SWAMI KESHVANAND INSTITUTE OF TECHNOLOGY, MANAGEMENT AND GRAMOTHAN, JAIPUR

Hands on Lab Guide
(Lab Manual)

BIG DATA ANALYTICS LAB
IV Year B.Tech. VIII SEM
(Course Code: 8CS4-21)
Session 2020-2021

Department of Computer Science & Engineering
SKIT, JAIPUR

Swami Keshvanand Institute of Technology,

Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017

LAB MANUAL

Big Data Analytics Lab (8CS4-21)

VERSION 1.0

	AUTHOR / OWNER	REVIEWED BY	APPROVED BY
NAME	Dr. Pankaj Dadheech, Mr. Sunil Dhankhar Ms. Anjana Sangwan	Mr. M. K. Beniwal	Prof. (Dr.) Mukesh Kumar Gupta
DESIGNATION	Associate Professor	Associate Professor	HOD (CSE)
SIGNATURE & REVIEW DATE			
SIGNATURE & REVIEW DATE			
SIGNATURE & REVIEW DATE			

Department of Computer Science & Engineering,
Jaipur – 302017, Rajasthan (INDIA)
E-Mail: hodcs@skit.ac.in, URL: www.skit.ac.in

LAB RULES

Responsibilities of Users

Users are expected to follow some fairly obvious rules of conduct:

Always:

- Enter the lab on time and leave at proper time.
- Wait for the previous class to leave before the next class enters.
- Keep the bag outside in the respective racks.
- Utilize lab hours in the corresponding.
- Turn off the machine before leaving the lab unless a member of lab staff has specifically told you not to do so.
- Leave the labs at least as nice as you found them.
- If you notice a problem with a piece of equipment (e.g. a computer doesn't respond) or the room in general (e.g. cooling, heating, lighting) please report it to lab staff immediately. Do not attempt to fix the problem yourself.

Never:

- Don't abuse the equipment.
- Do not adjust the heat or air conditioners. If you feel the temperature is not properly set, inform lab staff; we will attempt to maintain a balance that is healthy for people and machines.
- Do not attempt to reboot a computer. Report problems to lab staff.
- Do not remove or modify any software or file without permission.
- Do not remove printers and machines from the network without being explicitly told to do so by lab staff.
- Don't monopolize equipment. If you're going to be away from your machine for more than 10 or 15 minutes, log out before leaving. This is both for the security of your account, and to ensure that others are able to use the lab resources while you are not.
- Don’t use internet, internet chat of any kind in your regular lab schedule.
- Do not download or upload of MP3, JPG or MPEG files.
- No games are allowed in the lab sessions.
- No hardware including USB drives can be connected or disconnected in the labs without prior permission of the lab in-charge.
- No food or drink is allowed in the lab or near any of the equipment. Aside from the fact that it leaves a mess and attracts pests, spilling anything on a keyboard or other piece of computer equipment could cause permanent, irreparable, and costly damage. (and in fact *has*) If you need to eat or drink, take a break and do so in the canteen.
- Don’t bring any external material in the lab, except your lab record, copy and books.
- Don’t bring the mobile phones in the lab. If necessary then keep them in silence mode.
- Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.

If you are having problems or questions, please go to either the faculty, lab in-charge or the lab supporting staff. They will help you. We need your full support and cooperation for smooth functioning of the lab.

INSTRUCTIONS

BEFORE ENTERING IN THE LAB

- All the students are supposed to prepare the theory regarding the next experiment/ Program.
- Students are supposed to bring their lab records as per their lab schedule.
- Previous experiment/program should be written in the lab record.
- If applicable trace paper/graph paper must be pasted in lab record with proper labeling.
- All the students must follow the instructions, failing which he/she may not be allowed in the lab.

WHILE WORKING IN THE LAB

- Adhere to experimental schedule as instructed by the lab in-charge/faculty.
- Get the previously performed experiment/ program signed by the faculty/ lab in charge.
- Get the output of current experiment/program checked by the faculty/ lab in charge in the lab copy.
- Each student should work on his/her assigned computer at each turn of the lab.
- Take responsibility of valuable accessories.
- Concentrate on the assigned practical and don't play games.
- If anyone is caught red-handed carrying any equipment of the lab, then he/she will have to face serious consequences.

Marking/Assessment System

Total Marks -50

Distribution of Marks - 30 (Sessional)

Attendance	File Work	Performance	Viva	Total
05	05	10	10	30

Distribution of Marks - 20 (End Term) Depends on Examiner

File Work	Performance	Viva	Total
05	10	05	20

RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

IV Year-VIII Semester: B.Tech. Computer Science and Engineering

8CS4-21: Big Data Analytics Lab

Credit: 2 **Max. Marks: 50(IA:30, ETE:20)**

0L+0T+2P **End Term Exam: 2 Hours**

SN	List of Experiments
1	Implement the following Data structures in Java i) Linked Lists ii) Stacks iii) Queues iv) Set v) Map
2	Perform setting up and Installing Hadoop in its three operating modes: a) Standalone, Pseudo distributed, Fully distributed.
3	Implement the following file management tasks in Hadoop: • Adding files and directories • Retrieving files • Deleting files Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.
4	Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.
5	Write a Map Reduce program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi structured and record-oriented.
6	Implement Matrix Multiplication with Hadoop Map Reduce.
7	Install and Run Pig then write Pig Latin scripts to sort, group, join, project, and filter your data.
8	Install and Run Hive then use Hive to create, alter, and drop databases, tables, views, functions, and indexes.
9	Solve some real life big data problems.

Beyond Syllabus:

1. Write a Program for Developing and Running a Spark WordCount Application.

INDEX

LAB PLAN

Total number of experiment 10

Total number of turns required 10

Number of turns required for

Experiment Number	Turns	Scheduled Day
Exp. 1	1	Day 1
Exp. 2	1	Day 2
Exp. 3	1	Day 3
Exp. 4	1	Day 4
Exp. 5	1	Day 5
Exp. 6	1	Day 6
Exp. 7	1	Day 7
Exp. 8	1	Day 8
Exp. 9	1	Day 9
Exp. 10	1	Day 10

Distribution of Lab Hours:

Attendance	05 minutes
Explanation of features of language	15 minutes
Explanation of experiment	15 minutes
Performance of experiment	70 minutes
Viva / Quiz / Queries	15 minutes
Total	**120 Minutes (2 Hrs.)**

Lab Objective and Outcome

Objective

The objective of this course is to impart necessary and practical knowledge of components of Big Data Analytics and develop skills required to build real-life based projects.

- Understand and implement the basics of data structures like Linked list, stack, queue, set and map in Java.
- Demonstrate the knowledge of big data analytics and implement different file management task in Hadoop
- Understand Map Reduce Paradigm and develop data applications using variety of systems.
- Analyze and perform different operations on data using Pig Latin scripts.
- Illustrate and apply different operations on relations and databases using Hive.
- Understand Big Data for Business Intelligence.
- Learn business case studies for Big Data Analytics.
- Understand NoSQL Big Data Management.
- Perform Map-Reduce Analytics using Hadoop and related tool.

Course Outcomes

After completion of this course, students will be able to –

8CS4-21.1	**Understand** and implement the basics of data structures like linked list, stack, queue, set and map in Java.
8CS4-21.2	**Demonstrate** the knowledge of Big Data Analytics and implement different file management task in Hadoop.
8CS4-21.3	**Understand** Map Reduce Paradigm and develop data applications using variety of Systems.
8CS4-21.4	**Analyze** and perform different operations on data using Pig Latin Scripts.
8CS4-21.5	**Illustrate** and apply different operations on relations and databases using Hive.

Experiments

Experiment – 1

Aim: Implement the following Data structures in Java
i) Linked Lists ii) Stacks iii) Queues iv) Set v) Map

i) Linked List :

```
import java.util.*;
public class LinkedListDemo {
public static void main(String args[]) {
// create a linked list
LinkedList ll = new LinkedList();
// add elements to the linked list
ll.add("F");
ll.add("B");
ll.add("D");
ll.add("E");
ll.add("C");
ll.addLast("Z"):
ll.addFirst("A");
ll.add(1, "A2");
System.out.println("Original contents of ll: " +
ll); // remove elements from the linked list
ll.remove("F");
ll.remove(2);
System.out.println("Contents of ll after deletion: " +
ll); // remove first and last elements
  ll.removeFirst();
  ll.removeLast();
System.out.println("ll after deleting first and last: "+ ll);
// get and set a value
Object val = ll.get(2);
ll.set(2, (String) val + " Changed");
System.out.println("ll after change: " + ll);
}
} Output:
```

Original contents of ll: [A, A2, F, B, D, E, C, Z] Contents of ll after deletion: [A, A2, D, E, C, Z] ll after deleting first and last: [A2, D, E, C]
ll after change: [A2, D, E Changed, C]

ii) Stacks Program:

```
import java.util.*;
public class StackDemo {
  static void showpush(Stack st, int a) {
    st.push(new Integer(a));
    System.out.println("push(" + a + ")");
    System.out.println("stack: " + st);
  }
  static void showpop(Stack st) {
    System.out.print("pop -> ");
    Integer a = (Integer) st.pop();
    System.out.println(a);
    System.out.println("stack: " + st);
  }
  public static void main(String args[]) {
    Stack st = new Stack();
    System.out.println("stack: " + st);
    showpush(st, 42);
    showpush(st, 66);
    showpush(st, 99);
    showpop(st);
    showpop(st);
```

```
    showpop(st);
    try {
      showpop(st);
    }catch (EmptyStackException e) {
      System.out.println("empty stack");
    }
  }
}
```

output:

```
stack: [ ]
push(42)
stack: [42]
push(66)
stack: [42, 66]
push(99)
stack: [42, 66, 99]
pop -> 99
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: [ ]
pop -> empty stack
```

iii) Queues

```
// Java program to demonstrate working of Queue interface in Java
```

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueExample
{
  public static void main(String[] args)
  {
    Queue<Integer> q = new LinkedList<>();
    // Adds elements {0, 1, 2, 3, 4} to queue
    for (int i=0; i<5; i++)
     q.add(i);
    // Display contents of the queue.
    System.out.println("Elements of queue-"+q);
     // To remove the head of queue.
    int removedele = q.remove();
    System.out.println("removed element-" + removedele);
    System.out.println(q);
    // To view the head of queue
    int head = q.peek();
System.out.println("head of queue-" + head);
    // Rest all methods of collection interface,
    // Like size and contains can be used with this
    // implementation.
    int size = q.size();
    System.out.println("Size of queue-" + size);
  }
}
```

Output:

Elements of queue-[0, 1, 2, 3, 4]

removed element-0

[1, 2, 3, 4]

head of queue-1

Size of queue-4

iv) Set

```
import java.util.*; public
class SetDemo {
public static void main(String args[]) {
int count[] = {34, 22,10,60,30,22};
Set<Integer> set = new HashSet<Integer>();
try{
for(int i = 0; i<5; i++){
set.add(count[i]);
}
System.out.println(set);
TreeSet sortedSet = new TreeSet<Integer>(set);
System.out.println("The sorted list is:");
System.out.println(sortedSet);
System.out.println("The First element of the set is: "+(Integer)sortedSet.first());
System.out.println("The last element of the set is: "+(Integer)sortedSet.last());
}
catch(Exception e){}
}
}
```

Output:

[34, 22, 10, 60, 30]

The sorted list is:

[10, 22, 30, 34, 60]

The First element of the set is: 10

The last element of the set is: 60

v) Map Program:

```
import java.awt.Color;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
public class MapDemo
{

public static void main(String[] args)
{
Map<String, Color> favoriteColors = new HashMap<String, Color>();
favoriteColors.put("sai", Color.BLUE); favoriteColors.put("Ram",
Color.GREEN); favoriteColors.put("krishna", Color.RED);
favoriteColors.put("narayana", Color.BLUE); // Print all keys and values in
the map
Set<String> keySet = favoriteColors.keySet(); for (String key :
keySet) {
Color value = favoriteColors.get(key);
System.out.println(key + " : " + value);
}
  }}
```

Output:

narayana : java.awt.Color[r=0,g=0,b=255]

sai : java.awt.Color[r=0,g=0,b=255]

krishna : java.awt.Color[r=255,g=0,b=0]

Ram : java.awt.Color[r=0,g=255,b=0]

Experiment – 2

Aim: Perform setting up and Installing Hadoop in its three operating modes:
Standalone, Pseudo distributed, Fully distributed.

1. **Installation of Hadoop:**

Hadoop software can be installed in three modes of operation:

- **Stand Alone Mode**: Hadoop is a distributed software and is designed to run on a commodity of machines. However, we can install it on a single node in stand-alone mode. In this mode, Hadoop software runs as a single monolithic java process. This mode is extremely useful for debugging purpose. You can first test run your Map-Reduce application in this mode on small data, before actually executing it on cluster with big data.

- **Pseudo Distributed Mode**: In this mode also, Hadoop software is installed on a Single Node. Various daemons of Hadoop will run on the same machine as separate java processes. Hence all the daemons namely NameNode, DataNode, SecondaryNameNode, JobTracker, TaskTracker run on single machine.

- **Fully Distributed Mode**: In Fully Distributed Mode, the daemons NameNode, JobTracker, SecondaryNameNode (Optional and can be run on a separate node) run on the Master Node. The daemons DataNode and TaskTracker run on the Slave Node.

Hadoop Installation: Ubuntu Operating System in stand-alone mode

Steps for Installation

1. sudo apt-get update
2. 2. In this step, we will install latest version of JDK (1.8) on the machine.
   The Oracle JDK is the official JDK; however, it is no longer provided by Oracle as a default installation. for Ubuntu. You can still install it using apt-get. To install any version, first execute the following
   commands:
   a. sudo apt-get install python-software properties
   b. sudo add-apt-repository ppa:webupd8team/

java

c. sudo apt-get update

Then, depending on the version you want to install, execute one of the following commands:

Oracle JDK 7: sudo apt-get install oraclejava7-installer

Oracle JDK 8: sudo apt-get install oraclejava8-installe

3. . Now, let us setup a new user account for Hadoop installation. This step is optional, but recommended because it gives you flexibility to have a separate account for Hadoop installation by separating this installation from other software installation

a. sudo adduser hadoop_dev (Upon executing this command, you will prompted to enter the new password for this user. Please enter the password and enter other details. Don't forget to save the details at the end)

b. su- hadoop_dev (Switches the user from current user to the new user created i.e Hadoop_dev)

4. Download the latest Hadoop distribution.

a. Visit this URL and choose one of the mirror sites. You can copy the download link and also use "wget" to download it from command prompt:

Wgethttp://
apache.mirrors.lucidnetworks.net/hadoop/
common/hadoop-2.7.0/hadoop-2.7.0.tar.gz

5. Untar the file :

```
tar xvzf hadoop-2.7.0.tar.gz
```

6. Rename the folder to hadoop2

```
mv hadoop-2.7.0 hadoop2
```

7. Edit configuration file /home/hadoop_dev/ hadoop2/etc/hadoop/hadoop-env.sh and set JAVA_HOME in that file.

a.

