function, key (or a subset of the key) is used to derive the partition.

According to the key-value each mapper output is partitioned and records having the same key value go into the same partition (within each mapper), and then each partition is sent to a reducer.

Partition class determines which partition a given (key, value) pair will go. Partition phase takes place after map phase and before reduce phase.

Note: The Default Hadoop Partitioner in Hadoop MapReduce is Hash Partitioner which computes a hash value for the key and assigns the partition based on this result.

--The total number of Partitioner that run in Hadoop is equal to the number of reducers i.e. Partitioner will divide the data according

to the number of reducers which is set by JobConf.setNumReduceTasks() method.

The default Partitioner in Hadoop is the HashPartitioner, which has a method called getPartition. It takes key.hashCode() & Integer.MAX_VALUE and finds the modulus using the number of reduce tasks.

Here is the code for the default Partitioner:

```
public class HashPartitioner<K, V> extends Partitioner<K, V> {
    public int getPartition(K key, V value, int numReduceTasks) {
        return (key.hashCode() & Integer.MAX_VALUE) %
numReduceTasks;
    }
}
```

**Creating Custom Partitioner:** Here we want to create the custom Partitioner that partition the key-value pair based on the key length.

```
public class MyPartitoner implements Partitioner<Text, IntWritable>{
    public void configure(JobConf conf){

    }
    public int getPartition(Text key,IntWritable value,int
numOfReducer){
        String s=key.toString();
        if(s.length()==1){
            return 0;
        }
        if(s.length()==2){
            return 1;
        }
        if(s.length()==3){
            return 2;
        }
        if(s.length()==4){
            return 3;
        }
    }
}
```

Here we can configure the Partitioner class in the Driver class main method by using this statement:

```
    conf.setPartitionClass(MyPartitoner.Class);
```

we can configure the number of reducer in the Driver class main method by using this statement:

```
conf.setNumReduceTasks (4);
```

We can work with upto part-00000 to part-99999(i.e. 100000) reducers.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# UNIT- 3 (Hadoop I/O)

## Writeable Interface in Hadoop:-

Writable is an interface in Hadoop. Writable in Hadoop acts as a wrapper class to almost all the primitive data type of Java. That is how **int** of java has become **IntWritable** in Hadoop and **String** of Java has become **Text** in Hadoop.
Writables are used for creating serialized data types in Hadoop.

Hadoop framework definitely needs Writable type of interface in order to perform the following tasks:

- Implement serialization
- Transfer data between clusters and networks
- Store the deserialized data in the local disk of the system

Implementation of writable is similar to implementation of interface in Java. It can be done by simply writing the keyword 'implements' and overriding the default writable method.

Need of Writeable Interface: When we write a key as IntWritable in the Mapper class and send it to the reducer class, there is an intermediate phase between the Mapper and Reducer class i.e., shuffle and sort, where each key has to be compared with many other keys. If the keys are not comparable, then shuffle and sort phase won't be executed or may be executed with high amount of overhead.

If a key is taken as IntWritable by default, then it has comparable feature because of RawComparator acting on that variable. It will compare the key taken with the other keys in the network. This cannot take place in the absence of Writable.

For implementing Writables, we need few more methods in Hadoop:

```
public interface Writable {
void readFields(DataInput in);
void write(DataOutput out);
}
```

Here, readFields reads the data from network and write will write the data into local disk. Both are necessary for transferring data through clusters. DataInput and DataOutput classes (part of java.io) contain methods to serialize the most basic types of data.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Creating Custom Writable Datatype in Hadoop: Suppose we want to make a composite key in Hadoop by combining, two Writables then follow the steps below:

```
public class add implements Writable {
    public int a;
    public int b;
    public add(){
        this.a=a;
        this.b=b;
    }
    public void write(DataOutput out) throws IOException {
        out.writeInt(a);
        out.writeInt(b);
    }
    public void readFields(DataInput in) throws IOException {
        a = in.readInt();
        b = in.readInt();
    }
     public String toString() {
        return Integer.toString(a) + ", " + Integer.toString(b)
    }
}
```

Thus, we can create our custom Writables in a way similar to custom types in Java but with two additional methods, write and read Fields. The custom writable can travel through networks and can reside in other systems.

Writable Comparable and comparators: WritableComparables can be compared to each other, typically via Comparators. Any type, which is to be used as a key in the Hadoop Map-Reduce framework, should implement this interface.

Writable variables in Hadoop have the default properties of Comparable.

How can WritableComparable be implemented in Hadoop?

The implementation of WritableComparable is similar to Writable but with an additional 'CompareTo' method inside it. These are the following implementation of WritableComparable.

```
public interface WritableComparable extends Writable, Comparable
{
    void readFields(DataInput in);
    void write(DataOutput out);
    int compareTo(WritableComparable o)
}
```

Create Custom WritableComparable: if we have made our custom type, Writable rather than WritableComparable our data won't be compared with other data types.

```
public class add implements WritableComparable{
    public int a;
    public int b;
    public add(){
        this.a=a;
        this.b=b;
    }

    public void write(DataOutput out) throws IOException {
        out.writeint(a);
        out.writeint(b);
    }

    public void readFields(DataInput in) throws IOException {
        a = in.readint();
        b = in.readint();
    }

    public int CompareTo(add c){
        int presentValue=this.value;
        int CompareValue=c.value;
        return (presentValue < CompareValue ? -1 :
(presentValue==CompareValue ? 0 : 1));
    }

    public int hashCode() {
        return Integer.IntToIntBits(a)^ Integer.IntToIntBits(b);
    }
}
```

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# Difference between WritableComparable & WritableComparator in Hadoop:

| WritableComparable | WritableComparator |
|---|---|
| WritableComparables can be compared to each other, typicallyvia Comparators. Any type which is to be used as a key in the Hadoop Map-Reduce framework should implement this interface. | A Comparator for WritableComparables. This base implemenation uses the natural ordering. To define alternate orderings, override compare (WritableComparable, WritableComparable) . |
| org.apache.hadoop.io.WritableComparable | org.apache.hadoop.io.WritableComparator |
| For implementing a WritableComparable we must have compareTo method apart from readFields and write methods, as shown below: | A Comparator that operates directly on byte representations of objects. **compare**(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) Compare two |

```
public interface WritableComparable
extends Writable, Comparable
{
void readFields(DataInput in);
void write(DataOutput out);
int compareTo(WritableComparable o)
}
```

objects in binary. b1[s1:l1] is the first object, and b2[s2:l2] is the second object. **Parameters:** b1 – The first byte array. s1 – The position index in b1. The object under comparison's starting index. l1 – The length of the object in b1. b2 – The second byte array. s2 – The position index in b2. The object under comparison's starting index. l2 – The length of the object under comparison in b2. **Returns:** An integer result of the comparison.

Raw Comparator to enhance performance analysis (Speed): A comparator that operates/compares directly on the bytes of data is called as raw comparator.

Raw comparator operates directly on the byte representation of the data.

A Raw comparator is used to enhance the speed of processing the comparison of keys in the Hadoop/map reduce.

The Signature of the Raw comparator is represented like this:

int compare(byte[ ] b1, int s1, int l1, byte [ ], int s2,int l2)

Compare two objects in binary. B1 [s1 to l1 ] is the first object and B2 [s2 to l2 ] second object.

b1 : first byte of array

s1 : first index of the first byte array

l1: length of the objects in b1

b2: second byte array

s2: first index of second byte array

l2: lenth of the objects in b2

org.apache.hadoop.io.RawComparator interface will definitely help speed up your Map/Reduce (MR) Jobs. As you may recall, a MR Job is composed of receiving and sending key-value pairs. The process looks like the following.

(K1,V1) -> Map -> (K2,V2)

(K2,List[V2]) -> Reduce -> (K3,V3)

The key-value pairs (K2,V2) are called the intermediary key-value pairs. They are passed from the mapper to the reducer. Before these intermediary key-value pairs reach the reducer, a shuffle and sort step is performed.

The shuffle is the assignment of the intermediary keys (K2) to reducers and the sort is the sorting of these keys. In this blog, by implementing the RawComparator to compare the intermediary keys, this extra effort will greatly improve sorting. Sorting is improved because the RawComparator will compare the keys by byte. If we did not use RawComparator, the intermediary keys would have to be completely deserialized to perform a comparison.

Writable Classes – Hadoop Data Types: All these primitive writable wrappers have get() and set() methods to read or write the wrapped value. Below is the list of primitive writable data types available in Hadoop.

- BooleanWritable
- ByteWritable
- IntWritable
- VIntWritable
- FloatWritable
- LongWritable
- VLongWritable
- DoubleWritable

In the above list VIntWritable and VLongWritable are used for variable length Integer types and variable length long types respectively.

Array Writable Classes: Hadoop provided two types of array writable classes, one for single-dimensional and another for two-dimensional arrays. But the elements of these arrays must be other writable objects like IntWritable or LongWritable only but not the java native data types like int or float.