```
vim /home/hadoop_dev/hadoop2/etc/hadoop/
```

hadoop-env.sh

b. uncomment JAVA_HOME and update it following line:

export JAVA_HOME=/usr/lib/jvm/java-8-oracle (Please check for your relevant java installation and set this value accordingly. Latest versions of Hadoop require > JDK1.7)

8. Let us verify if the installation is successful or not (change to home directory cd /home/hadoop_dev/hadoop2/) :

a. bin/hadoop (running this command should prompt you with various options)

9. This finishes the Hadoop setup in stand-alone mode.

10. Let us run a sample hadoop programs that is provided to you in the download package:

$ mkdir input (create the input directory)

$ cp etc/hadoop/*.xml input (copy over all the xml files to input folder)

$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar grep input output 'dfs[a-z.]+' (grep/find all the files matching the pattern ‘dfs[a-z.]+’ and copy

those files to output directory)

$ cat output/* (look for the output in the output

directory that Hadoop creates for you).

Hadoop Installation: Psuedo Distributed Mode (Locally)

Steps for Installation

1. Edit the file /home/Hadoop_dev/hadoop2/etc/ hadoop/core-site.xml as below:

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Note: This change sets the namenode ip and port.

2. Edit the file /home/Hadoop_dev/hadoop2/etc/ hadoop/hdfs-site.xml as below:

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

Note: This change sets the default replication count for blocks used by HDFS.

3. We need to setup password less login so that the master will be able to do a password-less ssh to start the daemons on all the slaves.

Check if ssh server is running on your host or not:

a. ssh localhost (enter your password and if you are able to login then ssh server is running)

b. In step a. if you are unable to login, then install ssh as follows:

sudo apt-get install ssh

c. Setup password less login as below:

i. ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa

ii. cat ~/.ssh/id_dsa.pub >> ~/.ssh/

authorized_key

4. We can run Hadoop jobs locally or on YARN in this mode. In this Post, we will focus on running the jobs **locally.**

5. Format the file system. When we format namenode it formats the meta-data related to data nodes. By doing that, all the information on the datanodes are lost and they can be reused for new data:

a. bin/hdfs namenode –format

6. Start the daemons

a. sbin/start-dfs.sh (Starts NameNode and DataNode)

You can check If NameNode has started successfully or not by using the following web interface: http://0.0.0.0:50070 . If you are unable to see this, try to check the logs in the /home/ hadoop_dev/hadoop2/logs folder.

7.You can check whether the daemons are running or not by issuing Jps command.

8. This finishes the installation of Hadoop in pseudo distributed mode.

9. Let us run the same example we can in the previous blog post:

i) Create a new directory on the hdfs

bin/hdfs dfs -mkdir –p /user/hadoop_dev

ii) Copy the input files for the program to hdfs:

bin/hdfs dfs -put etc/hadoop input

Run the program:

iv) View the output on hdfs:

bin/hdfs dfs -cat output/*

10. Stop the daemons when you are done executing the jobs, with the below command:

sbin/stop-dfs.sh

Hadoop Installation – Psuedo Distributed Mode (YARN)

Steps for Installation

1. Edit the file /home/hadoop_dev/hadoop2/etc/ hadoop/mapred-site.xml as below:

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

2. Edit the fie /home/hadoop_dev/hadoop2/etc/ hadoop/yarn-site.xml as below:

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce_shuffle</value>

</property>

</configuration>

Note: This particular configuration tells MapReduce how to do its shuffle. In this case it uses the mapreduce_shuffle.

3. Format the NameNode:

```
bin/hdfs namenode –format
```

4. Start the daemons using the command:

```
sbin/start-yarn.sh
```

This starts the daemons ResourceManager and NodeManager.

Once this command is run, you can check if ResourceManager is running or not by visiting the following URL on browser : http://0.0.0.0:8088 . If you are unable to see this, check for the logs in the directory: /home/hadoop_dev/hadoop2/logs

5. To check whether the services are running, issue a jps command. The following shows all the services necessary to run YARN on a single server:

```
$ jps
15933 Jps
15567 ResourceManager
15785 NodeManager
```

6. Let us run the same example as we ran before:

i) Create a new directory on the hdfs

```
bin/hdfs dfs -mkdir –p /user/hadoop_dev
```

ii) Copy the input files for the program to hdfs:

```
bin/hdfs dfs -put etc/hadoop input
```

iii) Run the program:

```
bin/yarn jar share/hadoop/mapreduce/
hadoop-mapreduce-examples-2.6.0.jar grep
input output 'dfs[a-z.]+'
```

iv) View the output on hdfs:

```
bin/hdfs dfs -cat output/*
```

7. Stop the daemons when you are done executing the jobs, with the below command:

```
sbin/stop-yarn.sh
```

This completes the installation part of Hadoop.

Experiment – 3

Aim: Implement the following file management tasks in Hadoop:

- Adding files and directories
- Retrieving files
- Deleting files

Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.

File Management tasks in Hadoop

1. Create a directory in HDFS at given path(s).

Usage:

hadoop fs -mkdir <paths>

Example:

hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2

2. List the contents of a directory.

Usage :

hadoop fs -ls <args>

Example:

hadoop fs -ls /user/saurzcode

3. Upload and download a file in HDFS.

*Upload***:**

hadoop fs -put:

Copy single src file, or multiple src files from local file system to the Hadoop data file system

Usage:

hadoop fs -put <localsrc> ... <HDFS_dest_Path>

Example:

hadoop fs -put /home/saurzcode/Samplefile.txt /user/ saurzcode/dir3/

Download:

hadoop fs -get:

Copies/Downloads files to the local file system

Usage:

hadoop fs -get <hdfs_src> <localdst>

Example:

hadoop fs -get /user/saurzcode/dir3/Samplefile.txt /home/

4. See contents of a file Same as unix cat command:

Usage:

hadoop fs -cat <path[filename]>

Example:

hadoop fs -cat /user/saurzcode/dir1/abc.txt

5. Copy a file from source to destination

This command allows multiple sources as well in which case the destination must be a directory.

Usage:

hadoop fs -cp <source> <dest>

Example:

hadoop fs -cp /user/saurzcode/dir1/abc.txt /user/saurzcode/ dir2

6. Copy a file from/To Local file system to HDFS

copyFromLocal

Usage:

hadoop fs -copyFromLocal <localsrc> URI

Example:

hadoop fs -copyFromLocal /home/saurzcode/abc.txt /user/ saurzcode/abc.txt

Similar to put command, except that the source is restricted to a local file reference.

copyToLocal

Usage:

hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

Similar to get command, except that the destination is restricted to a local file reference.

7. Move file from source to destination.

Note:- Moving files across filesystem is not permitted.

Usage :

hadoop fs -mv <src> <dest>

Example:

hadoop fs -mv /user/saurzcode/dir1/abc.txt /user/saurzcode/ dir2

8. Remove a file or directory in HDFS.

Remove files specified as argument. Deletes directory only when it is empty

Usage :

hadoop fs -rm <arg>

Example:

hadoop fs -rm /user/saurzcode/dir1/abc.txt

Recursive version of delete.

Usage :

hadoop fs -rmr <arg>

Example:

hadoop fs -rmr /user/saurzcode/

9. Display last few lines of a file. Similar to tail command in Unix.

Usage :

hadoop fs -tail <path[filename]>

Example:

hadoop fs -tail /user/saurzcode/dir1/abc.txt

10. Display the aggregate length of a file.

Usage :

hadoop fs -du <path>

Example:

hadoop fs -du /user/saurzcode/dir1/abc.tx

Experiment – 4

Aim: Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

Word Count Map Reduce program to understand Map Reduce Paradigm

Source code:

```
import java.io.IOException;
import java.util.StringTokenizer;
import
org.apache.hadoop.io.IntWritable;
import
org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import
org.apache.hadoop.mapreduce.Mapper; import
org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import
org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;
public class WordCount
{
public static class Map extends
Mapper<LongWritable,Text,Text,IntWritable> { public void
```

```
map(LongWritable key, Text value,Context context) throws
IOException,InterruptedException{
String line = value.toString();
StringTokenizer tokenizer = new
StringTokenizer(line); while
(tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new
IntWritable(1)); }
}
}
public static class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> { public void reduce(Text key,
Iterable<IntWritable> values,Context context) throws
IOException,InterruptedException {
int sum=0;
(IntWritable x: values)
{
sum+=x.get()
; }
context.write(key, new
IntWritable(sum)); }
}
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.cla
ss); 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);
}
}
```

The entire MapReduce program can be fundamentally divided into three parts:

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

We will understand the code for each of these three parts sequentially.

Mapper code:

```
public static class Map extends
Mapper<LongWritable,Text,Text,IntWritable>
{
public void map(LongWritable key, Text value, Context context)
throws IOException,InterruptedException {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new
IntWritable(1)); }
```

We have created a class Map that extends the class Mapper which is already defined in the MapReduce Framework.We define the data types of input and output key/value pair after the class declaration using angle brackets. Both the input and output of the Mapper is a key/value pair.

Input: The key is nothing but the offset of each line in the text file:LongWritable
The value is each individual line (as shown in the figure at the right): Text

Output: The key isthe tokenized words: Text. We have the hardcoded value in our case which is 1: IntWritable.Example – Dear 1, Bear 1, etc.We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to 1.

Reducer Code:

```
public static class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable>
{
public void reduce(Text key, Iterable<IntWritable>
values,Context context)
throws IOException,InterruptedException {
int sum=0;
for(IntWritable x:
values) {
sum+=x.get();
}
context.write(key, new
IntWritable(sum)); }
}
```

We have created a class Reduce which extends class Reducer like that of Mapper.
We define the data types of input and output key/value pair after the class declaration using angle brackets as done for Mapper.
Both the input and the output of the Reducer is a key- value pair.

Input:

The key nothing but those unique words which have been generated after the sorting and shuffling phase: Text
The value is a list of integers corresponding to each key: IntWritable

Example – Bear, [1, 1], etc.

Output:

The key is all the unique words present in the input text file: Text

The value is the number of occurrences of each of the unique words: IntWritable

Example – Bear, 2; Car, 3, etc.

We have aggregated the values present in each of the list corresponding to each key and produced the final answer.

In general, a single reducer is created for each of the unique words, but, you can specify the number of reducer in mapred-site.xml.

Driver Code:

```
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]));
```

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 that how a Mapper will read the input data or what will be the unit of work. Here, we have chosen

TextInputFormat so that 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.

Run the MapReduce code:

The command for running a MapReduce code is:

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

Experiment – 5

Aim: Write a Map Reduce program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi structured and record-oriented.

Weather Report POC-Map Reduce Program to analyse time-temperature statistics and generate report with max/min temperature.

Problem Statement:

1. The system receives temperatures of various cities (Austin, Boston, etc) of USA captured at regular intervals of time on each day in an input file.
2. System will process the input data file and generates a report with Maximum and Minimum temperatures of each day along with time.
3. Generates a separate output report for each city.

Ex: Austin-r-00000
Boston-r-00000
Newjersy-r-00000
Baltimore-r-00000
California-r-00000
Newyork-r-00000

Expected output:- In each output file record should be like this:

25-Jan-2014 Time: 12:34:542 MinTemp: -22.3 Time: 05:12:345 MaxTemp: 35.7

First download input file which contains temperature statistics with time for multiple cities.Schema of record set : CA_25-Jan-2014 00:12:345 15.7 01:19:345 23.1 02:34:542 12.3 .....

CA is city code, here it stands for California followed by date. After that each pair of values represent time and temperature.

Mapper class and map method:-

The very first thing which is required for any map reduce problem is to understand what will be the type of keyIn, ValueIn, KeyOut,ValueOut for the given Mapper class and followed by type of map method parameters.

- public class WhetherForcastMapper extends Mapper <**Object, Text, Text, Text**>
- Object (keyIn) - Offset for each line, line number 1, 2...

- Text **(**ValueIn**) -** Whole string for each line (CA_25-Jan-2014 00:12:345 .....)
- Text **(**KeyOut**) -** City information with date information as string
- Text **(**ValueOut**) -** Temperature and time information which need to be passed to reducer as string.
- public void map(**Object keyOffset, Text dayReport, Context con**) { } · *KeyOffset* is like line number for each line in input file.
- *dayreport* is input to map method - whole string present in one line of input file.
- *con* is context where we write mapper output and it is used by reducer. **Reducer class and Reducer method**:-

Similarly,we have to decide what will be the type of keyIn, ValueIn, KeyOut,ValueOut for the given Reducer class and followed by type of reducer method parameters.

- public class WhetherForcastReducer extends Reducer<**Text, Text, Text, Text**>
- Text(keyIn) - it is same as keyOut of Mapper.
- Text(ValueIn)- it is same as valueOut of Mapper. ·
  Text(KeyOut)- date as string
- text(ValueOut) - reducer writes max and min temperature with time as string
- public void reduce(**Text key, Iterable<Text> values, Context context**)
- Text key is value of mapper output. i.e:- City & date information
- Iterable<Text> values - values stores multiple temperature values for a given city and date.
- context object is where reducer write it's processed outcome and finally written in file.

Multiple Outputs :- In general, reducer generates output file(i.e: part_r_0000), however in this use case we want to generate multiple output files. In order to deal with such scenario we need to use MultipleOutputs of "org.apache.hadoop.mapreduce.lib.output.MultipleOutputs" which provides a way to write multiple file depending on reducer outcome. See below reducer class for more details.For each reducer task multipleoutput object is created and key/result is written to appropriate file.

Lets create a Map/Reduce project in eclipse and create a class file name it as Calculate Max And Min Temerature With Time. For simplicity, here we have written mapper and reducer class as inner static class. Copy following code lines and paste in newly created class file.

/**

```
* Question:- To find Max and Min temperature from
record set stored in
* text file. Schema of record set :- tab
separated (\t) CA_25-Jan-2014
*              00:12:345 15.7 01:19:345 23.1 02:34:542
12.3 03:12:187 16 04:00:093
*            -14 05:12:345 35.7 06:19:345 23.1 07:34:542
12.3 08:12:187 16
*          09:00:093 -7 10:12:345 15.7 11:19:345 23.1
12:34:542 -22.3 13:12:187
* 16 14:00:093 -7 15:12:345 15.7 16:19:345
23.1 19:34:542 12.3
* 20:12:187 16 22:00:093 -7
* Expected output:- Creates files for each city and
store maximum & minimum
* temperature for each day along
with time.
*/
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.mapreduce.lib.output.MultipleOutput
s;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat
;
```

```
import
org.apache.hadoop.mapreduce.lib.output.FileOutputForm
at;
import
org.apache.hadoop.mapreduce.lib.output.TextOutputForm
at;

/**
* @author devinline
*/
public class CalculateMaxAndMinTemeratureWithTime {
public static String calOutputName =
"California";
public static String nyOutputName = "Newyork";
public static String njOutputName = "Newjersy";
public static String ausOutputName = "Austin";
public static String bosOutputName = "Boston";
public static String balOutputName =
"Baltimore";
public static class WhetherForcastMapper extends
Mapper<Object, Text, Text, Text> {
public void map(Object keyOffset, Text
dayReport, Context con)
throws IOException, InterruptedException {
StringTokenizer strTokens = new
StringTokenizer(
dayReport.toString(), "\t");
int counter = 0;
Float currnetTemp = null;
Float minTemp = Float.MAX_VALUE;
Float maxTemp = Float.MIN_VALUE;
String date = null;
String currentTime = null;
```

```
String minTempANDTime = null;
String maxTempANDTime = null;
while (strTokens.hasMoreElements()) {
if (counter == 0) {
date = strTokens.nextToken();
} else {
if (counter % 2 == 1) {
currentTime = strTokens.nextToken();
} else {
currnetTemp =
Float.parseFloat(strTokens.nextToken());
if (minTemp > currnetTemp) {
minTemp = currnetTemp;
minTempANDTime = minTemp + "AND" +
currentTime;
}
if (maxTemp < currnetTemp) {
maxTemp = currnetTemp;
maxTempANDTime = maxTemp + "AND" +
currentTime;
}
}
}
counter++;
}
// Write to context - MinTemp, MaxTemp and
corresponding time
Text temp = new Text();
temp.set(maxTempANDTime);
Text dateText = new Text();
dateText.set(date);
try {
con.write(dateText, temp);
```