- ArrayWritable
- TwoDArrayWritable

Other Writable Classes

NullWritable: NullWritable is a special type of Writable representing a null value. No bytes are read or written when a data type is specified as NullWritable. So, in Mapreduce, a key or a value can be declared as a NullWritable when we don't need to use that field.

ObjectWritable: This is a general-purpose generic object wrapper which can store any objects like Java primitives, String, Enum, Writable, null, or arrays.

Text : Text can be used as the Writable equivalent of java.lang.String and It's max size is 2 GB. Unlike java's String data type, Text is mutable in Hadoop.

BytesWritable: BytesWritable is a wrapper for an array of binary data.

GenericWritable: It is similar to ObjectWritable but supports only a few types. User need to subclass this GenericWritable class and need to specify the types to support.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# UNIT-4 (PIG: Hadoop Programming Made Easier)

**Overview of PIG:** Pig is an open source Apache project for analyzing unstructured or semi-structured data available on Hadoop. It provides a level of abstraction over MapReduce thereby eliminating the need for writing low-level MapReduce code.

In 2006, Pig was started as a research project at Yahoo!. Yahoo developed a new language called Pig Latin for their Hadoop users that fit well between the declarative style of SQL and the procedural style of MapReduce.

Pig was open sourced through Apache and was first released in 2008. Later it became one of the sub projects of Apache Hadoop. By 2009 many companies started adopting Pig for their data processing.

Pig provides an engine for executing various data pipeline operations (load, filter, sort, store etc.) in parallel on Hadoop. It executes these operations with the help of MapReduce.

Pig is rich in terms of data structures and operators. Pig Latin is the language used for writing data flows with Pig.

## Philosophy

- Eats anything : Pig can process or analyse any type of data such as relational, unstructured or nested in nature
- Lives anywhere : Pig is not tied to any particular parallel framework although it is first implemented for Hadoop
- we can customize or extend Pig features by adding user defined functions written in Java or other languages

- Can fly : Pig is capable of processing terabytes of data very quickly

## Key features

- Pig provides various built in operators to perform all kinds of ETL Extract , Transform, Load) operations on the data
- Pig Latin scripts are internally converted to a series of MapReduce jobs
- Pig has an optimizer that optimizes the Pig Scripts to give consistent performance by reordering or rearranging the operations involved in the data flow.

## Pig Execution modes

Pig scripts can be executed in two ways

- Local mode: This mode is suitable for small sized data. Here Pig scripts are executed locally on a single machine. Prototyping and debugging scenarios can be easily implemented with this mode.
- Distributed mode: Here the Pig scripts are executed on a distributed cluster and is suitable for huge sized data. This mode is the default one which translates Pig scripts into series of MapReduce jobs

Interactive mode using Grunt shell:

Interactive mode execution is done with the command shell named 'grunt'

It enables line-by-line execution of the Pig scripts

Grunt shell is invoked using the below commands

In case of distributed mode, pig or pig -x is used to invoke grunt shell

```
[pig_User001@pig_trngsrvr01 ~]$ pig
18/11/26 12:52:12 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
18/11/26 12:52:12 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
18/11/26 12:52:12 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
grunt>
```

- In case of local mode, **pig -x local** is used to invoke grunt shell

```
[pig_User001@pig_trngsrvr01 ~]$ pig -x local
18/11/26 10:44:45 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
18/11/26 10:44:45 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
grunt>
```

Batch mode:

- Pig scripts can be saved with a **.pig** extension
- File with **.pig** extension can be executed using **run** or **exec** commands in grunt shell
- Pig file can be executed along with the mode of execution as below

```
[pig_User001@pig_trngsrvr01 ~]$ pig -x local DemoScript.pig
```

## Embedded mode:

- User can write their own User Defined Functions (UDF) in programming languages like Java
- UDFs can be embedded with Pig scripts for data manipulation

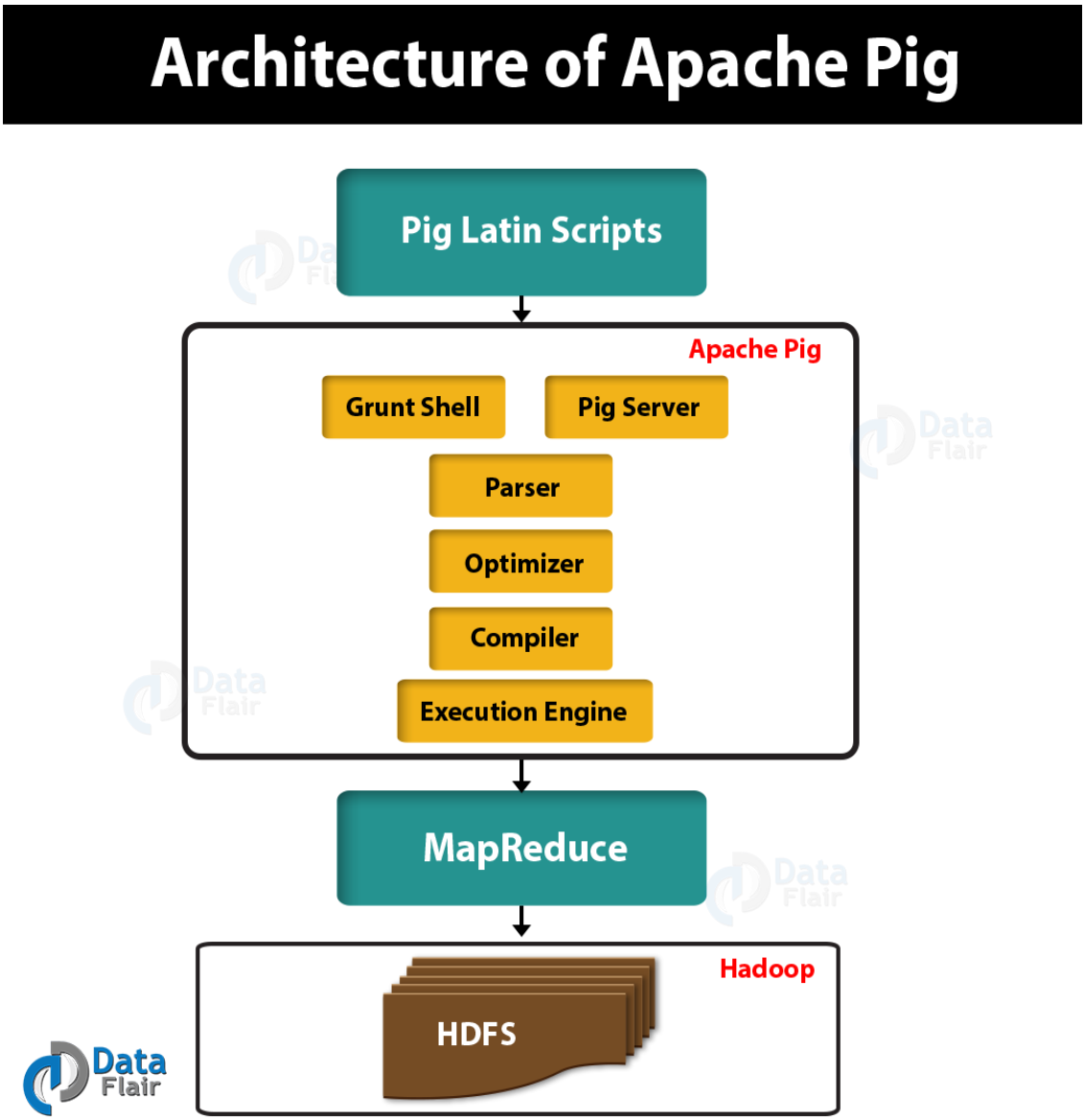
**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

## Going with the Pig Latin Application Flow:

**Pig Architecture:** In Pig, there is a language we use to analyse data in **Hadoop**. That is what we call Pig Latin. In addition, a high-level data processing language offers a rich set of data types and operators to perform several operations on the data.

Moreover, in order to perform a particular task, 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) using Pig.

To produce the desired output, these scripts will go through a series of transformations applied by the Pig Framework, after execution. Pig converts these scripts into a series of **MapReduce** jobs internally. Therefore, it makes the programmer's job easy. Here, is the architecture of Apache Pig.




i. Parser

At first, the Parser handles all the Pig Scripts. Parser checks the syntax of the script, does type checking, and other miscellaneous checks. Afterwards, Parser has output will be a

DAG (directed acyclic graph) that represents the Pig Latin statements as well as logical operators.

The logical operators of the script are represented as the nodes and the data flows are represented as edges in DAG (the logical plan)

ii. Optimizer

Afterwards, the logical plan (DAG) is passed to the logical optimizer. It carries out the logical optimizations further such as projection and push down.

iii. Compiler

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

iv. Execution engine

Eventually, all the MapReduce jobs are submitted to Hadoop in a sorted order. Ultimately, it produces the desired results while these MapReduce jobs are executed on Hadoop.

Pig Latin Data Model: Pig Latin data model is fully nested. Also, it allows complex non-atomic data types like map and tuple. Let's discuss this data model in detail:

i. Atom

Atom is defined as any single value in Pig Latin, irrespective of their data. We can use it as string and number and store it as the string. Atomic values of Pig are int, long, float, double, char array, and byte array. Moreover, a field is a piece of data or a simple atomic value in Pig.

For Example - 'Shubham' or '25'

ii. Tuple

Tuple is a record that is formed by an ordered set of fields. However, the fields can be of any type. In addition, a tuple is similar to a row in a table of RDBMS.

For Example - (Shubham, 25)

iii. Bag

An unordered set of tuples is what we call Bag. To be more specific, a Bag is a collection of tuples (non-unique). Moreover, each tuple can have any number of fields (flexible schema).

Generally, we represent a bag by '{}'.

For Example - {(Shubham, 25), (Pulkit, 35)}

In addition, when a bag is a field in a relation, in that way it is known as the inner bag.

Example - {Shubham, 25, {9826022258, Shubham@gmail.com,}}

iv. Map

A set of key-value pairs is what we call a map (or data map). Basically, the key needs to be of type char array and should be unique. Also, the value might be of any type. And, we represent it by '[]'

For Example - [name#Shubham, age#25]

v. Relation

A bag of tuples is what we call Relation. In Pig Latin, the relations are unordered. Also, there is no guarantee that tuples are processed in any particular order.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

# Scripting With Pig Latin:

## What is Apache Pig Running Scripts?

Basically, to place Pig Latin statements and Pig commands in a single file, we use Pig scripts. It is good practice to identify the file using the *pig extension.

### Executing Pig Script in Batch mode

Further, follow these steps, while we execute Pig script in batch mode.

### Step 1

At very first, write all the required Pig Latin statements and commands in a single file. Then save it as a .pig file.

## Step 2

Afterwards, execute the Apache Pig script. To execute Pig script from the shell (Linux), see:

- **Local mode**

```
$ pig -x local Sample_script.pig
```

- **MapReduce mode**

```
$ pig -x MapReduce Sample_script.pig
```

It is possible to execute it from the Grunt shell as well using the exec command.

```
grunt> exec /sampleScript.pig
```

Executing a Pig Script from HDFS: Also, we can execute a Pig script that resides in the HDFS. Let's assume there is a Pig script with the name Sample_script.pig in the HDFS directory named /pig_data/. To execute it, see.

```
$ pig -x mapreduce hdfs://localhost:9000/pig_data/Sample_script.pig
```

Suppose we have a file Employee_details.txt in HDFS with the following content.

```
Employee_details.txt
001,mehul,chourey,21,9848022337,Hyderabad
002,Ankur,Dutta,22,9848022338,Kolkata
003,Shubham,Sengar,22,9848022339,Delhi
004,Prerna,Tripathi,21,9848022330,Pune
005,Sagar,Joshi,23,9848022336,Bhuwaneshwar
006,Monika,sharma,23,9848022335,Chennai
007,pulkit,pawar,24,9848022334,trivendram
008,Roshan,Shaikh,24,9848022333,Chennai
```

Now, also we have a sample script with the name sample_script.pig, in the same HDFS directory. It contains statements performing operations and transformations on the Employee relation.

```
Employee = LOAD 'hdfs://localhost:9000/pig_data/Employee_details.txt' USING
PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, phone:chararray,
city:chararray);

Employee_order = ORDER Employee BY age DESC;

Employee_limit = LIMIT Employee_order 4;

Dump Employee_limit;
```

- The script will load the data in the file named Employee_details.txt as a relation named Employee, in the first statement.

- Moreover, the script will arrange the tuples of the relation in descending order, based on age, and store it as Employee_order, in the second statement.
- The script will store the first 4 tuples of Employee_order as Employee_limit, in the third statement.
- Ultimately, last and the 4rth statement will dump the content of the relation Employee_limit.

Further, let's execute the sample_script.pig.

$ pig -x mapreduce

hdfs://localhost:9000/pig_data/sample_script.pig

In this way, Pig gets executed and gives you the output like:

```
(7,Pulkit,Pawar,24,9848022334,trivendram)
(8,Roshan,Shaikh,24,9848022333,Chennai)
(5,Sagar,Joshi,23,9848022336,Bhuwaneshwar)
(6,Monika,Sharma,23,9848022335,Chennai)
2015-10-19 10:31:27,446 [main] INFO  org.apache.pig.Main - Pig script completed in 12
minutes, 32 seconds and 751 milliseconds (752751 ms)
```

## Word Count Problem Solution with Pig Script:

1.Load the data from HDFS:

Use Load statement to load the data into a relation .

As keyword used to declare column names, as we dont have any columns, we declared only one column named line.

```
data_input= LOAD 'hdfs://localhost:9000/wc_count/myfile.txt'
as (line:Chararray);
```

2. Convert the Sentence into words:

The data we have is in sentences. So we have to convert that data into words using

TOKENIZE Function

TOKENIZE(line));

(or)

If we have any delimeter like space we can specify as

TOKENIZE(line,' '));

3. Convert Column into Rows: I mean we have to convert every line of data into multiple rows ,for this we have function called FLATTEN in pig.

Using FLATTEN function the bag is converted into tuple, means the array of strings converted into multiple rows.

Words = FOREACH data_input GENERATE FLATTEN(TOKENIZE(line,' ')) AS word;

3. Apply GROUP BY:

We have to count each word occurance, for that we have to group all the words.

```
Grouped = GROUP Words BY word;
```

4. Generate word count:

```
wordcount = FOREACH Grouped GENERATE group, COUNT(Words);
```

5. We can print the word count on console using Dump.

```
DUMP wordcount;
```

Output will be like below.

```
(a,2)
(is,2)
(This,1)
(class,1)
(hadoop,2)
(bigdata,1)
(technology,1)
```

Below is the complete given script. That can be saved with word_count.pig extension.

```
data_input= LOAD 'hdfs://localhost:9000/wc_count/myfile.txt' as (line:Chararray);
Words = FOREACH data_input GENERATE FLATTEN(TOKENIZE(line,' ')) AS word;
Grouped = GROUP Words BY word;
wordcount = FOREACH Grouped GENERATE group, COUNT(Words);
DUMP wordcount;
```

Script can be executed by using following command:

```
grunt >  pig -x mapreduce wordcount.pig
```

# UNIT-V ( Applying Structure to Hadoop Data with Hive)

Introduction to Hive: **Apache Hive** is a data warehouse software project built on top of Apache Hadoop for providing data summarization, query and analysis. Hive gives an SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.

Most of the Data Scientists use SQL queries in order to explore the data and get valuable insights from them. Now, as the volume of data is growing at such a high pace, we need new dedicated tools to deal with big volumes of data.

Initially, Hadoop came up and became one of the most popular tools to process and store big data. But developers were required to write complex map-reduce codes to work with Hadoop. This is Facebook's Apache Hive came to rescue. It is another tool designed to work with Hadoop. We can write SQL like queries in the hive and in the backend it converts them into the map-reduce jobs.

Apache Hive is a data warehouse system developed by Facebook to process a huge amount of structure data in Hadoop. We know that to process the data using Hadoop, we need to right complex map-reduce functions which is not an easy task for most of the developers. Hive makes this work very easy for us.

Introduction

Most of the Data Scientists use SQL queries in order to explore the data and get valuable insights from them. Now, as the volume of data is growing at such a high pace, we need new dedicated tools to deal with big volumes of data.

Initially, Hadoop came up and became one of the most popular tools to process and store big data. But developers were required to write complex map-reduce codes to work with Hadoop. This is Facebook's Apache Hive came to rescue. It is another tool designed to work with Hadoop. We can write SQL like queries in the hive and in the backend it converts them into the map-reduce jobs.