```
} catch (Exception e) {
e.printStackTrace();
}
temp.set(minTempANDTime);
dateText.set(date);
con.write(dateText, temp);
}
}
public static class WhetherForcastReducer extends
Reducer<Text, Text, Text, Text> {
MultipleOutputs<Text, Text> mos;
public void setup(Context context) {
mos = new MultipleOutputs<Text,
Text>(context);
}
public void reduce(Text key, Iterable<Text>
values, Context context)
throws IOException, InterruptedException {
int counter = 0;
String reducerInputStr[] = null;
String f1Time = "";
String f2Time = "";
String f1 = "", f2 = "";
Text result = new Text();
for (Text value : values) {
if (counter == 0) {
reducerInputStr =
value.toString().split("AND");
f1 = reducerInputStr[0];
f1Time = reducerInputStr[1];
}
else {
reducerInputStr =
```

```
value.toString().split("AND");
f2 = reducerInputStr[0];
f2Time = reducerInputStr[1];
}
counter = counter + 1;
}
if (Float.parseFloat(f1) >
Float.parseFloat(f2)) {
result = new Text("Time: " + f2Time + "
MinTemp: " + f2 + "\t"
+ "Time: " + f1Time + " MaxTemp: " + f1);
} else {
result = new Text("Time: " + f1Time + "
MinTemp: " + f1 + "\t"
+ "Time: " + f2Time + " MaxTemp: " + f2);
}
String fileName = "";
if (key.toString().substring(0, 2).equals("CA"))
{
fileName =
CalculateMaxAndMinTemeratureTime.calOutputName;
} else if (key.toString().substring(0,
2).equals("NY")) {
fileName =
CalculateMaxAndMinTemeratureTime.nyOutputName;
} else if (key.toString().substring(0,
2).equals("NJ")) {
fileName =
CalculateMaxAndMinTemeratureTime.njOutputName;
} else if (key.toString().substring(0,
3).equals("AUS")) {
fileName =
CalculateMaxAndMinTemeratureTime.ausOutputName;
```

```
} else if (key.toString().substring(0,
3).equals("BOS")) {
fileName =
CalculateMaxAndMinTemeratureTime.bosOutputName;
} else if (key.toString().substring(0,
3).equals("BAL")) {
fileName =
CalculateMaxAndMinTemeratureTime.balOutputName;
}
String strArr[] = key.toString().split("_");
key.set(strArr[1]); //Key is date value
mos.write(fileName, key, result);
}
@Override
public void cleanup(Context context) throws
IOException,
InterruptedException {
mos.close();
}
}
public static void main(String[] args) throws
IOException,
ClassNotFoundException,
InterruptedException {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "Wheather
Statistics of USA");
job.setJarByClass(CalculateMaxAndMinTemeratureW
ithTime.class);
job.setMapperClass(WhetherForcastMapper.class);
job.setReducerClass(WhetherForcastReducer.class)
;
job.setMapOutputKeyClass(Text.class);
```

```
job.setMapOutputValueClass(Text.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
MultipleOutputs.addNamedOutput(job,
calOutputName,
TextOutputFormat.class, Text.class,
Text.class);
MultipleOutputs.addNamedOutput(job,
nyOutputName,
TextOutputFormat.class, Text.class,
Text.class);
MultipleOutputs.addNamedOutput(job,
njOutputName,
TextOutputFormat.class, Text.class,
Text.class);
MultipleOutputs.addNamedOutput(job,
bosOutputName,
TextOutputFormat.class, Text.class,
Text.class);
MultipleOutputs.addNamedOutput(job,
ausOutputName,
TextOutputFormat.class, Text.class,
Text.class);
MultipleOutputs.addNamedOutput(job,
balOutputName,
TextOutputFormat.class, Text.class,
Text.class);
// FileInputFormat.addInputPath(job, new
Path(args[0]));
// FileOutputFormat.setOutputPath(job, new
Path(args[1]));
Path pathInput = new Path(
"hdfs://192.168.213.133:54310/weatherInputData/
```

```
input_temp.txt");
Path pathOutputDir = new Path(
"hdfs://192.168.213.133:54310/user/hduser1/
testfs/output_mapred3");
FileInputFormat.addInputPath(job, pathInput);
FileOutputFormat.setOutputPath(job,
pathOutputDir);
try {
System.exit(job.waitForCompletion(true) ? 0 :
1);
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
} }
```

Now execute above sample program. Run -> Run as hadoop. Wait for a moment and check whether output directory is in place on HDFS. Execute following command to verify the same.

```
hduser1@ubuntu:/usr/local/hadoop2.6.1/bin$ ./hadoop
fs -ls /user/hduser1/testfs/output_mapred3
Found 8 items
-rw-r--r-- 3 zytham supergroup 438
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
Austin-r-00000
-rw-r--r-- 3 zytham supergroup 219
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
Baltimore-r-00000
-rw-r--r-- 3 zytham supergroup 219
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
Boston-r-00000
-rw-r--r-- 3 zytham supergroup 511
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
```

California-r-00000

-rw-r--r-- 3 zytham supergroup 146

2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/

Newjersy-r-00000

-rw-r--r-- 3 zytham supergroup 219

2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/

Newyork-r-00000

-rw-r--r-- 3 zytham supergroup 0

2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/

_SUCCESS

-rw-r--r-- 3 zytham supergroup 0

2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/

part-r-00000

Open one of the file and verify expected output schema, execute following command for the same.

hduser1@ubuntu:/usr/local/hadoop2.6.1/bin$ ./hadoop fs -cat /user/hduser1/testfs/output_mapred3/

Austin-r-00000

25-Jan-2014 Time: 12:34:542 MinTemp: -22.3 Time: 05:12:345 MaxTemp: 35.7

26-Jan-2014 Time: 22:00:093 MinTemp: -27.0 Time: 05:12:345 MaxTemp: 55.7

27-Jan-2014 Time: 02:34:542 MinTemp: -22.3 Time: 05:12:345 MaxTemp: 55.7

29-Jan-2014 Time: 14:00:093 MinTemp: -17.0 Time: 02:34:542 MaxTemp: 62.9

30-Jan-2014 Time: 22:00:093 MinTemp: -27.0 Time: 05:12:345 MaxTemp: 49.2

31-Jan-2014 Time: 14:00:093 MinTemp: -17.0 Time: 03:12:187 MaxTemp: 56.0

Experiment – 6

Aim: Implement Matrix Multiplication with Hadoop Map Reduce.

Implementing Matrix Multiplication with Hadoop Map Reduce

```
import java.io.IOException; import
java.util.*;
import java.util.AbstractMap.SimpleEntry; import
java.util.Map.Entry;
import org.apache.hadoop.fs.Path; import
org.apache.hadoop.conf.*; import
org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.input.TextInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; import
org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class TwoStepMatrixMultiplication {
        public static class Map extends Mapper<LongWritable, Text, Text, Text> {
                public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
                        String line = value.toString();
                        String[] indicesAndValue = line.split(","); Text outputKey = new
                        Text();
                        Text outputValue = new Text();
                        if (indicesAndValue[0].equals("A")) { outputKey.set(indicesAndValue[2]);
                                outputValue.set("A," + indicesAndValue[1] + "," +
indicesAndValue[3]);
                                context.write(outputKey, outputValue); } else {
                                outputKey.set(indicesAndValue[1]); outputValue.set("B," +
                                indicesAndValue[2] + "," +
indicesAndValue[3]);
                                context.write(outputKey, outputValue); }
```

```
        } }
    public static class Reduce extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
            String[] value;
            ArrayList<Entry<Integer, Float>> listA = new
ArrayList<Entry<Integer, Float>>();
            ArrayList<Entry<Integer, Float>> listB = new
ArrayList<Entry<Integer, Float>>();
            for (Text val : values) {
                value = val.toString().split(","); if
                (value[0].equals("A")) {
                    listA.add(new SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));
                } else {
                    listB.add(new SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));
                } }
            String i; float
            a_ij; String k;
            float b_jk;
            Text outputValue = new Text();
            for (Entry<Integer, Float> a : listA) { i =
                Integer.toString(a.getKey()); a_ij = a.getValue();
                for (Entry<Integer, Float> b : listB) { k =
                    Integer.toString(b.getKey()); b_jk = b.getValue();
                    outputValue.set(i + "," + k + "," +
Float.toString(a_ij*b_jk));
                    context.write(null, outputValue); }
            } }
    }
    public static void main(String[] args) throws Exception { Configuration conf = new
        Configuration();
```

```
        Job job = new Job(conf,
"MatrixMatrixMultiplicationTwoSteps");
        job.setJarByClass(TwoStepMatrixMultiplication.class);
        job.setOutputKeyClass(Text.class); job.setOutputValueClass(Text.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path("hdfs://
127.0.0.1:9000/matrixin"));
          FileOutputFormat.setOutputPath(job, new Path("hdfs://
127.0.0.1:9000/matrixout"));

        job.waitForCompletion(true);
    }
}
```

Experiment – 7

Aim: Install and Run Pig then write Pig Latin scripts to sort, group, join, project, and filter your data.

Pig Latin scripts to sort, group, join, project, and filter your data.

ORDER BY

Sorts a relation based on one or more fields.

Syntax

alias = ORDER alias BY { * [ASC|DESC] | field_alias [ASC|DESC] [, field_alias [ASC|DESC] …] } [PARALLEL n];

Terms

alias	The name of a relation.
*	The designator for a tuple.
field_alias	A field in the relation. The field must be a simple type.
ASC	Sort in ascending order.
DESC	Sort in descending order.
PARALLEL n	Increase the parallelism of a job by specifying the number of reduce tasks, n. For more information, see Use the Parallel Features.

Usage

Note: ORDER BY is NOT stable; if multiple records have the same ORDER BY key, the order in which these records are returned is not defined and is not guaranteed to be the same from one run to the next.

In Pig, relations are unordered (see Relations, Bags, Tuples, Fields):

- If you order relation A to produce relation X (X = ORDER A BY * DESC;) relations A and X still contain the same data.
- If you retrieve relation X (DUMP X;) the data is guaranteed to be in the order you specified (descending).
- However, if you further process relation X (Y = FILTER X BY $0 > 1;) there is no guarantee that the data will be processed in the order you originally specified (descending).

Pig currently supports ordering on fields with simple types or by tuple designator (*). You cannot order on fields with complex types or by expressions.

```
A = LOAD 'mydata' AS (x: int, y: map[]);
B = ORDER A BY x; -- this is allowed because x is a simple
```

type
B = ORDER A BY y; -- this is not allowed because y is a
complex type
B = ORDER A BY y#'id'; -- this is not allowed because y#'id'
is an expression

Examples

Suppose we have relation A.

A = LOAD 'data' AS (a1:int,a2:int,a3:int);

DUMP A;

(1,2,3)

(4,2,1)

(8,3,4)

(4,3,3)

(7,2,5)

(8,4,3)

In this example relation A is sorted by the third field, f3 in descending order. Note that the order of the three tuples ending in 3 can vary.

X = ORDER A BY a3 DESC;

DUMP X; (7,2,5) (8,3,4) (1,2,3) (4,3,3) (8,4,3) (4,2,1)

Returns each tuple with the rank within a relation.

Syntax

alias = RANK alias [BY { * [ASC|DESC] | field_alias [ASC|DESC] [, field_alias [ASC|DESC] …] } [DENSE]];

Terms

alias	The name of a relation.
*	The designator for a tuple.
field_alias	A field in the relation. The field must be a simple type.
ASC	Sort in ascending order.
DESC	Sort in descending order.
DENSE	No gap in the ranking values.

Usage

When specifying no field to sort on, the RANK operator simply prepends a sequential value to each tuple.

Otherwise, the RANK operator uses each field (or set of fields) to sort the relation. The rank of a tuple is one plus the number of different rank values preceding it. If two or more tuples tie on the sorting field values, they will receive the same rank.

NOTE: When using the option **DENSE**, ties do not cause gaps in ranking values.

Examples

Suppose we have relation A.

A = load 'data' AS (f1:char array, f2:int, f3:char array);

```
DUMP A;
(David,1,N)
(Tete,2,N)
(Ranjit,3,M)
(Ranjit,3,P)
(David,4,Q)
(David,4,Q)
(Jillian,8,Q)
(JaePak,7,Q)
(Michael,8,T)
(Jillian,8,Q)
(Jose,10,V)
```

In this example, the RANK operator does not change the order of the relation and simply prepends to each tuple a sequential value.

```
B = rank A;
dump B;
(1,David,1,N)
(2,Tete,2,N)
(3,Ranjit,3,M)
(4,Ranjit,3,P)
```

(5,David,4,Q)
(6,David,4,Q)
(7,Jillian,8,Q)
(8,JaePak,7,Q)
(9,Michael,8,T)
(10,Jillian,8,Q)
(11,Jose,10,V)

In this example, the RANK operator works with f1 and f2 fields, and each one with different sorting order. RANK sorts the relation on these fields and prepends the rank value to each tuple. Otherwise, the RANK operator uses each field (or set of fields) to sort the relation. The rank of a tuple is one plus the number of different rank values preceding it. If two or more tuples tie on the sorting field values, they will receive the same rank.

C = rank A by f1 DESC, f2 ASC;

dump C;
(1,Tete,2,N)
(2,Ranjit,3,M)
(2,Ranjit,3,P)
(4,Michael,8,T)
(5,Jose,10,V)
(6,Jillian,8,Q)
(6,Jillian,8,Q)
(8,JaePak,7,Q)
(9,David,1,N)
(10,David,4,Q)
(10,David,4,Q)

Same example as previous, but DENSE. In this case there are no gaps in ranking values.

C = rank A by f1 DESC, f2 ASC DENSE;

dump C;

(1,Tete,2,N)

(2,Ranjit,3,M)

(2,Ranjit,3,P)

(3,Michael,8,T)

(4,Jose,10,V)

(5,Jillian,8,Q)

(5,Jillian,8,Q)

(6,JaePak,7,Q)

(7,David,1,N)

(8,David,4,Q)

(8,David,4,Q)

Experiment – 8

Aim: Install and Run Hive then use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

Hive Databases, Tables, Views, Functions and Indexes

Databases in Hive

The Hive concept of a database is essentially just a *catalog* or *namespace* of tables. However, they are very useful for larger clusters with multiple teams and users, as a way of avoiding table name collisions. It's also common to use databases to organize production tables into logical groups. If you don't specify a database, the default database is used.

The simplest syntax for creating a database is shown in the following example:

hive> **CREATE DATABASE** financials;

Hive will throw an error if financials already exists. You can suppress these warnings with his variation:

hive> **CREATE DATABASE** IF **NOT EXISTS** financials;

While normally you might like to be warned if a database of the same name already exists, the IF NOT EXISTS clause is useful for scripts that should create a database on-the-fly, if necessary, before proceeding.

You can also use the keyword SCHEMA instead of DATABASE in all the database-related commands.

At any time, you can see the databases that already exist as follows:

hive> **SHOW** DATABASES;
default
financials
hive> **CREATE DATABASE** human_resources;
hive> **SHOW** DATABASES;
default
financials
human_resources

If you have a lot of databases, you can restrict the ones listed using a *regular expression*, a concept we'll explain in LIKE and RLIKE, if it is new to you. The following example lists only those databases that start with the letter h and end with any other characters (the .* part):

```
hive> SHOW DATABASES LIKE 'h.*';
human_resources
hive> ...
```

Hive will create a directory for each database. Tables in that database will be stored in subdirectories of the database rectory. The exception is tables in the default database, which doesn't have its own directory.

The database directory is created under a top-level directory specified by the property hive.metastore.warehouse.dir, which we discussed in Local Mode Configuration and Distributed and Pseudo distributed Mode Configuration. Assuming you are using the default value for this property,*/user/hive/warehouse*, when the financials database is created, Hive will create the directory */user/hive/warehouse/financials.db*. Note

the .*db*extension.

You can override this default location for the new directory as shown in this example:

```
hive> CREATE DATABASE financials
    > LOCATION '/my/preferred/directory';
```

You can add a descriptive comment to the database, which will be shown by the DESCRIBE DATABASE <database> command.