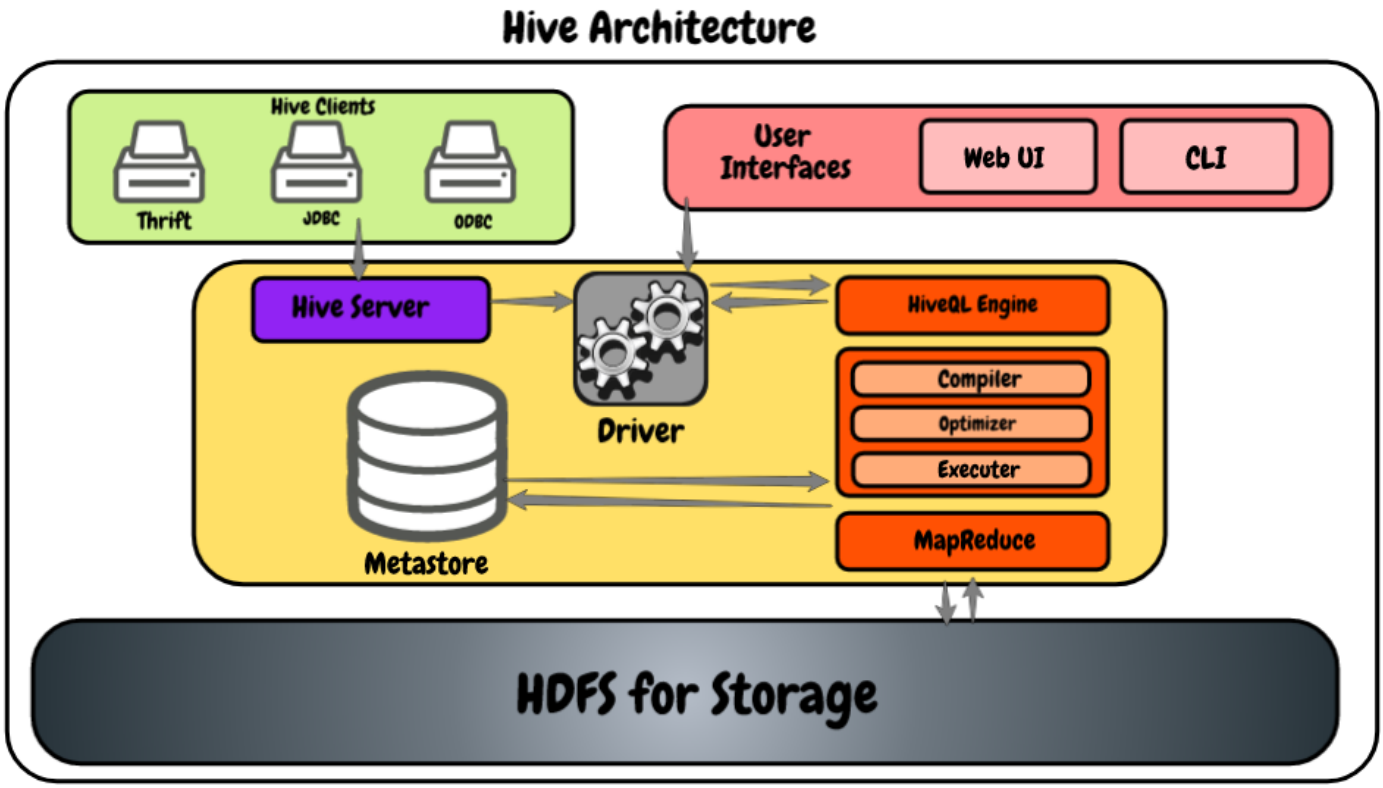
## What is Apache Hive?

Apache Hive is a data warehouse system developed by Facebook to process a huge amount of structure data in Hadoop. We know that to process the data using Hadoop, we need to right complex map-reduce functions which is not an easy task for most of the developers. Hive makes this work very easy for us.

It uses a scripting language called HiveQL which is almost similar to the SQL. So now, we just have to write SQL-like commands and at the backend of Hive will automatically convert them into the map-reduce jobs.

## Apache Hive Architecture:

Let's have a look at the following diagram, which shows the architecture.

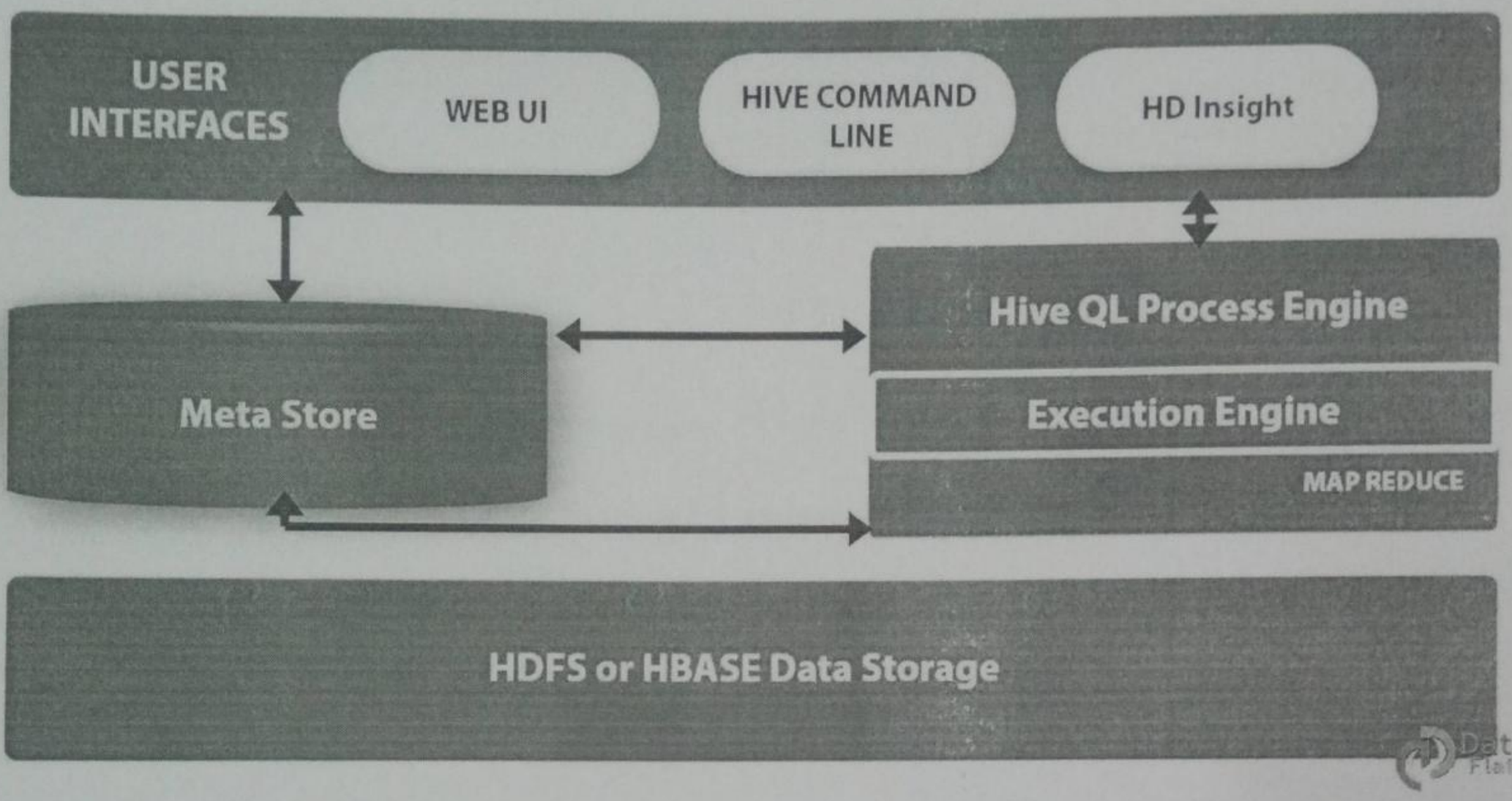



- **Hive Clients:** It allows us to write hive applications using different types of clients such as thrift server, JDBC driver for Java, and Hive applications and also supports the applications that use ODBC protocol.
- **Hive Services:** As a developer, if we wish to process any data, we need to use the hive services such as hive CLI (Command Line Interface). In addition to that, hive also provides a web-based interface to run the hive applications.

- **Hive Driver:** It is capable of receiving queries from multiple resources like thrift, JDBC, and ODBS using the hive server and directly from hive CLI and web-based UI. After receiving the queries, it transfers it to the compiler.
- **HiveQL Engine:** It receives the query from the compiler and converts the SQL like query into the map-reduce jobs.
- **Meta Store:** Here hive stores the meta-information about the databases like schema of the table, data types of the columns, location in the HDFS, etc
- **HDFS:** It is simply the Hadoop distributed file system used to store the data.

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Hive Architecture: The following diagram shows the hive architecture introduce by HIVE.




There are several different units in this component diagram. Now, let's describes each unit:

a. User Interface

As we know it is a data warehouse infrastructure software. It can create interaction between user and HDFS. Moreover, there are various user interfaces that Hive supports. They are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).

b. Meta Store

Basically, to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping, it chooses respective database servers.

c. HiveQL Process Engine

Also, we can say HiveQL is same as SQL Especially, for querying on schema info on the Metastore. In addition, for MapReduce program, it is one of the replacements of the traditional approach. Moreover, we can write a query for MapReduce job and process it, instead of writing MapReduce program in Java.

d. Execution Engine

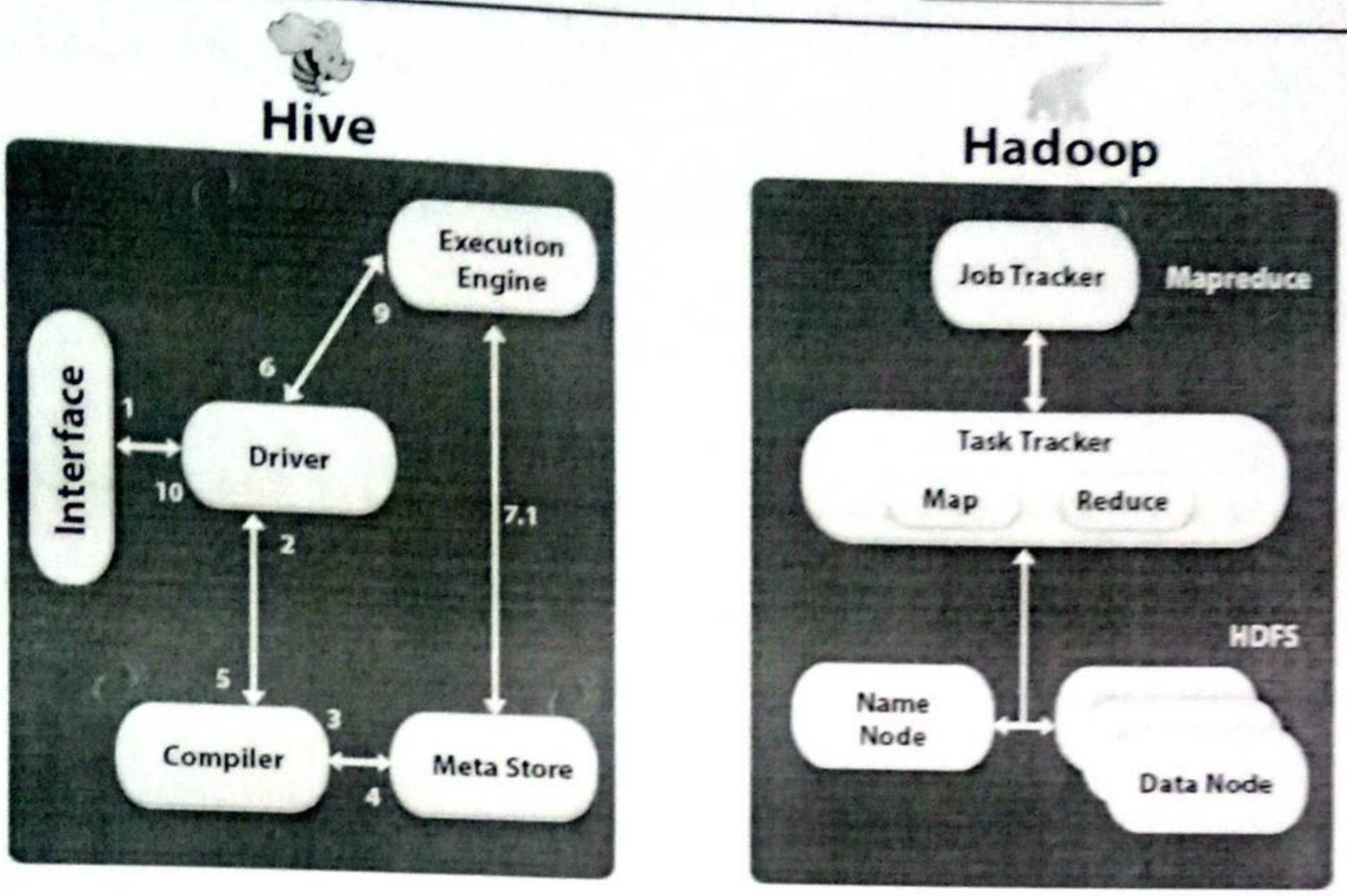
Although, Hive Execution Engine is the conjunction part of HiveQL process Engine and MapReduce. Execution engine processes the query and generates results as same as MapReduce results. Also, it uses the flavour of MapReduce.

e. HDFS or HBase

Basically, to store data into file system Hadoop distributed file system

or HBase is the data storage techniques.

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in




The following table defines how Hive interacts with Hadoop framework.

## Step-1 Execute Query

At very first, the Hive interface (Command Line or Web UI) sends the query to Driver (any database driver such as JDBC, ODBC etc.) to execute.

## Step-2 Get Plan

Afterwards, the driver takes the help of query compiler, which parses the query to check the syntax and query plan or the requirement of the query.

## Step-3 Get Metadata

Further, the compiler sends metadata request to Metastore (any database)

### Step-4 Send Metadata

After that Metastore sends metadata as a response to the compiler

### Step-5 Send Plan

Then the compiler checks the requirement and resends the plan to the driver

However, the parsing and compiling of a query are complete, Up to here

### Step-6 Execute Plan

Further, the driver sends the execution plan to the execution engine

### Step-7 Execute Job

Then, the process of execution job is a MapReduce job, internally Also, the execution engine sends the job to JobTracker, which is in name node and it assigns this job to TaskTracker, which is in data node. Moreover, the query executes MapReduce job, here.

- Metadata Ops

During the execution, the execution engine can execute metadata operations with Metastore.

### Step-8 Fetch Result

While execution is over, the execution engine receives the results from Data nodes

## Step-9 Send Results

After fetching results, execution engine sends those resultant values to the driver

## Step-10 Send Results

At last, the driver sends the results to Hive interfaces

## Features of Apache Hive

There are so many features of Apache Hive. Let's discuss them one by one-

- Hive provides data summarization, query, and analysis in much easier manner.
- Hive supports external tables, which make it possible to process data without actually storing in HDFS.
- Apache Hive fits the low-level interface requirement of Hadoop perfectly
- It also supports partitioning of data at the level of tables to improve performance.
- Hive has a rule-based optimizer for optimizing logical plans.
- It is scalable, familiar, and extensible.
- Using HiveQL does not require any knowledge of programming language, Knowledge of basic SQL query is enough.

- We can easily process structured data in Hadoop using Hive
- Querying in Hive is very simple as it is similar to SQL
- We can also run Ad-hoc queries for the data analysis using Hive.

## Limitation of Apache Hive

Hive has the following limitations-

- Apache does not offer real-time queries and row level updates.
- Hive also provides acceptable latency for interactive data browsing
- It is not good for online transaction processing.
- Latency for Apache Hive queries is generally very high

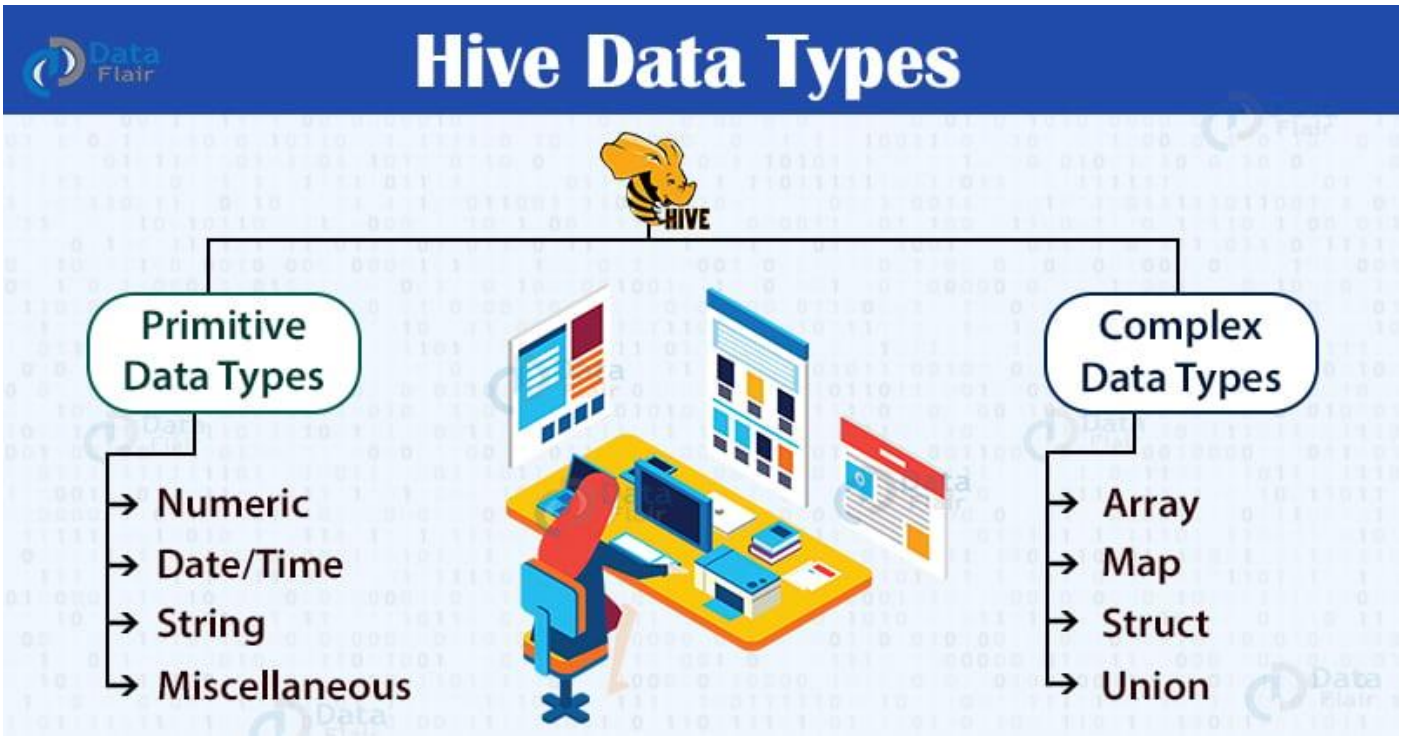
# Hive Clients:

- Command Line
- JDBC
  - JDBC Client Sample Code
  - Running the JDBC Command Line
  - JDBC
  - Python
  - PHP
  - Thrift Java Client
  - ODBC
  - Thrift C++ Client