```
hive> CREATE DATABASE financials
    > COMMENT 'Holds all financial tables';

hive> DESCRIBE DATABASE financials;
financials      Holds all financial tables
  hdfs://master-server/user/hive/warehouse/financials.db
```

Note that DESCRIBE DATABASE also shows the directory location for the database. In this example, the *URI scheme* is hdfs. For a MapR installation, it would be maprfs. For an Amazon Elastic MapReduce (EMR) cluster, it would also be hdfs, but you could set hive.metastore.warehouse.dir to use Amazon S3 explicitly (i.e., by specifying s3n://bucketname/… as the property value). You could use s3 as the scheme, but the newer s3n is preferred.

In the output of DESCRIBE DATABASE, we're showing master-server to indicate the URI *authority*, in this case a DNS name and optional port number (i.e., server:port) for the "master node" of the file system (i.e., where the *NameNode* service is running for HDFS). If you are running in

pseudo-distributed mode, then the master server will be localhost. For *local* mode, the path will be a local path, *file:///user/hive/warehouse/financials.db*.

If the authority is omitted, Hive uses the master-server name and port defined by the property fs.default.name in the Hadoop configuration files, found in the *$HADOOP_HOME/ conf* directory.

To be clear, *hdfs:///user/hive/warehouse/financials.db* is equivalent to*hdfs://master-server/user/hive/warehouse/ financials.db*, where master-server is your master node's DNS name and optional port.

For completeness, when you specify a *relative* path (e.g., *some/ relative/path*), Hive will put this under your home directory in the distributed filesystem (e.g.,*hdfs:///user/<user-name>*) for HDFS. However, if you are running in *local mode*, your current working directory is used as the parent of*some/relative/path*.

For script portability, it's typical to omit the authority, only specifying it when referring to another distributed filesystem instance (including S3 buckets).

Lastly, you can associate key-value properties with the database, although their only function currently is to provide a way of adding information to the output of DESCRIBE DATABASE EXTENDED <database>:

```
hive> CREATE DATABASE financials
    > WITH DBPROPERTIES ('creator' = 'Mark Moneybags', 'date'
= '2012-01-02');

hive> DESCRIBE DATABASE financials;
financials      hdfs://master-server/user/hive/warehouse/
financials.db

hive> DESCRIBE DATABASE EXTENDED financials;
financials      hdfs://master-server/user/hive/warehouse/
financials.db
  {date=2012-01-02, creator=Mark Moneybags);
```

The USE command sets a database as your working database, analogous to changing working directories in a filesystem:

```
hive> USE financials;
```

Now, commands such as SHOW TABLES; will list the tables in this database.

Unfortunately, there is no command to show you which database is your current working database! Fortunately, it's always safe to repeat the USE … command; there is no concept in Hive of nesting of databases.

Recall that we pointed out a useful trick in Variables and Properties for setting a property to print the current database as part of the prompt (Hive v0.8.0 and later):

hive> **set** hive.cli.print.**current**.db=**true**;

hive (financials)> USE **default**;

hive (**default**)> **set** hive.cli.print.**current**.db=**false**;

hive> ...

Finally, you can drop a database:

hive> **DROP DATABASE** IF **EXISTS** financials;

The IF EXISTS is optional and suppresses warnings

if financials doesn't exist.

By default, Hive won't permit you to drop a database if it contains tables. You can either drop the tables first or append

the CASCADE keyword to the command, which will cause the Hive to drop the tables in the database first:

hive> **DROP DATABASE** IF **EXISTS** financials **CASCADE**;

Using the RESTRICT keyword instead of CASCADE is equivalent to the default behavior, where existing tables must be dropped before dropping the database. When a database is dropped, its directory is also deleted.

Alter Database

You can set key-value pairs in the DBPROPERTIES associated with a database using the ALTER DATABASE command. No other metadata about the database can be changed, including its name and directory location:

hive> **ALTER DATABASE** financials **SET** DBPROPERTIES ('edited-by'
= 'Joe Dba');

There is no way to delete or "unset" a DBPROPERTY.

Creating Tables

The CREATE TABLE statement follows SQL conventions, but Hive's version offers significant extensions to support a wide range of flexibility where the data files for tables are stored, the formats used, etc. We discussed many of these options in Text File Encoding of Data Values.

In this section, we describe the other options available for the CREATE TABLE statement, adapting the employees table declaration we used previously in Collection Data Types:

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

First, note that you can prefix a database name, mydb in this case, if you're not currently working in the target database.

If you add the option IF NOT EXISTS, Hive will silently ignore the statement if the table already exists. This is useful in scripts that should create a table the first time they run.

However, the clause has a gotcha you should know. If the schema specified differs from the schema in the table that already exists, Hive won't warn you. If your intention is for this table to have the new schema, you'll have to drop the old table, losing your data, and then re-create it. Consider if you should use one or more ALTER TABLE statements to change the existing table schema instead.

See Alter Table for details.

You can add a comment to any column, after the type. Like databases, you can attach a comment to the table itself and you can define one or more table *properties*. In most cases, the primary benefit of TBLPROPERTIES is to add additional documentation in a key-value format. However, when we examine Hive's integration with databases such as DynamoDB (see DynamoDB), we'll see that the TBLPROPERTIES can be used to express essential metadata about the database connection.

Hive automatically adds two table properties: last_modified_by holds the username of the last user to modify the table, and last_modified_timeholds the epoch time in seconds of that modification.

Finally, you can optionally specify a location for the table data (as opposed to *metadata*, which the *metastore* will always hold). In this example, we are showing the default location that Hive would use, */ user/hive/warehouse/mydb.db/employees*, where */user/hive/ warehouse* is the default "warehouse" location (as discussed previously), *mydb.db* is the database directory, and *employees* is the table directory.

By default, Hive always creates the table's directory under the directory for the enclosing database. The exception is the *default* database. It doesn't have a directory under */user/hive/ warehouse*, so a table in the *default* database will have its directory created directly in */user/hive/warehouse* (unless explicitly overridden).

You can also copy the schema (but not the data) of an existing table:

```
CREATE TABLE IF NOT EXISTS mydb.employees2
LIKE mydb.employees;
```

This version also accepts the optional LOCATION clause, but note that no other properties, including the schema, can be defined; they are determined from the original table.
The SHOW TABLES command lists the tables. With no additional arguments, it shows the tables in the current working database. Let's assume we have already created a few other tables, table1 and table2, and we did so in the mydbdatabase:

```
hive> USE mydb;
hive> SHOW TABLES;
employees
```

table1

table2

If we aren't in the same database, we can still list the tables in that database:

hive> USE **default**;

hive> **SHOW** TABLES **IN** mydb;

employees

table1

table2

If we have a lot of tables, we can limit the ones listed using a *regular expression*, a concept we'll discuss in detail in LIKE and RLIKE:

hive> USE mydb;

hive> **SHOW** TABLES 'empl.*';

employees

Not all regular expression features are supported. If you know regular expressions, it's better to test a candidate regular expression to make sure it actually works!

The regular expression in the single quote looks for all tables with names starting with empl and ending with any other characters (the .* part).

We can also use the DESCRIBE EXTENDED mydb.employees command to show details about the table. (We can drop the mydb. prefix if we're currently using the mydb database.) We have reformatted the output for easier reading and we have suppressed many details to focus on the items that interest us now:

hive> **DESCRIBE** EXTENDED mydb.employees;

name string Employee name

salary float Employee salary

subordinates array<string> **Names of** subordinates

deductions **map**<string,float> Keys **are** deductions **names**, **values are** percentages

addess struct<street:string,city:string,**state**:string,zip:int>
Home address

Detailed **Table** Information **Table**(tableName:employees,
dbName:mydb, **owner**:me,
...
location:hdfs://master-server/**user**/hive/warehouse/mydb.db/
employees,
parameters:{creator=me, created_at='2012-01-02 10:00:00',
last_modified_user=me,
last_modified_time=1337544510,
comment:Description **of** the **table**, ...}, ...)

Replacing EXTENDED with FORMATTED provides more readable but also more verbose output.

The first section shows the output of DESCRIBE without EXTENDED or FORMATTED (i.e., the schema including the comments for each column).

If you only want to see the schema for a particular column, append the column to the table name. Here, EXTENDED adds no additional output:

hive> **DESCRIBE** mydb.employees.salary;
salary float Employee salary

Returning to the extended output, note the line in the description that starts with location:. It shows the full URI path in HDFS to the directory where Hive will keep all the data for this table, as we discussed above.

The tables we have created so far are called *managed* tables or sometimes called *internal* tables, because Hive controls the lifecycle of their data (more or less). As we've seen, Hive stores the data for these tables in a subdirectory under the directory defined by hive.metastore.warehouse.dir (e.g.,*/user/hive/ warehouse*), by default.

When we drop a managed table (see Dropping Tables), Hive deletes the data in the table.

However, managed tables are less convenient for sharing with other tools. For example, suppose we have data that is created and used primarily by *Pig* or other tools, but we want to run some queries against it, but not give Hive *ownership* of the data. We can define an *external* table that points to that data, but doesn't take ownership of it.

External Tables

Suppose we are analyzing data from the stock markets. Periodically, we ingest the data for NASDAQ and the NYSE from a source like Infochimps (http://infochimps.com/datasets) and we want to study this data with many tools. (See the data sets named infochimps_dataset_4777_download_16185 andinf ochimps_dataset_4778_download_16677, respectively, which are actually sourced from Yahoo! Finance.) The schema we'll use next matches the schemas of both these data sources. Let's assume the data files are in the distributed filesystem directory */data/ stocks*.

The following table declaration creates an *external* table that can read all the data files for this comma-delimited data in */data/ stocks*:

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

The EXTERNAL keyword tells Hive this table is external and the LOCATION …clause is required to tell Hive where it's located.

Because it's external, Hive does not assume it *owns* the data. Therefore, dropping the table *does not* delete the data, although the *metadata* for the table will be deleted.

There are a few other small differences between managed and external tables, where some HiveQL constructs are not permitted for external tables. We'll discuss those when we come to them.

However, it's important to note that the differences between managed and external tables are smaller than they appear at first. Even for managed tables, you *know* where they are located, so you can use other tools, hadoop dfs commands, etc., to modify and even delete the files in the directories for managed tables. Hive may technically own these directories and files, but it doesn't have full control over them! Recall, in Schema on Read, we said that Hive really has no control over the integrity of the files used for storage and whether or not their contents are consistent with the table schema. Even managed tables don't give us this control.

Still, a general principle of good software design is to express intent. If the data is shared between tools, then creating an external table makes this ownership explicit.

You can tell whether or not a table is managed or external using the output of DESCRIBE EXTENDED tablename. Near the end of the Detailed Table Information output, you will see the following for managed tables:

... tableType:MANAGED_TABLE)

For external tables, you will see the following:

... tableType:EXTERNAL_TABLE)

As for managed tables, you can also copy the schema (but not the data) of an existing table:Partitioned, Managed Tables

The general notion of partitioning data is an old one. It can take many forms, but often it's used for distributing load horizontally,

moving data physically closer to its most frequent users, and other purposes.

Hive has the notion of partitioned tables. We'll see that they have important performance benefits, and they can help organize data in a logical fashion, such as hierarchically.

We'll discuss partitioned managed tables first. Let's return to our employee stable and imagine that we work for a very large multinational corporation. Our HR people often run

queries with WHERE clauses that restrict the results to a particular country or to a particular *first-level subdivision* (e.g., *state* in the United States or *province* in Canada). (First-level subdivision is an actual term, used here, for example: http://www.commondatahub.com/state_source.jsp.) We'll just use the word *state* for simplicity. We have redundant state information in the address field. It is distinct from the state partition. We could remove the state element from address. There is no ambiguity in queries, since we have to use address.state to project the value inside the address. So, let's partition the data first by country and then by state:

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

Partitioning tables changes how Hive structures the data storage. If we create this table in the mydb database, there will still be an *employees* directory for the table:

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

However, Hive will now create subdirectories reflecting the

partitioning structure. For example:

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

Yes, those are the actual directory names. The state directories will contain zero or more files for the employees in those states.

Once created, the partition *keys* (country and state, in this case) behave like regular columns. There is one known exception, due to a bug (see Aggregate functions). In fact, users of the table don't need to *care* if these "columns" are partitions or not, except when they want to optimize query performance.

For example, the following query selects all employees in the state of Illinois in the United States:

```
SELECT * FROM employees
WHERE country = 'US' AND state = 'IL';
```

Note that because the country and state values are encoded in directory names, there is no reason to have this data in the data files themselves. In fact, the data just gets in the way in the files, since you have to account for it in the table schema, and this data wastes space.

Perhaps the most important reason to partition data is for faster queries. In the previous query, which limits the results to employees in Illinois, it is only necessary to scan the contents of *one* directory. Even if we have thousands of country and state directories, all but one can be ignored. For very large data sets, partitioning can dramatically improve query performance, but *only* if the partitioning scheme reflects common *range* filtering (e.g., by locations, timestamp ranges).

When we add predicates to WHERE clauses that filter on partition values, these predicates are called *partition filters*.

Even if you do a query across the entire US, Hive only reads the 65 directories covering the 50 states, 9 territories, and the District of Columbia, and 6 military "states" used by the armed services. You can see the full list here: http://www.50states.com/ abbreviations.htm.
Of course, if you need to do a query for all employees around the globe, you can still do it. Hive will have to read every directory, but hopefully these broader disk scans will be relatively rare.

However, a query across all partitions could trigger an enormous MapReduce job if the table data and number of partitions are large. A highly suggested safety measure is putting Hive into "strict" mode, which prohibits queries of partitioned tables without a WHERE clause that filters on partitions. You can set the mode to "nonstrict," as in the following session:

```
hive> set hive.mapred.mode=strict;

hive> SELECT e.name, e.salary FROM employees e LIMIT 100;
FAILED: Error in semantic analysis: No partition predicate
found for
  Alias "e" Table "employees"

hive> set hive.mapred.mode=nonstrict;

hive> SELECT e.name, e.salary FROM employees e LIMIT 100;
John Doe    100000.0
...
```

You can see the partitions that exist with the SHOW PARTITIONS command:

```
hive> SHOW PARTITIONS employees;
...
          Country=CA/state=AB
           country=CA/state=BC
...
country=US/state=AL
country=US/state=AK
...
```

If you have a lot of partitions and you want to see if partitions have been defined for particular partition keys, you can further restrict the command with an optional PARTITION clause that specifies one or more of the partitions with specific values:

```
hive> SHOW PARTITIONS employees PARTITION(country='US');
          country=US/state=AL
          country=US/state=AK
...

hive> SHOW PARTITIONS employees PARTITION(country='US',
```

state='AK');
country=US/**state**=AK
The DESCRIBE EXTENDED employees command shows the partition keys:

```
hive> DESCRIBE EXTENDED employees;
name            string,
salary          float,
...
address         struct<...>,
country         string,
state           string
Detailed Table Information...
partitionKeys:[FieldSchema(name:country, type:string,
comment:null),
FieldSchema (name:state, type:string, comment:null)],
...
```

The schema part of the output lists the country and state with the other columns, because they are columns as far as queries are concerned. The Detailed Table information includes the Country and State as partition keys. The comments for both of these keys are null; we could have added comments just as for regular columns.

You create partitions in managed tables by loading data into them. The following example creates a US and CA (California) partition while loading data into it from a local directory, *$HOME/ california-employees*. You must specify a value for each partition column. Notice how we reference the HOME environment variable in HiveQL:

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

The directory for this partition, *.../employees/country=US/ state=CA*, will be created by Hive and all data files in *$HOME/ california-employees* will be copied into it. See Loading Data into Managed Tables for more information on populating tables.

External Partitioned Tables

You can use partitioning with external tables. In fact, you may find that this is your most common scenario for managing large production data sets. The combination gives you a way to “share”

data with other tools, while still optimizing query performance. You also have more flexibility in the directory structure used, as you define it yourself. We'll see a particularly useful example in a moment.

Let's consider a new example that fits this scenario well: logfile analysis. Most organizations use a standard format for log messages, recording a timestamp, severity (e.g., ERROR, WARNING, INFO), perhaps a server name and process ID, and then an arbitrary text message. Suppose our Extract, Transform, and Load (ETL) process ingests and aggregates logfiles in our environment, converting each log message to a tab-delimited record and also decomposing the timestamp into separate year, month, and day fields, and a combined hms field for the remaining hour, minute, and second parts of the timestamp, for reasons that will become clear in a moment. You could do this parsing of log messages using the string parsing functions built into Hive or Pig, for example. Alternatively, we could use smaller integer types for some of the timestamp-related fields to conserve space. Here, we are ignoring subsequent resolution.

Here's how we might define the corresponding Hive table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS log_messages (
  hms           INT,
  severity      STRING,
  server        STRING,
  process_id    INT,
  message       STRING)
PARTITIONED BY (year INT, month INT, day INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

We're assuming that a day's worth of log data is about the correct size for a useful partition and finer grain queries over a day's data will be fast enough.

Recall that when we created the nonpartitioned external stocks table, a LOCATION … clause was required. It isn't used for external partitioned tables. Instead, an ALTER TABLE statement is used to add *each* partition separately. It must specify a value for each partition key, the year, month, and day, in this case (see Alter Table for more details on this feature). Here is an example, where we add a partition for January 2nd, 2012:

```
ALTER TABLE log_messages ADD PARTITION(year = 2012, month = 1,
```

```
day = 2)
LOCATION 'hdfs://master_server/data/log_messages/2012/01/02';
```

The directory convention we use is completely up to us. Here, we follow a hierarchical directory structure, because it's a logical way to organize our data, but there is no requirement to do so.

You don't have to be an Amazon Elastic MapReduce user to use S3 this way. S3 support is part of the Apache Hadoop distribution. You can *still* query this data, even queries that cross the month-old "boundary," where some data is read from HDFS and some data is read from S3!

By the way, Hive doesn't care if a partition directory doesn't exist for a partition or if it has no files. In both cases, you'll just get no results for a query that filters for the partition. This is convenient when you want to set up partitions before a separate process starts writing data to the. As soon as data is there, queries will return results from that data.

This feature illustrates another benefit: new data can be written to a dedicated directory with a clear distinction from older data in other directories. Also, whether you move old data to an "archive" location or delete it outright, the risk of tampering with newer data is reduced since the data subsets are in separate directories.

As for nonpartitioned external tables, Hive does not own the data and it does not delete the data if the table is dropped.

As for managed partitioned tables, you can see an external table's partitions with SHOW PARTITIONS:

```
hive> SHOW PARTITIONS log_messages;
...
year=2011/month=12/day=31
        year=2012/month=1/day=1
        year=2012/month=1/day=2
...
```

Similarly, the DESCRIBE EXTENDED log_messages shows the partition keys both as part of the schema and in the list of partitionKeys:

```
hive> DESCRIBE EXTENDED log_messages;
...
message                 string,
year                    int,
month                   int,
day                     int

Detailed Table Information...
partitionKeys:[FieldSchema(name:year, type:int, comment:null),
FieldSchema(name:month, type:int, comment:null),
FieldSchema(name:day, type:int, comment:null)],
...
```

This output is missing a useful bit of information, the actual location of the partition data. There is a location field, but it only shows Hive's default directory that would be used if the table were a managed table. However, we can get a partition's location as follows:

```
hive> DESCRIBE EXTENDED log_messages PARTITION (year=2012,
month=1, day=2);
...
location:s3n://ourbucket/logs/2011/01/02,
...
```

We frequently use external partitioned tables because of the many benefits they provide, such as logical data management, performant queries, etc.

ALTER TABLE … ADD PARTITION is not limited to external tables. You can use it with managed tables, too, when you have (or will have) data for partitions in directories created outside of the LOAD and INSERT options we discussed above. You'll need to remember that not all of the table's data will be under the usual Hive "warehouse" directory, and this data *won't* be deleted when you drop the managed table! Hence, from a "sanity" perspective, it's questionable whether you should dare to use this feature with managed tables.

Customizing Table Storage Formats

In Text File Encoding of Data Values, we discussed that Hive defaults to a text file format, which is indicated by the optional clause STORED AS TEXTFILE, and you can overload the default values for the various delimiters when creating the table. Here we repeat the definition of the employees table we used in that discussion:

```
CREATE TABLE employees (
    name            STRING,
    salary          FLOAT,
    subordinates ARRAY<STRING>,
    deductions      MAP<STRING, FLOAT>,
    address         STRUCT<street:STRING, city:STRING,
state:STRING, zip:INT>
)
ROW FORMAT
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

TEXTFILE implies that all fields are encoded using alphanumeric characters, including those from international character sets, although we observed that Hive uses non-printing characters as "terminators" (delimiters), by default. When TEXTFILE is used, each line is considered a separate record.

You can replace TEXTFILE with one of the other built-in file formats supported by Hive, including SEQUENCEFILE and RCFILE, both of which optimize disk space usage and I/O bandwidth performance using binary encoding and optional compression. Hive draws a distinction between how records are encoded into files and how columns are encoded into records. You customize these behaviors separately.

The record encoding is handled by an *input format* object (e.g., the Java code behind TEXTFILE.) Hive uses a Java *class* (compiled module) namedorg.apache.hadoop.mapred.TextInputForm at. If you are unfamiliar with Java, the dotted name syntax indicates a hierarchical namespace tree

of*packages* that actually corresponds to the directory structure for the Java code. The last name, TextInputFormat, is a *class* in the lowest-level
package mapred.
The record parsing is handled by a *serializer/*

deserializer or *SerDe* for short. For TEXTFILE and the encoding we described in Chapter 3 and repeated in the example above, the SerDe Hive uses is another Java class
called org.apache.hadoop.hive.serde2.lazy.LazySimple SerDe.
For completeness, there is also an *output format* that Hive uses for writing the output of queries to files and to the console.
For TEXTFILE, the Java class named
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat is used for output.

Third-party input and output formats and SerDes can be specified, a feature which permits users to customize Hive for a wide range of file formats not supported natively.

Here is a complete example that uses a custom SerDe, input format, and output format for files accessible through the *Avro* protocol, which we will discuss in detail in Avro Hive SerDe:

```
CREATE TABLE kst
PARTITIONED BY (ds string)
ROW FORMAT SERDE 'com.linkedin.haivvreo.AvroSerDe'
WITH SERDEPROPERTIES ('schema.url'='http://schema_provider/
kst.avsc')
STORED AS
INPUTFORMAT 'com.linkedin.haivvreo.AvroContainerInputFormat'
OUTPUTFORMA
'com.linkedin.haivvreo.AvroContainerOutputFormat';
```

The ROW FORMAT SERDE … specifies the SerDe to use. Hive provides the WITH SERDEPROPERTIES feature that allows users to pass configuration information to the SerDe. Hive knows nothing about the meaning of these properties. It's up to the SerDe to decide their meaning. Note that the name and value of each property must be a quoted string.

Finally, the STORED AS INPUTFORMAT … OUTPUTFORMAT … clause specifies the Java classes to use for the input and output formats, respectively. If you specify one of these formats, you are required to specify both of them.

Note that the DESCRIBE EXTENDED table command lists the input and output formats, the SerDe, and any SerDe properties in
the DETAILED TABLE INFORMATION. For our example, we would see the following:

```
hive> DESCRIBE EXTENDED kst
...
inputFormat:com.linkedin.haivvreo.AvroContainerInputFormat,
outputFormat:com.linkedin.haivvreo.AvroContainerOutputFormat,
...
serdeInfo:SerDeInfo(name:null,
serializationLib:com.linkedin.haivvreo.AvroSerDe,
    parameters:{schema.url=http://schema_provider/kst.avsc})
...
```

Finally, there are a few additional CREATE TABLE clauses that describe more details about how the data is supposed to be stored. Let's extend our previous stocks table example from External Tables.

```
CREATE EXTERNAL TABLE IF NOT EXISTS stocks (
    exchange            STRING,
    symbol              STRING,
    ymd                 STRING,
    price_open          FLOAT,
    price_high          FLOAT,
    price_low           FLOAT,
    price_close         FLOAT,
    volume              INT,
    price_adj_close FLOAT)
CLUSTERED BY (exchange, symbol)
SORTED BY (ymd ASC)
INTO 96 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
```

LOCATION '/data/stocks';

The CLUSTERED BY … INTO … BUCKETS clause, with an optional SORTED BY … clause is used to optimize certain kinds of queries, which we discuss in detail in Bucketing Table Data Storage.

Dropping Tables: The familiar DROP TABLE command from SQL is supported.

DROP TABLE IF **EXISTS** employees;

The IF EXISTS keywords are optional. If not used and the table doesn't exist, Hive returns an error.

For *managed* tables, the table metadata *and* data are deleted.

For *external* tables, the metadata is deleted *but* the data is not.

Alter Table

Most table properties can be altered with ALTER TABLE statements, which change *metadata* about the table but not the data itself. These statements can be used to fix mistakes in schema, move partition locations (as we saw in External Partitioned Tables), and do other operations.

Use this statement to rename the table log_messages to logmsgs:

ALTER TABLE log_messages **RENAME TO** logmsgs;

Adding, Modifying, and Dropping a Table Partition

As we saw previously, ALTER TABLE table ADD PARTITION … is used to add a new partition to a table (usually an *external* table). Here we repeat the same command shown previously with the additional options available:

ALTER TABLE log_messages **ADD** IF **NOT EXISTS**
PARTITION (**year** = 2011, **month** = 1, **day** = 1) **LOCATION** '/logs/2011/01/01'
PARTITION (**year** = 2011, **month** = 1, **day** = 2) **LOCATION** '/logs/2011/01/02'
PARTITION (**year** = 2011, **month** = 1, **day** = 3) **LOCATION** '/logs/2011/01/03'
...;

Multiple partitions can be added in the same query when using Hive v0.8.0 and later. As always, IF NOT EXISTS is optional and has the usual meaning.

Similarly, you can change a partition location, effectively moving it:

```
ALTER TABLE log_messages PARTITION(year = 2011, month = 12,
day = 2)
SET LOCATION 's3n://ourbucket/logs/2011/01/02';
```

This command does not move the data from the old location, nor does it delete the old data.

Finally, you can drop a partition:

```
ALTER TABLE log_messages DROP IF EXISTS PARTITION(year = 2011,
month = 12, day = 2);
```

The IF EXISTS clause is optional, as usual. For managed tables, the data for the partition is *deleted*, along with the metadata, even if the partition was created using ALTER TABLE … ADD PARTITION. For external tables, the data is not deleted.

There are a few more ALTER statements that affect partitions discussed later in Alter Storage Properties and Miscellaneous Alter Table Statements.

Changing Columns

You can rename a column, change its position, type, or comment:

```
ALTER TABLE log_messages
CHANGE COLUMN hms hours_minutes_seconds INT
COMMENT 'The hours, minutes, and seconds part of the
timestamp'
AFTER severity;
```

You have to specify the old name, a new name, and the type, even if the name or type is not changing. The keyword COLUMN is optional as is the COMMENTclause. If you aren't moving the column, the AFTER other_column clause is not necessary. In the example shown, we move the column after the severitycolumn. If you want to move the column to the first position, use FIRST instead of AFTER other_column.

As always, this command changes metadata only. If you are moving columns, the data must already match the new schema or you must change it to match by some other means.

Adding Columns.

You can add new columns to the end of the existing columns, before any partition columns.

```
ALTER TABLE log_messages ADD COLUMNS (
  app_name      STRING COMMENT 'Application name',
  session_id LONG      COMMENT 'The current session id');
```

The COMMENT clauses are optional, as usual. If any of the new columns are in the wrong position, use an ALTER COLUMN table CHANGE COLUMN statement for each one to move it to the correct position.

Deleting or Replacing Columns

The following example removes *all* the existing columns and replaces them with the new columns specified:

```
ALTER TABLE log_messages REPLACE COLUMNS (
  hours_mins_secs INT                  COMMENT 'hour, minute, seconds from
timestamp',
  severity                  STRING COMMENT 'The message severity'
  message                   STRING COMMENT 'The rest of the message');
```

This statement effectively renames the original hms column and removes the server and process_id columns from the original schema definition. As for all ALTER statements, only the table metadata is changed.

The REPLACE statement can only be used with tables that use one of the native*SerDe* modules: DynamicSerDe or MetadataTypedColumn setSerDe. Recall that the SerDe determines how records are parsed into columns (deserialization) and how a record's columns are written to storage (serialization). SeeChapter 15 for more details on SerDes.

Alter Table Properties

You can add additional table properties or modify existing properties, but not remove them:

```
ALTER TABLE log_messages SET TBLPROPERTIES (
  'notes' = 'The process id is no longer captured; this column
```

is always NULL');

Alter Storage Properties

There are several ALTER TABLE statements for modifying format and SerDe properties.

The following statement changes the storage format for a partition to be SEQUENCEFILE, as we discussed in Creating Tables:

```
ALTER TABLE log_messages
PARTITION(year = 2012, month = 1, day = 1)
SET FILEFORMAT SEQUENCEFILE;
```

The PARTITION clause is required if the table is partitioned.

You can specify a new SerDe along with SerDe properties or change the properties for the existing SerDe. The following example specifies that a table will use a Java class named com.example.JSONSerDe to process a file of JSON-encoded records:

```
ALTER TABLE table_using_JSON_storage
SET SERDE 'com.example.JSONSerDe'
WITH SERDEPROPERTIES (
  'prop1' = 'value1',
  'prop2' = 'value2');
```

The SERDEPROPERTIES are passed to the SerDe module (the Java class com.example.JSONSerDe, in this case). Note that both the property names (e.g., prop1) and the values (e.g., value1) must be quoted strings.

The SERDEPROPERTIES feature is a convenient mechanism that SerDe implementations can exploit to permit user customization.

We'll see a real-world example of a JSON SerDe and how it uses SERDEPROPERTIES in JSON SerDe.