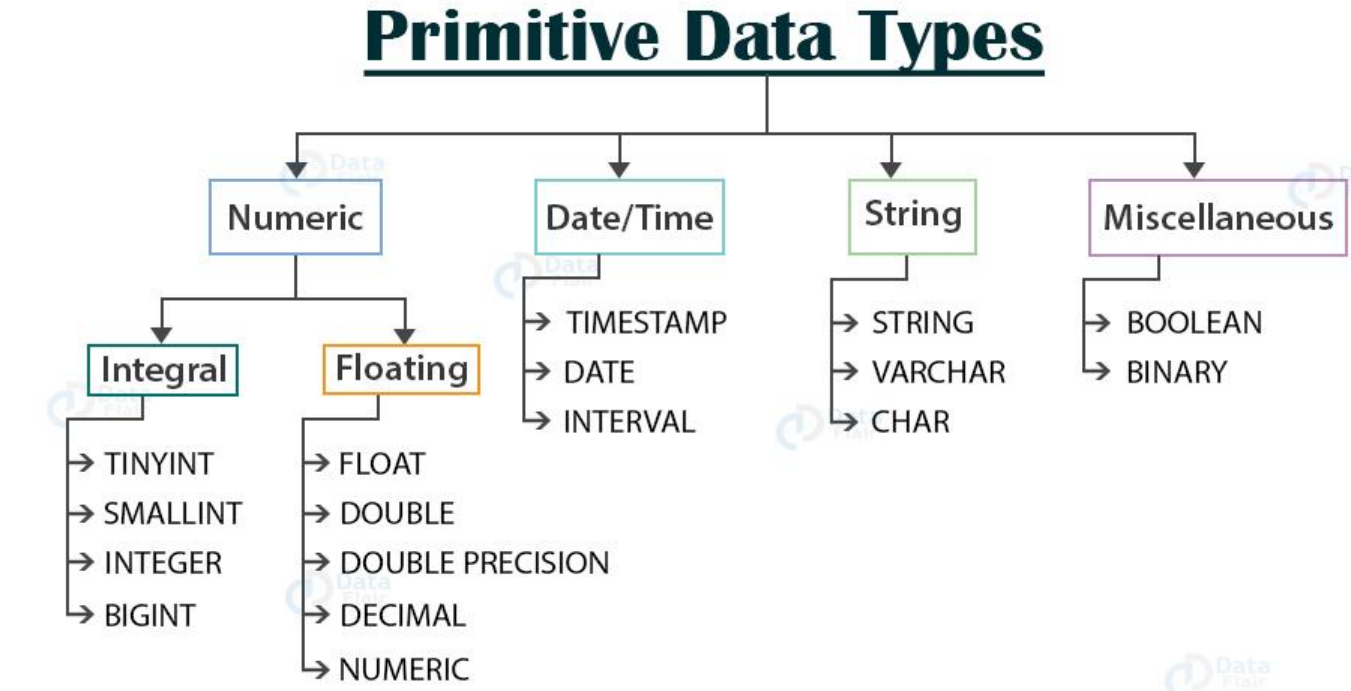
This page describes the different clients supported by Hive. The command line client currently only supports an embedded server. The JDBC and thrift-java clients support both embedded and standalone servers. Clients in other languages only support standalone server.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

**Working With Hive Data Type:** Data Types in Hive specifies the column type in Hive tables, which describes the two categories of Hive Data types that are **primitive data type** and **complex data type.**




## Primitive Data Types:




Integral data type

a. TINYINT – (1-byte signed integer ranging from -128 to 127)

b. SMALLINT – (2-byte signed integer ranging from -32, 768 to 32, 767)

c. INTEGER – (4-byte signed integer ranging from -2, 147, 483, 648 to 2, 147, 483, 647)

d. BIGINT – (8-byte signed integer ranging from -9, 223, 372, 036, 854, 775, 808 to 9, 223, 372, 036, 854, 775, 807)

1.2 Floating data type

a. FLOAT

It is a 4-byte single-precision floating-point number.

b. DOUBLE

It is an 8-byte double-precision floating-point number.

c. DOUBLE PRECISION

It is an alias for DOUBLE. It is only available starting with Hive 2.2.0

d. DECIMAL

It was introduced in Hive 0.11.0. It is based on Java's BigDecimal. DECIMAL types support both scientific and non-scientific notations.

e. NUMERIC

It started with Hive 3.0.0. The NUMERIC data type is the same as the DECIMAL type.

[ps2id id='Date-time' target="/]2. Date/Time data type:

a. TIMESTAMP

Timestamps were introduced in Hive 0.8.0. It supports traditional UNIX timestamp with the optional nanosecond precision.

The supported Timestamps format is yyyy-mm-dd hh:mm:ss[.f...] in the text files.

a. STRING

In Hive, String literals are represented either with the single quotes(' ') or with double-quotes(" ").

Hive uses C-style escaping.

## b. VARCHAR

In Hive, VARCHAR data types are of different lengths, but we have to specify the maximum number of characters allowed in the character string.

If the string value assigned to the varchar is less than the maximum length, then the remaining space will be freed out.

Also, if the string value assigned is more than the maximum length, then the string is silently truncated.

The length of the varchar is between(1 to 65535).

## c. CHAR

CHAR data types are fixed-length.

The values shorter than the specified length are padded with the spaces.

Unlike VARCHAR, trailing spaces are not significant in CHAR types during comparisons.

The maximum length of CHAR is fixed at 255.

[ps2id id='Miscellaneous' target="/]4. Miscellaneous data type

a. BOOLEAN

Boolean types in Hive store either true or false.

b. BINARY

BINARY type in Hive is an array of bytes.

This is all about Hive Primitive Data Types. Let us now study Hive Complex Data Types.

## Complex Data Types:

1. Array:

   **Example:** array('Data','Flair'). The second element is accessed as array[1].

2. Map: **Example:** 'first' -> 'John', 'last' -> 'Deo', represented as map('first', 'John', 'last', 'Deo'). Now 'John' can be accessed with map['first'].

Struct: STRUCT <col_name : data_type [COMMENT col_comment], ...>

**Example:** For a column c3 of type STRUCT {c1 INTEGER; c2 INTEGER}, the c1 field is accessed by the expression c3.c1.

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

**Creating and Managing Databases and Tables:** The table in the hive is consists of multiple columns and records. The table we create in any database will be stored in the sub-directory of that database. The default location where the database is stored on HDFS is /user/hive/warehouse. The way of creating tables in the hive is very much similar to the way we create tables in SQL. We can perform the various operations with these tables like Joins, Filtering, etc.

**Creating Table in Hive:** Firstly create a database so that we can create tables inside it. The command for creating a database is shown below.

**Syntax To Make Database:**

CREATE DATABASE <database-name>;

**Command:**

```
CREATE DATABASE student_detail;

# this will create database student_detail

SHOW DATABASES;

# list down all the available databases

USE student_detail;
```

## Syntax To Create Table in Hive:

```
CREATE TABLE [IF NOT EXISTS] <table-name> (
<column-name>    <data-type>,
<column-name>    <data-type> COMMENT 'Your Comment',
<column-name>    <data-type>,
.
.
.
<column-name>    <data-type>
)
COMMENT 'Add if you want'
LOCATION 'Location On HDFS'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Note:

1. We can add a comment to the table as well as to each individual column.

2. ROW FORMAT DELIMITED shows that whenever a new line is encountered the new record entry will start.

3. FIELDS TERMINATED BY ',' shows that we are using ',' delimiter to separate each column.

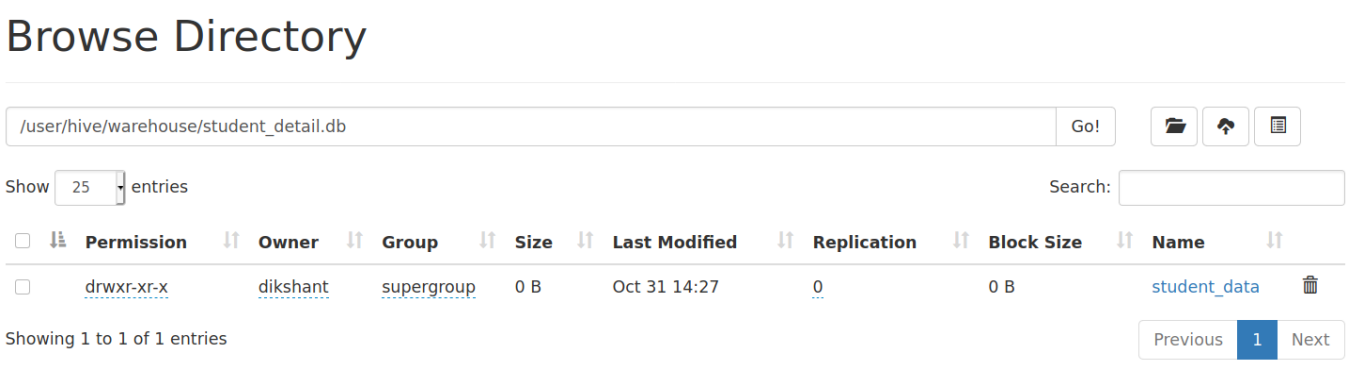
4. We can also override the default database location with the LOCATION option.

So let's create the table student_data in our student_detail database with the help of the command shown below.