The following example demonstrates how to add new SERDEPROPERTIES for the current SerDe:

```
ALTER TABLE table_using_JSON_storage
SET SERDEPROPERTIES (
```

```
 'prop3' = 'value3',
 'prop4' = 'value4');
```

You can alter the storage properties that we discussed in Creating Tables:

```
ALTER TABLE stocks
CLUSTERED BY (exchange, symbol)
SORTED BY (symbol)
INTO 48 BUCKETS;
```

The SORTED BY clause is optional, but the CLUSTER BY and INTO … BUCKETS are required. (See also Bucketing Table Data Storage for information on the use of data bucketing.)

Miscellaneous Alter Table Statements

In Execution Hooks, we'll discuss a technique for adding execution

"hooks" for various operations. The ALTER TABLE … TOUCH statement is used to trigger these hooks:

```
ALTER TABLE log_messages TOUCH
PARTITION(year = 2012, month = 1, day = 1);
```

The PARTITION clause is required for partitioned tables. A typical scenario for this statement is to trigger execution of the hooks when table storage files have been modified outside of Hive. For example, a script that has just written new files for the 2012/01/01 partition for log_message can make the following call to the Hive CLI:

```
hive -e 'ALTER TABLE log_messages TOUCH PARTITION(year = 2012,
month = 1, day = 1);'
```

This statement won't create the table or partition if it doesn't already exist. Use the appropriate creation commands in that case.

The ALTER TABLE … ARCHIVE PARTITION statement captures the partition files into a Hadoop archive (HAR) file. This only reduces the number of files in the filesystem, reducing the load on the *NameNode*, but doesn't provide any space savings (e.g., through compression):

```
ALTER TABLE log_messages ARCHIVE
PARTITION(year = 2012, month = 1, day = 1);
```

To reverse the operation, substitute UNARCHIVE for ARCHIVE. This feature is only available for individual partitions of partitioned tables.

Finally, various protections are available. The following statements prevent the partition from being dropped and queried:

```
ALTER TABLE log_messages
PARTITION(year = 2012, month = 1, day = 1) ENABLE NO_DROP;
ALTER TABLE log_messages
PARTITION(year = 2012, month = 1, day = 1) ENABLE OFFLINE;
```

To reverse either operation, replace ENABLE with DISABLE. These operations also can't be used with nonpartitioned tables.

What is a View?

Views are similar to tables, which are generated based on the requirements.

- We can save any result set data as a view in Hive · Usage is similar to as views used in SQL
- All type of DML operations can be performed on a view Creation of View:

Syntax:

Create VIEW < VIEWNAME> AS SELECT

Example:

Hive>Create VIEW Sample_ViewAS SELECT * FROM employees WHERE salary>25000

In this example, we are creating view Sample_View where it will display all the row values with salary field greater than 25000.

What is Index?

Indexes are pointers to particular column name of a table.

- The user has to manually define the index
- Wherever we are creating index, it means that we are creating pointer to particular column name of table

- Any Changes made to the column present in tables are stored using the index value created on the column name.

Syntax:

Create INDEX < INDEX_NAME> ON TABLE <
TABLE_NAME(column names)>

Example:

Create INDEX sample_Index ON TABLE
guruhive_internaltable(id)
Here we are creating index on table guruhive_internaltable for column name id.

HIVE FUNCTIONS

Functions are built for a specific purpose to perform operations like Mathematical, arithmetic, logical and relational on the operands of table column names.

Built-in functions

These are functions that already available in Hive. First, we have to check the application requirement, and then we can use this built in functions in our applications. We can call these functions directly in our application.
The syntax and types are mentioned in the following section.
Types of Built-in Functions in HIVE

- Collection Functions ·
  Date Functions
- Mathematical Functions ·
  Conditional Functions
- String Functions ·
  Misc. Functions

Collection Functions:

These functions are used for collections. Collections mean the grouping of elements and returning single or array of elements depends on return type mentioned in function name.

Return Type	Function Name	Description
INT	size(Map<K.V>)	It will fetch and give the components number in the map type
INT	size(Array<T>)	It will fetch and give the elements number in the array type
Array <K>	Map_keys(Map <K.V>)	It will fetch and gives an array containing the keys of the input map. Here array is in unordered
Array <V>	Map_values(Ma p<K.V>)	It will fetch and gives an array containing the values of the input map. Here array is in unordered
Array <t>	Sort_array(Arra y<T>)	sorts the input array in ascending order of array and elements and returns it

Date Functions:

These are used to perform Date Manipulations and Conversion of Date types from one type to another type:

Function Name	Return Type	Description
Unix_Timestamp()	BigInt	We will get current Unix timestamp in seconds
To_date(string timestamp)	string	It will fetch and give the date part of a timestamp string:
year(string date)	INT	It will fetch and give the year part of a date or a timestamp string
quarter(date/ timestamp/string)	INT	It will fetch and give the quarter of the year for a date, timestamp, or string in the range 1 to 4
month(string date)	INT	It will give the month part of a date or a timestamp string
hour(string date)	INT	It will fetch and gives the hour of the timestamp
minute(string date)	INT	It will fetch and gives the minute of the timestamp
Date_sub(string starting date, int days)	string	It will fetch and gives Subtraction of number of days to starting date
Current_date	date	It will fetch and gives the current date at the start of query evaluation
LAST _day(string date)	string	It will fetch and gives the last day of the month which the date belongs to

trunc(string date, string format)	string	It will fetch and gives date truncated to the unit specified by the format. **Supported formats in this :** MONTH/MON/MM, YEAR/YYYY/ YY.

Mathematical Functions:

These functions are used for Mathematical Operations. Instead of creating UDFs, we have some inbuilt mathematical functions in Hive.

Function Name	Return Type	Description
round(DOUBL E X)	DOUBLE	It will fetch and returns the rounded BIGINT value of X
round(DOUBL E X, INT d)	DOUBLE	It will fetch and returns X rounded to d decimal places
bround(DOUB LE X)	DOUBLE	It will fetch and returns the rounded BIGINT value of X using HALF_EVEN rounding mode
floor(DOUBLE X)	BIGINT	It will fetch and returns the maximum BIGINT value that is equal to or less than X value
ceil(DOUBLE a), ceiling(DOUBL E a)	BIGINT	It will fetch and returns the minimum BIGINT value that is equal to or greater than X value
rand(), rand(INT seed)	DOUBLE	It will fetch and returns a random number that is distributed uniformly from 0 to 1

Conditional Functions:

These functions used for conditional values checks.

Function Name	Return Type	Description
if(Boolean testCondition, T valueTrue, T valueFalseOrNull)	T	It will fetch and gives value True when Test Condition is of true, gives value False Or Null otherwise.
ISNULL(X)	Boolean	It will fetch and gives true if X is NULL and false otherwise.
ISNOTNULL(X)	Boolean	It will fetch and gives true if X is not NULL and false otherwise.

String Functions:

String manipulations and string operations these functions can be called.

Function Name	Return Type	Description
reverse(str ing X)	string	It will give the reversed string of X
rpad(string str, int length, string pad)	string	It will fetch and gives str, which is right-padded with pad to a length of length(integer value)
rtrim(string X)	string	It will fetch and returns the string resulting from trimming spaces from the end (right hand side) of X **For example**, rtrim(' results ')
space(INT n)	string	It will fetch and gives a string of n spaces.
split(STRI NG str, STRING pat)	array	Splits str around pat (pat is a regular expression).
Str_to_ma p(text[, delimiter1, delimiter2])	map<S tring ,S tring>	It will split text into key-value pairs using two delimiters.

UDFs (User Defined Functions):

In Hive, the users can define own functions to meet certain client requirements. These are known as UDFs in Hive. User Defined Functions written in Java for specific modules.Some of UDFs are specifically designed for the reusability of code in application frameworks. The developer will develop these functions in Java and integrate those UDFs with the Hive.

During the Query execution, the developer can directly use the code, and UDFs will return outputs according to the user defined tasks. It will provide high performance in terms of coding and execution.

For example, for string stemming we don't have any predefined function in Hive, for this we can write stem UDF in Java. Wherever we require Stem functionality, we can directly call this Stem UDF in Hive.

Here stem functionality means deriving words from its root words. It is like stemming algorithm reduces the words "wishing", "wished", and "wishes" to the root word "fish." For performing this type functionality, we can write UDF in java and integrate with Hive.

Depending on the use cases the UDFs can be written, it will accept and produce different numbers of input and output values.

The general type of UDF will accept single input value and produce a single output value. If the UDF used in the query, then UDF will be called once for each row in the result data set.

In the other way, it can accept a group of values as input and return single output value as well.

Experiment – 9

Aim: Solve some real life big data problems.

Description:-

In this era where every aspect of our day-to-day life is gadget oriented, there is a huge volume of data that has been emanating from various digital sources.

Needless to say, we have faced a lot of challenges in the analysis and study of such a huge volume of data with the traditional data processing tools. To overcome these challenges, some big data solutions were introduced such as Hadoop. These big data tools really helped realize the applications of big data.

More and more organizations, both big and small, are leveraging from the benefits provided by big data applications. Businesses find that these benefits can help them grow fast.

Some of the famous applications of big data in detail:

1). Big Data in Education Industry

Education industry is flooding with huge amounts of data related to students, faculty, courses, results, and what not. Now, we have realized that proper study and analysis of this data can provide insights which can be used to improve the operational effectiveness and working of educational institutes.

Following are some of the fields in the education industry that have been transformed by big data-motivated changes:

- **Customized and Dynamic Learning Programs**

Customized programs and schemes to benefit individual students can be created using the data collected on the bases of each student's learning history. This improves the overall student results.

- **Reframing Course Material**

Reframing the course material according to the data that is collected on the basis of what a student learns and to what extent by real-time monitoring of the components of a course is beneficial for the students.

- **Grading Systems**

New advancements in grading systems have been introduced as a result of a proper analysis of student data.

- **Career Prediction**

Appropriate analysis and study of every student's records will help understand each student's progress, strengths, weaknesses, interests, and more. It would also help in determining which career would be the most suitable for the student in future.

The applications of big data have provided a solution to one of the biggest pitfalls in the education system, that is, the one-size-fits-all fashion of academic set-up, by contributing in e-learning solutions.

- **Example**

The University of Alabama has more than 38,000 students and an ocean of data. In the past when there were no real solutions to analyze that much of data, some of them seemed useless. Now, administrators are able to use analytics and data visualizations for this data to draw out patterns of students revolutionizing the university's operations, recruitment, and retention efforts.

2). Big Data in Healthcare Industry

Healthcare is yet another industry which is bound to generate a huge amount of data. Following are some of the ways in which big data has contributed to healthcare:

- Big data reduces costs of treatment since there is less chances of having to perform unnecessary diagnosis.
- It helps in predicting outbreaks of epidemics and also in deciding what preventive measures could be taken to minimize the effects of the same.

- It helps avoid preventable diseases by detecting them in early stages. It prevents them from getting any worse which in turn makes their treatment easy and effective.
- Patients can be provided with evidence-based medicine which is identified and prescribed after doing research on past medical results.

- **Example**

Wearable devices and sensors have been introduced in the healthcare industry which can provide real-time feed to the electronic health record of a patient. One such technology is from Apple.

Apple has come up with Apple HealthKit, CareKit, and ResearchKit. The main goal is to empower the iPhone users to store and access their real-time health records on their phones.

3). Big Data in Government Sector

Governments, be it of any country, come face to face with a very huge amount of data on almost daily basis. The reason for this is, they have to keep track of various records and databases regarding their citizens, their growth, energy resources, geographical surveys, and many more. All this data contributes to big data. The proper study and analysis of this data, hence, helps governments in endless ways. Few of them are as follows:

Welfare Schemes

- In making faster and informed decisions regarding various political programs
- To identify areas that are in immediate need of attention
- To stay up to date in the field of agriculture by keeping track of all existing land and livestock.

- To overcome national challenges such as unemployment, terrorism, energy resources exploration, and much more.

Cyber Security

- Big Data is hugely used for deceit recognition.
- It is also used in catching tax evaders.

- **Example**

Food and Drug Administration (FDA) which runs under the jurisdiction of the Federal Government of USA leverages from the analysis of big data to discover patters and associations in order to identify and examine the expected or unexpected occurrences of food-based infections.

4). Big Data in Media and Entertainment Industry

With people having access to various digital gadgets, generation of large amount of data is inevitable and this is the main cause of the rise in big data in media and entertainment industry.

Other than this, social media platforms are another way in which huge amount of data is being generated. Although, businesses in the media and entertainment industry have realized the importance of this data, and they have been able to benefit from it for their growth.

Some of the benefits extracted from big data in the media and entertainment industry are given below:

- Predicting the interests of audiences
- Optimized or on-demand scheduling of media streams in digital media distribution platforms
- Getting insights from customer reviews
- Effective targeting of the advertisements

- **Example**

Spotify, an on-demand music providing platform, uses Big Data Analytics, collects data from all its users around the globe, and then uses the analyzed data to give informed music recommendations and suggestions to every individual user.

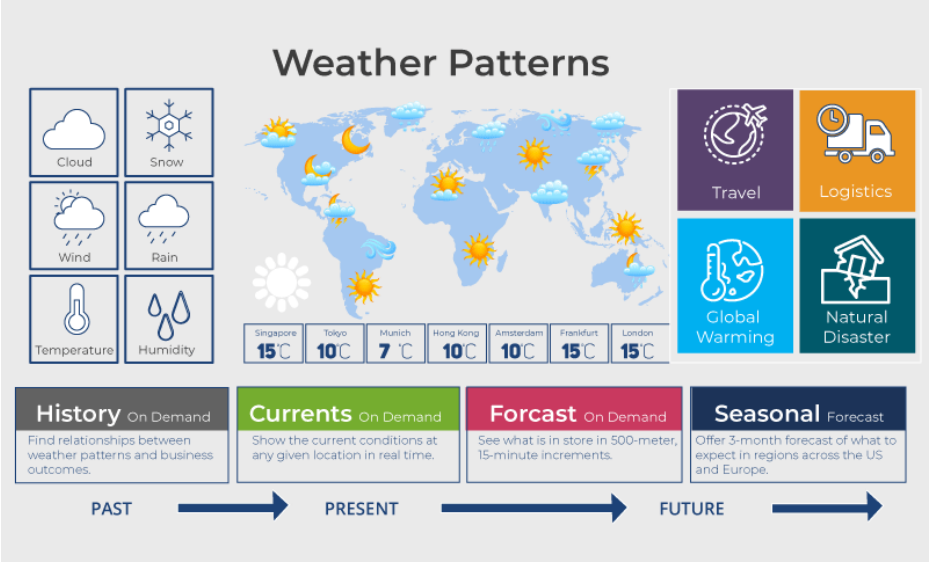
Amazon Prime that offers, videos, music, and Kindle books in a one-stop shop is also big on using big data.

5). Big Data in Weather Patterns

There are weather sensors and satellites deployed all around the globe. A huge amount of data is collected from them, and then this data is used to monitor the weather and environmental conditions.

All of the data collected from these sensors and satellites contribute to big data and can be used in different ways such as:

- In weather forecasting
- To study global warming
- In understanding the patterns of natural disasters
- To make necessary preparations in the case of crises
- To predict the availability of usable water around the world



- **Example**

IBM Deep Thunder, which is a research project by IBM, provides weather forecasting through high-performance computing of big data. IBM is also assisting Tokyo with the improved weather forecasting for natural disasters or predicting the probability of damaged power lines.

6). Big Data in Transportation Industry

Since the rise of big data, it has been used in various ways to make transportation more efficient and easy.

Following are some of the areas where big data contributes to transportation.