```
CREATE TABLE IF NOT EXISTS student_data(
Student_Name STRING COMMENT 'This col. Store the name of student',
Student_Rollno INT COMMENT 'This col. Stores the rollno of student',
Student_Marks FLOAT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

SHOW TABLES IN student_detail;

Finally, let's check the location on HDFS where our student_detail database and student_data table is made. Move to localhost:50070/ for Hadoop 2 and to localhost:9870/ for Hadoop 3. Then Utilities -> Browse the file system and go to /user/hive/warehouse which is a default location where hive databases are created.



**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

**Types of Tables in Hive:** There are two types of Table supported by the Hive.

## Managed Tables

In a managed table, both the table data and the table schema are managed by Hive. The data will be located in a folder named after the table within the Hive data warehouse, which is essentially just a file location in HDFS.

The location is user-configurable when Hive is installed. By managed or controlled we mean that if you drop (delete) a managed table, then Hive will delete both the Schema (the description of the table) and the data files associated with the table. Default location is **/user/hive/warehouse).**

### Creation of Managed Table:

```
CREATE TABLE IF NOT EXISTS stocks (exchange STRING,
symbol STRING,
```

```
price_open FLOAT,
price_high FLOAT,
price_low FLOAT,
price_adj_close FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;
```

## External Tables:

An external table is one where only the table schema is controlled by Hive. In most cases, the user will set up the folder location within HDFS and copy the data file(s) there. This location is included as part of the table definition statement.

When an external table is deleted, Hive will only delete the schema associated with the table. The data files are not affected.

## Syntax:

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

# Managed vs. External Table:

| Managed Table | External Table |
|---|---|
| Hive assumes that it owns the data for managed tables. | For external tables, Hive assumes that it does not manage the data. |
| If a managed table or partition is dropped, the data and metadata associated with that table or partition are deleted. | Dropping the table does not delete the data, although the metadata for the table will be deleted. |
| For **Managed tables**, **Hive** stores data into its warehouse directory | For **External Tables**, **Hive** stores the data in the LOCATION specified during creation of |

| | |
|---|---|
| | the **table**(generally not in warehouse directory) |
| Managed table provides ACID/transnational action support. | External Table does not provide ACID/transactional action support. |
| Statements: ARCHIVE, UNARCHIVE, TRUNCATE, MERGE, CONCATENATE supported | Not supported. |
| Query Result Caching supported(saves the results of an executed Hive query for reuse ). | Not Supported |

**Swami Keshvanand Institute of Technology, Management &Gramothan,**
**Ramnagaria, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

**DML Commands in Hive:** Hive Data Manipulation Language commands are used for inserting, retrieving, modifying, deleting, and updating data in the Hive table.

There are many Hive DML commands like LOAD, INSERT, UPDATE, etc.

1. LOAD Command

The LOAD statement in Hive is used to move data files into the locations corresponding to Hive tables.

- If a LOCAL keyword is specified, then the LOAD command will look for the file path in the local filesystem.
- If the LOCAL keyword is not specified, then the Hive will need the absolute URI of the file.

- In case the keyword OVERWRITE is specified, then the contents of the target table/partition will be deleted and replaced by the files referred by filepath.
- If the OVERWRITE keyword is not specified, then the files referred by filepath will be appended to the table.

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

Example:

```
dataflair@admin1-All-Series: ~/apache-hive-3.1.2-bin
File  Edit  View  Search  Terminal  Help
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH '/home/dataflair/dab' INTO TABLE emp_data;
INFO  : Compiling command(queryId=dataflair_20200207151436_f132ee00-9293-44d3-ac40-e5929be8ed87): LOAD DATA
 LOCAL INPATH '/home/dataflair/dab' INTO TABLE emp_data
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=dataflair_20200207151436_f132ee00-9293-44d3-ac40-e5929be8ed87);
 Time taken: 0.02 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=dataflair_20200207151436_f132ee00-9293-44d3-ac40-e5929be8ed87): LOAD DATA
 LOCAL INPATH '/home/dataflair/dab' INTO TABLE emp_data
INFO  : Starting task [Stage-0:MOVE] in serial mode
INFO  : Loading data to table default.emp_data from file:/home/dataflair/dab
INFO  : Starting task [Stage-1:STATS] in serial mode
INFO  : Completed executing command(queryId=dataflair_20200207151436_f132ee00-9293-44d3-ac40-e5929be8ed87);
 Time taken: 0.153 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
No rows affected (0.181 seconds)
0: jdbc:hive2://localhost:10000>
```

Insert into:

INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select_statement1 FROM from statement;

```
dataflair@admin1-All-Series: ~/apache-hive-3.1.2-bin
File  Edit  View  Search  Terminal  Help
0: jdbc:hive2://localhost:10000> CREATE TABLE IF NOT EXISTS example(id STRING, name S
TRING, dep STRING, state STRING, salary STRING, year STRING);
INFO  : Compiling command(queryId=dataflair_20200224135418_bb6fcff6-48bc-4761-b1f8-a6
484a7a74e5): CREATE TABLE IF NOT EXISTS example(id STRING, name STRING, dep STRING, s
tate STRING, salary STRING, year STRING)
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=dataflair_20200224135418_bb6fcff6-48bc-47
61-b1f8-a6484a7a74e5); Time taken: 0.016 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=dataflair_20200224135418_bb6fcff6-48bc-4761-b1f8-a6
484a7a74e5): CREATE TABLE IF NOT EXISTS example(id STRING, name STRING, dep STRING, s
tate STRING, salary STRING, year STRING)
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=dataflair_20200224135418_bb6fcff6-48bc-47
61-b1f8-a6484a7a74e5); Time taken: 0.069 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
No rows affected (0.094 seconds)
0: jdbc:hive2://localhost:10000>
```

INSERT statement to load data into table "example".

```
dataflair@admin1-All-Series: ~/apache-hive-3.1.2-bin
File  Edit  View  Search  Terminal  Help
0: jdbc:hive2://localhost:10000> INSERT INTO TABLE example SELECT emp.emp_id,emp.emp_name,emp.emp_dep,emp.s
tate,emp.salary,emp.year_of_joining FROM emp_data emp;
```

## Insert Overwrite:

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, ..) [IF NOT EXISTS]] select_statement FROM from_statement;

```
dataflair@admin1-All-Series: ~/apache-hive-3.1.2-bin
File  Edit  View  Search  Terminal  Help
0: jdbc:hive2://localhost:10000> INSERT OVERWRITE TABLE example SELECT dmy.enroll,dmy.name,dmy.department,d
my.state,dmy.salary,dmy.year FROM dummy dmy;
```

By using the SELECT statement, we can verify whether the existing data of the table 'example' is overwritten by the data of table 'dummy' or not.

Insert Value Statement:

```
dataflair@admin1-All-Series: ~/apache-hive-3.1.2-bin
File  Edit  View  Search  Terminal  Help
0: jdbc:hive2://localhost:10000> INSERT INTO TABLE student VALUES (101,'Callen','IT','7.8'), (103,'Joseph',
'CS','8.2'), (105,'Alex','IT','7.9');
```

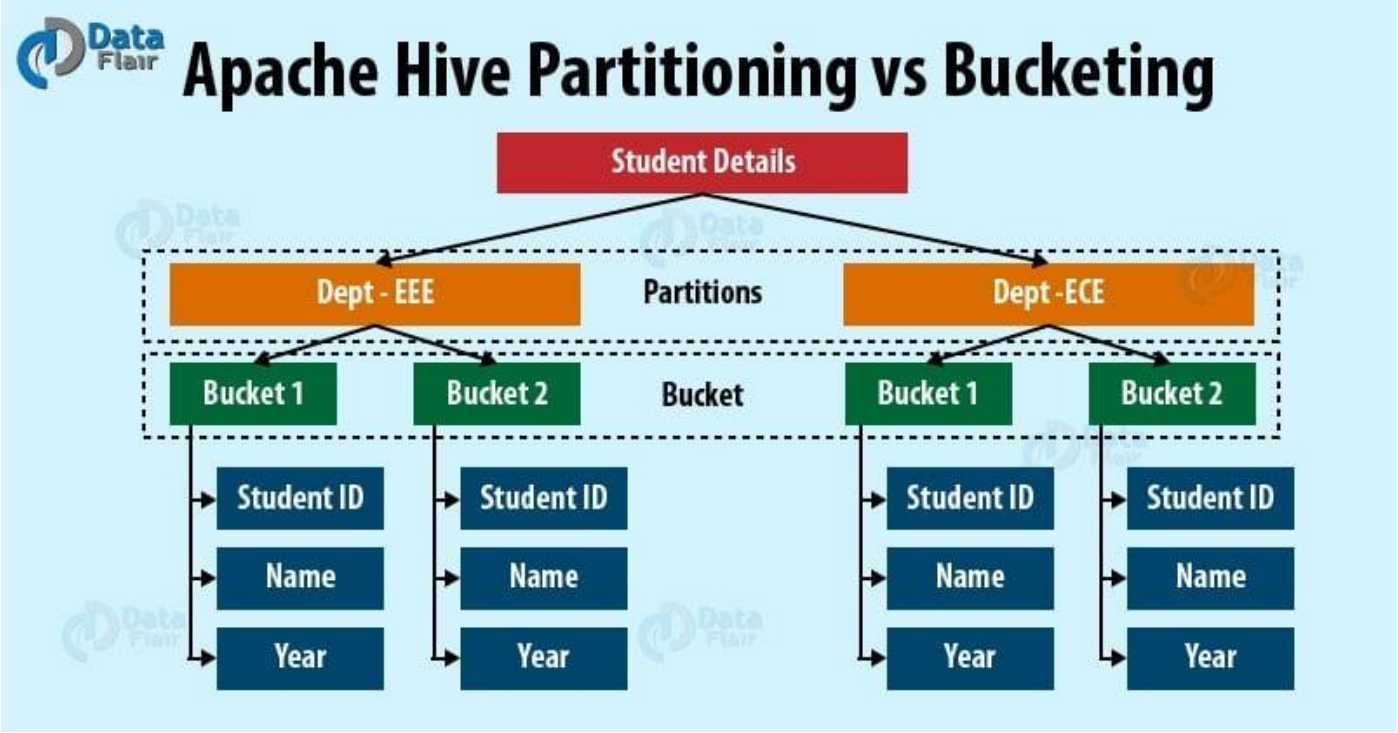
```
dataflair@admin1-All-Series: ~/apache-hive-3.1.2-bin
File  Edit  View  Search  Terminal  Help
0: jdbc:hive2://localhost:10000> SELECT * FROM student;
INFO  : Compiling command(queryId=dataflair_20200224171936_37d1120e-e1ce-4e85-aa8a-51ae9842c276): SELECT *
FROM student
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:student.roll_no, type:int, comment:nul
l), FieldSchema(name:student.name, type:string, comment:null), FieldSchema(name:student.branch, type:varcha
r(15), comment:null), FieldSchema(name:student.cgpa, type:varchar(5), comment:null)], properties:null)
INFO  : Completed compiling command(queryId=dataflair_20200224171936_37d1120e-e1ce-4e85-aa8a-51ae9842c276);
 Time taken: 0.085 seconds
INFO  : Executing command(queryId=dataflair_20200224171936_37d1120e-e1ce-4e85-aa8a-51ae9842c276): SELECT *
FROM student
INFO  : Completed executing command(queryId=dataflair_20200224171936_37d1120e-e1ce-4e85-aa8a-51ae9842c276);
 Time taken: 0.0 seconds
INFO  : OK
+------------------+----------------+-------------------+----------------+
| student.roll_no  | student.name   | student.branch    | student.cgpa   |
+------------------+----------------+-------------------+----------------+
| 101              | Callen         | IT                | 7.8            |
| 103              | Joseph         | CS                | 8.2            |
| 105              | Alex           | IT                | 7.9            |
+------------------+----------------+-------------------+----------------+
3 rows selected (0.137 seconds)
0: jdbc:hive2://localhost:10000>
```

4. DELETE command

The DELETE statement in Hive deletes the table data. If the WHERE clause is specified, then it deletes the rows that satisfy the condition in where clause

DELETE FROM tablename [WHERE expression];

**Hive Partitioning vs Bucketing:** Apache Hive is an open source data warehouse system used for querying and analyzing large datasets. Data in Apache Hive can be categorized into Table, Partition, and Bucket. The table in Hive is logically made up of the data being stored.




# i. Partitioning and Bucketing Commands in Hive

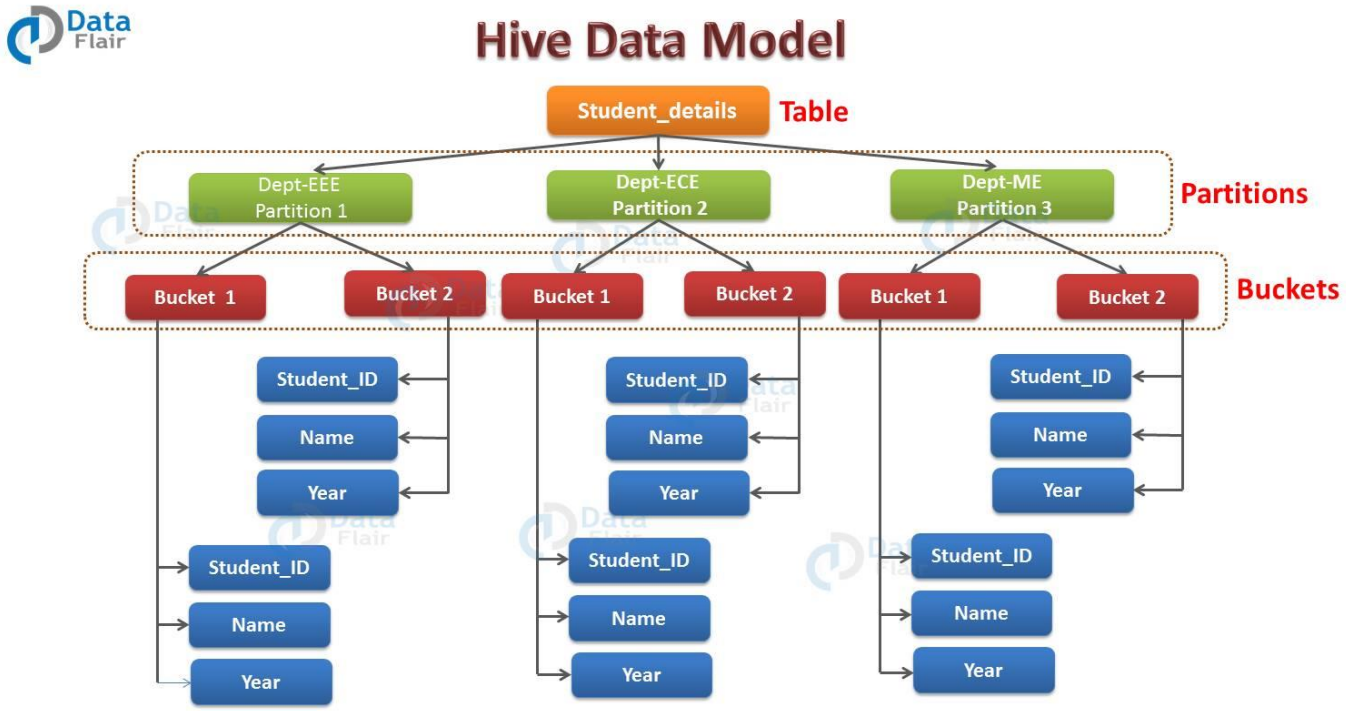
**a) Partitioning**

The Hive command for Partitioning is:

CREATE TABLE table_name (column1 data_type, column2 data_type) PARTITIONED BY (partition1 data_type, partition2 data_type,....);

b). Bucketing:

CREATE TABLE table_name PARTITIONED BY (partition1 data_type, partition2 data_type,....) CLUSTERED BY (column_name1, column_name2, ...) SORTED BY (column_name [ASC|DESC], ...)] INTO num_buckets BUCKETS;




## a) Hive Partitioning Example

For example, we have a table employee_details containing the employee information of some company like **employee_id, name, department, year,** etc. Now, if we want to perform partitioning on the basis of department column.

Then the information of all the employees belonging to a particular department will be stored together in that very partition. Physically, a partition in Hive is nothing but just a sub-directory in the table directory.

For example, we have data for three departments in our employee_details table – Technical, Marketing and Sales. Thus we will have three partitions in total for each of the departments as we can see clearly in diagram below.

For each department we will have all the data regarding that very department residing in a separate sub – directory under the table directory.

So for example, all the employee data regarding Technical departments will be stored in user/hive/warehouse/employee_details/dept.=Technical. So, the

queries regarding Technical employee would only have to look through the data present in the Technical partition.

Therefore from above example, we can conclude that partitioning is very useful. It reduces the query latency by scanning only **relevant** partitioned data instead of the whole data set.

## b) Hive Bucketing Example

Hence, from the above diagram, we can see that how each partition is bucketed into 2 buckets. Therefore each partition, says Technical, will have two files where each of them will be storing the Technical employee's data

- ### a) Pros and Cons of Hive Partitioning

  **Pros:**

- It 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.

Cons:

- There is the possibility of too many small partition creations- too many directories.
- Partition is effective for low volume data. 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.

## b) Pros and Cons of Hive Bucketing

### Pros:

- It provides faster query response like portioning.
- In bucketing due to equal volumes of data in each partition, joins at Map side will be quicker.

### Cons:

- We can define a number of buckets during table creation. But loading of an equal volume of data has to be done manually by programmers.