- **Route planning**: Big data can be used to understand and estimate users' needs on different routes and on multiple modes of transportation and then utilize route planning to reduce their wait time.
- **Congestion management and traffic control**: Using big data, real-time estimation of congestion and traffic patterns is now possible. For examples, people are using Google Maps to locate the least traffic-prone routes.
- **Safety level of traffic**: Using the real-time processing of big data and predictive analysis to identify accident-prone areas can help reduce accidents and increase the safety level of traffic.

- **Example**

Let's take Uber as an example here. Uber generates and uses a huge amount of data regarding drivers, their vehicles, locations, every trip from every vehicle, etc. All this data is analyzed and then used to predict supply, demand, location of drivers, and fares that will be set for every trip.

And guess what? We too make use of this application when we choose a route to save fuel and time, based on our knowledge of having taken that particular route sometime in the past. In this case, we analyzed and made use of the data that we had previously acquired on account of our experience, and then we used it to make a smart decision. It's pretty cool that big data has played parts not only in big fields but also in our smallest day-to-day life decisions too.

7). Big Data in Banking Sector

The amount of data in the banking sector is skyrocketing every second. According to GDC prognosis, this data is estimated to grow 700 percent by the end of the next year. Proper study and analysis of this data can help detect any and all illegal activities that are being carried out such as:

-
  - Misuse of credit/debit cards
  - Venture credit hazard treatment
  - Business clarity
  - Customer statistics alteration
  - Money laundering
  - Risk mitigation

- **Example**

Various anti-money laundering software such as SAS AML use **Data Analytics in Banking** for the purpose of detecting suspicious transactions and analyzing customer data. Bank of America has been a SAS AML customer for more than 25 years.

We have seen some of the applications of big data in real world. No wonder, there is so much hype for big data, given all of its applications. The importance of big data lies in how an organization is using the collected data and not in how much data they have been able to collect. There are Big Data solutions that make the analysis of big data easy and efficient. These Big Data solutions are used to gain benefits from the heaping amounts of data in almost all industry verticals.

Beyond Syllabus

Experiment :- 1

1. **Objective:** Write a Program for Developing and Running a Spark WordCount Application.

Description:

The application is an enhanced version of WordCount, the canonical MapReduce example. In this version of WordCount, the goal is to learn the distribution of letters in the most popular words in a corpus.

The application:

1) Creates a SparkConf and SparkContext. A Spark application corresponds to an instance of the SparkContext class. When running a shell, the SparkContext is created for you.
2) Gets a word frequency threshold.
3) Reads an input set of text documents.
4) Counts the number of times each word appears.
5) Filters out all words that appear fewer times than the threshold.
6) For the remaining words, counts the number of times each letter occurs.

In MapReduce, this requires two MapReduce applications, as well as persisting the intermediate data to HDFS between them. In Spark, this application requires about 90 percent fewer lines of code than one developed using the MapReduce API.

Scala WordCount Program:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SparkWordCount {
  def main(args: Array[String]) {
    // create Spark context with Spark configuration
    val sc = new SparkContext(new SparkConf().setAppName("Spark Count"))

    // get threshold
    val threshold = args(1).toInt

    // read in text file and split each document into words
    val tokenized = sc.textFile(args(0)).flatMap(_.split(" "))
```

```
    // count the occurrence of each word
    val wordCounts = tokenized.map((_, 1)).reduceByKey(_ + _)

    // filter out words with fewer than threshold occurrences
    val filtered = wordCounts.filter(_._2 >= threshold)

    // count characters
    val charCounts = filtered.flatMap(_._1.toCharArray).map((_, 1)).reduceByKey(_ + _)

    System.out.println(charCounts.collect().mkString(", "))
  }
}
```

Viva Questions

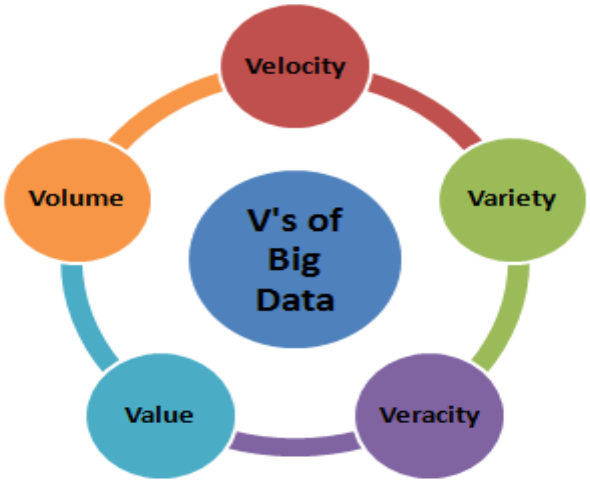
1. What do you know about the term "Big Data"?

Answer: Big Data is a term associated with complex and large datasets. A relational database cannot handle big data, and that's why special tools and methods are used to perform operations on a vast collection of data. Big data enables companies to understand their business better and helps them derive meaningful information from the unstructured and raw data collected on a regular basis. Big data also allows the companies to take better business decisions backed by data.

2. What are the five V's of Big Data?

Answer: The five V's of Big data is as follows:

- **Volume –** Volume represents the volume i.e. amount of data that is growing at a high rate i.e. data volume in Petabytes
- **Velocity –** Velocity is the rate at which data grows. Social media contributes a major role in the velocity of growing data.
- **Variety –** Variety refers to the different data types i.e. various data formats like text, audios, videos, etc.
- **Veracity –** Veracity refers to the uncertainty of available data. Veracity arises due to the high volume of data that brings incompleteness and inconsistency.
- **Value –**Value refers to turning data into value. By turning accessed big data into values, businesses may generate revenue.



5 V's of Big Data

3. Tell us how big data and Hadoop are related to each other.

Answer: Big data and Hadoop are almost synonyms terms. With the rise of big data, Hadoop, a framework that specializes in big data operations also became popular. The framework can be used by professionals to analyze big data and help businesses to make decisions.

Note: This question is commonly asked in a big data interview. *Y*ou can go further to answer this question and try to explain the main components of Hadoop.

4. How is big data analysis helpful in increasing business revenue?

Answer: Big data analysis has become very important for the businesses. It helps businesses to differentiate themselves from others and increase the revenue. Through predictive analytics, big data analytics provides businesses customized recommendations and suggestions. Also, big data analytics enables businesses to launch new products depending on customer needs and preferences. These factors make businesses earn more revenue, and thus companies are using big data analytics. Companies may encounter a significant increase of 5-20% in revenue by implementing big data analytics. Some popular companies those are using big data analytics to increase their revenue is – Walmart, LinkedIn, Facebook, Twitter, Bank of America etc.

5. Explain the steps to be followed to deploy a Big Data solution.

Answer: Followings are the three steps that are followed to deploy a Big Data Solution –

i. Data Ingestion

The first step for deploying a big data solution is the data ingestion i.e. extraction of data from various sources. The data source may be a CRM like Salesforce, Enterprise Resource Planning System like SAP, RDBMS like MySQL or any other log files, documents, social media feeds etc. The data can be ingested either through batch jobs or real-time streaming. The extracted data is then stored in HDFS.

Steps of Deploying Big Data Solution

ii. Data Storage

After data ingestion, the next step is to store the extracted data. The data either be stored in HDFS or NoSQL database (i.e. HBase). The HDFS storage works well for sequential access whereas HBase for random read/write access.

iii. Data Processing

The final step in deploying a big data solution is the data processing. The data is processed through one of the processing frameworks like Spark, MapReduce, Pig, etc.

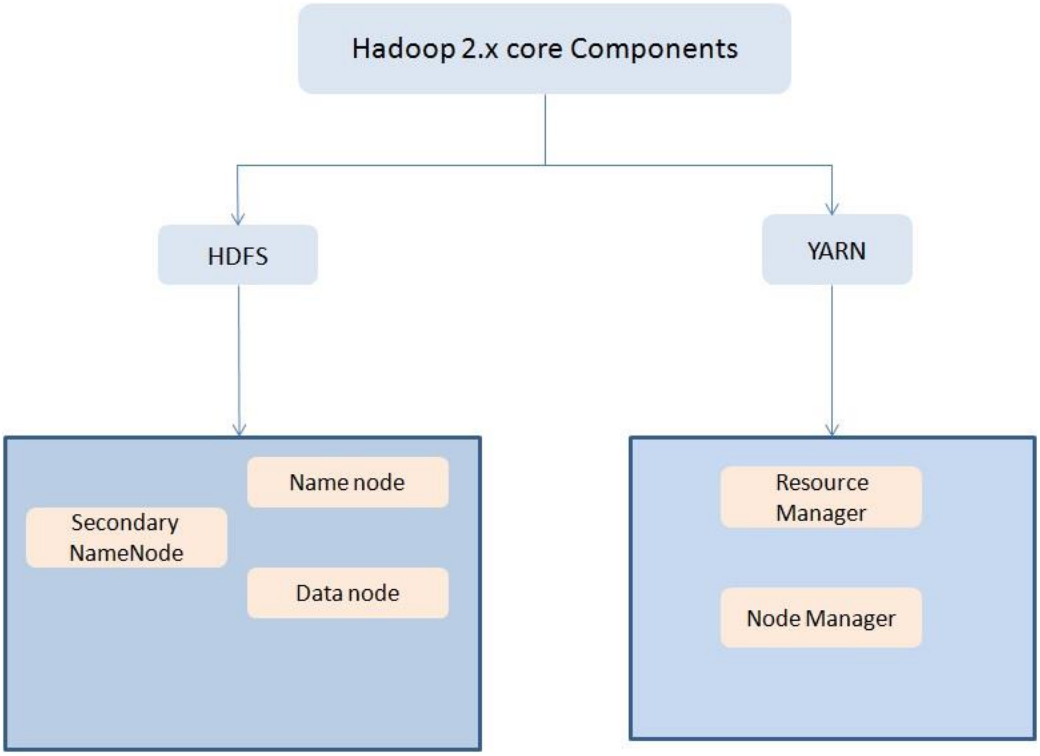
6. Define respective components of HDFS and YARN

Answer: The two main components of HDFS are-

- NameNode – This is the master node for processing metadata information for data blocks within the HDFS
- DataNode/Slave node – This is the node which acts as slave node to store the data, for processing and use by the NameNode

In addition to serving the client requests, the NameNode executes either of two following roles –

- CheckpointNode – It runs on a different host from the NameNode
- BackupNode- It is a read-only NameNode which contains file system metadata information excluding the block locations



The two main components of YARN are–

- ResourceManager– This component receives processing requests and accordingly allocates to respective NodeManagers depending on processing needs.
- NodeManager– It executes tasks on each single Data Node

7. Why is Hadoop used for Big Data Analytics?

Answer: Since data analysis has become one of the key parameters of business, hence, enterprises are dealing with massive amount of structured, unstructured and semi-structured data. Analyzing unstructured data is quite difficult where Hadoop takes major part with its capabilities of

- Storage
- Processing
- Data collection

Moreover, Hadoop is open source and runs on commodity hardware. Hence it is a cost-benefit solution for businesses.

8. What is fsck?

Answer: fsck stands for File System Check. It is a command used by HDFS. This command is used to check inconsistencies and if there is any problem in the file. For example, if there are any missing blocks for a file, HDFS gets notified through this command.

9. What are the main differences between NAS (Network-attached storage) and HDFS?

Answer: The main differences between NAS (Network-attached storage) and HDFS –

- HDFS runs on a cluster of machines while NAS runs on an individual machine. Hence, data redundancy is a common issue in HDFS. On the contrary, the replication protocol is different in case of NAS. Thus the chances of data redundancy are much less.
- Data is stored as data blocks in local drives in case of HDFS. In case of NAS, it is stored in dedicated hardware.

10. What is the Command to format the NameNode?

Answer: $ hdfs namenode -format

11. Which hardware configuration is most beneficial for Hadoop jobs?

Dual processors or core machines with a configuration of 4 / 8 GB RAM and ECC memory is ideal for running Hadoop operations. However, the hardware configuration varies based on the project-specific workflow and process flow and need customization accordingly.

12. What happens when two users try to access the same file in the HDFS?

HDFS NameNode supports exclusive write only. Hence, only the first user will receive the grant for file access and the second user will be rejected.

13. How to recover a NameNode when it is down?

The following steps need to execute to make the Hadoop cluster up and running:

1. Use the FsImage which is file system metadata replica to start a new NameNode.
2. Configure the DataNodes and also the clients to make them acknowledge the newly started NameNode.
3. Once the new NameNode completes loading the last checkpoint FsImage which has received enough block reports from the DataNodes, it will start to serve the client.

In case of large Hadoop clusters, the NameNode recovery process consumes a lot of time which turns out to be a more significant challenge in case of routine maintenance.

14. What do you understand by Rack Awareness in Hadoop?

It is an algorithm applied to the NameNode to decide how blocks and its replicas are placed. Depending on rack definitions network traffic is minimized between DataNodes within the same rack. For example, if we consider replication factor as 3, two copies will be placed on one rack whereas the third copy in a separate rack.

15. What is the difference between “HDFS Block” and “Input Split”?

The HDFS divides the input data physically into blocks for processing which is known as HDFS Block.

Input Split is a logical division of data by mapper for mapping operation.

Hadoop is one of the most popular Big Data frameworks, and if you are going for a Hadoop interview prepare yourself with these basic level interview questions for Big Data Hadoop. These

questions will be helpful for you whether you are going for a Hadoop developer or Hadoop Admin interview.

16. Explain the difference between Hadoop and RDBMS.

Answer: The difference between Hadoop and RDBMS is as follows –

Criteria	Hadoop	RDBMS
Schema	Based on 'Schema on Read'	Based on 'Schema on Write'
Data Types	Structured, Semi-structured and Unstructured data	Structured Data
Speed	Writes are Fast	Reads are Fast
Cost	Open source framework, free of cost	Licensed software, paid
Applications	Data discovery, Storage and processing of unstructured data	OLTP and complex ACID transactions

17. What are the common input formats in Hadoop?

Answer: Below are the common input formats in Hadoop –

- **Text Input Format –** The default input format defined in Hadoop is the Text Input Format.
- **Sequence File Input Format –** To read files in a sequence, Sequence File Input Format is used.
- **Key Value Input Format –** The input format used for plain text files (files broken into lines) is the Key Value Input Format.

18. Explain some important features of Hadoop.

Answer: Hadoop supports the storage and processing of big data. It is the best solution for handling big data challenges. Some important features of Hadoop are –

- **Open Source –** Hadoop is an open source framework which means it is available free of cost. Also, the users are allowed to change the source code as per their requirements.
- **Distributed Processing–** Hadoop supports distributed processing of data i.e. faster processing. The data in Hadoop HDFS is stored in a distributed manner and MapReduce is responsible for the parallel processing of data.

- **Fault Tolerance–** Hadoop is highly fault-tolerant. It creates three replicas for each block at different nodes, by default. This number can be changed according to the requirement. So, we can recover the data from another node if one node fails. The detection of node failure and recovery of data is done automatically.
- **Reliability–** Hadoop stores data on the cluster in a reliable manner that is independent of machine. So, the data stored in Hadoop environment is not affected by the failure of the machine.
- **Scalability–** Another important feature of Hadoop is the scalability. It is compatible with the other hardware and we can easily ass the new hardware to the nodes.
- **High Availability –** The data stored in Hadoop is available to access even after the hardware failure. In case of hardware failure, the data can be accessed from another path.

19. Explain the different modes in which Hadoop run.

Answer: Apache Hadoop runs in the following three modes –

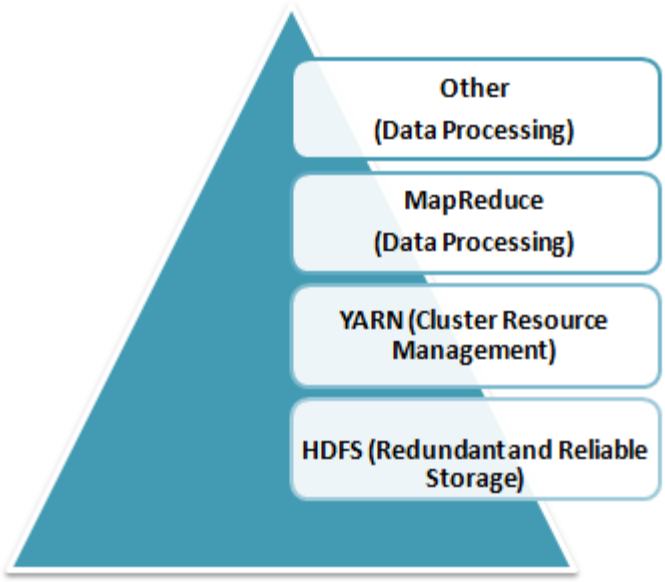
- **Standalone (Local) Mode –** By default, Hadoop runs in a local mode i.e. on a non-distributed, single node. This mode uses the local file system to perform input and output operation. This mode does not support the use of HDFS, so it is used for debugging. No custom configuration is needed for configuration files in this mode.
- **Pseudo-Distributed Mode –** In the pseudo-distributed mode, Hadoop runs on a single node just like the Standalone mode. In this mode, each daemon runs in a separate Java process. As all the daemons run on a single node, there is the same node for both the Master and Slave nodes.
- **Fully – Distributed Mode –** In the fully-distributed mode, all the daemons run on separate individual nodes and thus forms a multi-node cluster. There are different nodes for Master and Slave nodes.

20. Explain the core components of Hadoop.

Answer: Hadoop is an open source framework that is meant for storage and processing of big data in a distributed manner.

The core components of Hadoop are –

- **HDFS (Hadoop Distributed File System) –** HDFS is the basic storage system of Hadoop. The large data files running on a cluster of commodity hardware are stored in HDFS. It can store data in a reliable manner even when hardware fails.



Core Components of Hadoop

- **Hadoop MapReduce –** MapReduce is the Hadoop layer that is responsible for data processing. It writes an application to process unstructured and structured data stored in HDFS. It is responsible for the parallel processing of high volume of data by dividing data into independent tasks. The processing is done in two phases Map and Reduce. The Map is the first phase of processing that specifies complex logic code and the Reduce is the second phase of processing that specifies light-weight operations.
- **YARN –** The processing framework in Hadoop is YARN. It is used for resource management and provides multiple data processing engines i.e. data science, real-time streaming, and batch processing.

21. What are the configuration parameters in a "MapReduce" program?

The main configuration parameters in "MapReduce" framework are:

- Input locations of Jobs in the distributed file system
- Output location of Jobs in the distributed file system
- The input format of data
- The output format of data
- The class which contains the map function
- The class which contains the reduce function
- JAR file which contains the mapper, reducer and the driver classes

22. What is a block in HDFS and what is its default size in Hadoop 1 and Hadoop 2? Can we change the block size?

Blocks are smallest continuous data storage in a hard drive. For HDFS, blocks are stored across Hadoop cluster.

- The default block size in Hadoop 1 is: 64 MB
- The default block size in Hadoop 2 is: 128 MB

Yes, we can change block size by using the parameter – **dfs.block.size** located in the hdfs-site.xml file.

23. What is Distributed Cache in a MapReduce Framework

Distributed Cache is a feature of Hadoop MapReduce framework to cache files for applications. Hadoop framework makes cached files available for every map/reduce tasks running on the data nodes. Hence, the data files can access the cache file as a local file in the designated job.

24. What are the three running modes of Hadoop?

The three running modes of Hadoop are as follows:

i. Standalone or local: This is the default mode and does not need any configuration. In this mode, all the following components of Hadoop uses local file system and runs on a single JVM –

- NameNode
- DataNode
- ResourceManager
- NodeManager

ii. Pseudo-distributed: *In this mode, all the master and slave Hadoop services are deployed and executed on a single node.*

iii. Fully distributed: In this mode, Hadoop master and slave services are deployed and executed on separate nodes.

25. Explain JobTracker in Hadoop

JobTracker is a JVM process in Hadoop to submit and track MapReduce jobs.

JobTracker performs the following activities in Hadoop in a sequence –

- JobTracker receives jobs that a client application submits to the job tracker
- JobTracker notifies NameNode to determine data node
- JobTracker allocates TaskTracker nodes based on available slots.
- it submits the work on allocated TaskTracker Nodes,
- JobTracker monitors the TaskTracker nodes.
- When a task fails, JobTracker is notified and decides how to reallocate the task.

26. What are the different configuration files in Hadoop?

Answer: The different configuration files in Hadoop are –

core-site.xml – This configuration file contains Hadoop core configuration settings, for example, I/O settings, very common for MapReduce and HDFS. It uses hostname a port.

mapred-site.xml – This configuration file specifies a framework name for MapReduce by setting mapreduce.framework.name

hdfs-site.xml – This configuration file contains HDFS daemons configuration settings. It also specifies default block permission and replication checking on HDFS.

yarn-site.xml – This configuration file specifies configuration settings for ResourceManager and NodeManager.

27. What are the differences between Hadoop 2 and Hadoop 3?

Answer: Following are the differences between Hadoop 2 and Hadoop 3 –

Criteria	Hadoop 2	Hadoop 3
Java Version	Minimum supported Java Version is Java 7	Minimum supported Java Version is Java 8
Fault Tolerance	Fault tolerance is handled by Replication that waste the space	Fault tolerance is handled by Erasure coding
Data Balancing	HDFS balancer is used for data balancing	Intra-datanode balancer is used for data balancing
Overhead in Storage Space	HDFS has 200% overhead in storage space	HDFS has 50% overhead in storage space
Default Ports	Some default ports are in ephemeral range, so fails to bind at start up	All the default ports are out of the ephemeral range, binds well at start up

28. How can you achieve security in Hadoop?

Answer: Kerberos are used to achieve security in Hadoop. There are 3 steps to access a service while using Kerberos, at a high level. Each step involves a message exchange with a server.

1. **Authentication** – The first step involves authentication of the client to the authentication server, and then provides a time-stamped TGT (Ticket-Granting Ticket) to the client.
2. **Authorization** – In this step, the client uses received TGT to request a service ticket from the TGS (Ticket Granting Server).
3. **Service Request** – It is the final step to achieve security in Hadoop. Then the client uses service ticket to authenticate himself to the server.

29. What is commodity hardware?

Answer: Commodity hardware is a low-cost system identified by less-availability and low-quality. The commodity hardware comprises of RAM as it performs a number of services that require RAM for the execution. One doesn't require high-end hardware configuration or supercomputers to run Hadoop, it can be run on any commodity hardware.

30. How is NFS different from HDFS?

Answer: There are a number of distributed file systems that work in their own way. NFS (Network File System) is one of the oldest and popular distributed file storage systems whereas

HDFS (Hadoop Distributed File System) is the recently used and popular one to handle big data. The main differences between NFS and HDFS are as follows –

Criteria	NFS	HDFS
Data Size Support	NFS can store and process small amount of data	HDFS is mainly use to store and process big data
Data Storage	Data is stored on a single dedicated hardware	The data blocks are distributed on the local drives of hardware
Reliability	No reliability, data is not available in case of machine failure	Data is stored reliably, data is available even after machine failure
Data Redundancy	NFS runs on single machine, no chances of data redundancy	HDFS runs on a cluster of different machines, data redundancy may occur due to replication protocol

31. How do Hadoop MapReduce works?

There are two phases of MapReduce operation.

- Map phase – In this phase, the input data is split by map tasks. The map tasks run in parallel. These split data is used for analysis purpose.
- Reduce phase- In this phase, the similar split data is aggregated from the entire collection and shows the result.

32. What is MapReduce? What is the syntax you use to run a MapReduce program?

MapReduce is a programming model in Hadoop for processing large data sets over a cluster of computers, commonly known as HDFS. It is a parallel programming model.

The syntax to run a MapReduce program is – *hadoop_jar_file.jar /input_path /output_path.*

33. What are the Port Numbers for NameNode, Task Tracker, and Job Tracker?

- **NameNode** – Port 50070
- **Task Tracker** – Port 50060
- **Job Tracker** – Port 50030

34. What are the different file permissions in HDFS for files or directory levels?

Hadoop distributed file system (HDFS) uses a specific permissions model for files and directories.

Following user levels are used in HDFS –

- Owner
- Group
- Others.

For each of the user mentioned above following permissions are applicable –

- read (r)
- write (w)
- execute(x).

Above mentioned permissions work differently for files and directories.

For files –

- The **r** permission is for *reading* a file
- The **w** permission is for *writing* a file.

For directories –

- The **r** permission *lists the contents* of *a specific directory.*
- The **w** permission *creates or deletes a directory.*
- The **X** permission is for accessing a child directory.

35. What are the basic parameters of a Mapper?

The basic parameters of a Mapper are

- LongWritable and Text
- Text and IntWritable

36. How to restart all the daemons in Hadoop?

Answer: To restart all the daemons, it is required to stop all the daemons first. The Hadoop directory contains sbin directory that stores the script files to stop and start daemons in Hadoop.

Use stop daemons command /sbin/stop-all.sh to stop all the daemons and then use /sin/start-all.sh command to start all the daemons again.

37. What is the use of jps command in Hadoop?

Answer: The jps command is used to check if the Hadoop daemons are running properly or not. This command shows all the daemons running on a machine i.e. Datanode, Namenode, NodeManager, ResourceManager etc.

38. Explain the process that overwrites the replication factors in HDFS.

Answer: There are two methods to overwrite the replication factors in HDFS –

Method 1: On File Basis

In this method, the replication factor is changed on the basis of file using Hadoop FS shell. The command used for this is:

$hadoop fs – setrep –w2/my/test_file

Here, test_file is the filename that's replication factor will be set to 2.

Method 2: On Directory Basis

In this method, the replication factor is changed on directory basis i.e. the replication factor for all the files under a given directory is modified.

$hadoop fs –setrep –w5/my/test_dir

Here, test_dir is the name of the directory, the replication factor for the directory and all the files in it will be set to 5.

39. What will happen with a NameNode that doesn't have any data?

Answer: A NameNode without any data doesn't exist in Hadoop. If there is a NameNode, it will contain some data in it or it won't exist.

40. Explain NameNode recovery process.

Answer: The NameNode recovery process involves the below-mentioned steps to make Hadoop cluster running:

- In the first step in the recovery process, file system metadata replica (FsImage) starts a new NameNode.
- The next step is to configure DataNodes and Clients. These DataNodes and Clients will then acknowledge new NameNode.
- During the final step, the new NameNode starts serving the client on the completion of last checkpoint FsImage loading and receiving block reports from the DataNodes.

Note: Don't forget to mention, this NameNode recovery process consumes a lot of time on large Hadoop clusters. Thus, it makes routine maintenance difficult. For this reason, HDFS high availability architecture is recommended to use.

41. How Is Hadoop CLASSPATH essential to start or stop Hadoop daemons?

CLASSPATH includes necessary directories that contain jar files to start or stop Hadoop daemons. Hence, setting CLASSPATH is essential to start or stop Hadoop daemons.

However, setting up CLASSPATH every time is not the standard that we follow. Usually CLASSPATH is written inside */etc/hadoop/hadoop-env.sh* file. Hence, once we run Hadoop, it will load the CLASSPATH automatically.

42. Why is HDFS only suitable for large data sets and not the correct tool to use for many small files?

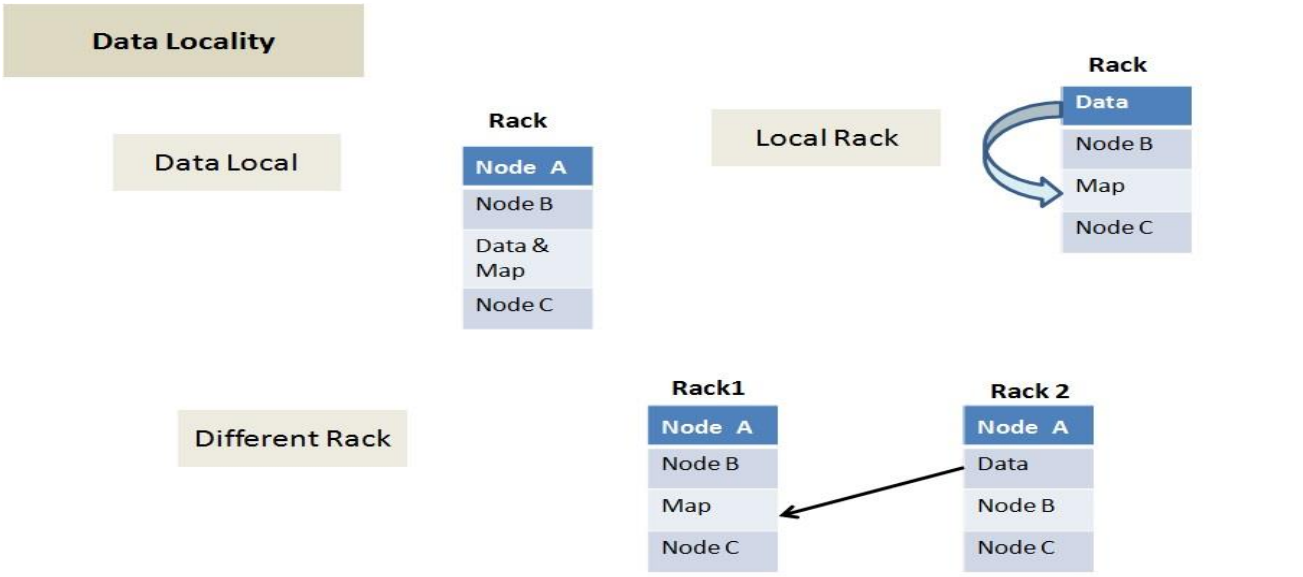
This is due to the performance issue of NameNode. Usually, NameNode is allocated with huge space to store metadata for the large-scale file. The metadata is supposed to be a from a single file for optimum space utilization and cost benefit. In case of small size files, NameNode does not utilize the entire space which is a performance optimization issue.

43. Why do we need Data Locality in Hadoop? Explain.

Datasets in HDFS store as blocks in DataNodes the Hadoop cluster. During the execution of a MapReduce job the individual Mapper processes the blocks (Input Splits). If the data does not reside in the same node where the Mapper is executing the job, the data needs to be copied from the DataNode over the network to the mapper DataNode.

Now if a MapReduce job has more than 100 Mapper and each Mapper tries to copy the data from other DataNode in the cluster simultaneously, it would cause serious network congestion which is a big performance issue of the overall system. Hence, data proximity to the computation is an

effective and cost-effective solution which is technically termed as Data locality in Hadoop. It helps to increase the overall throughput of the system.



Data locality can be of three types:

- **Data local –** In this type data and the mapper resides on the same node. This is the closest proximity of data and the most preferred scenario.
- **Rack Local –** In this scenarios mapper and data reside on the same rack but on the different data nodes.
- **Different Rack –** In this scenario mapper and data reside on the different racks.

44. DFS can handle a large volume of data then why do we need Hadoop framework?

Hadoop is not only for storing large data but also to process those big data. Though DFS(Distributed File System) too can store the data, but it lacks below features-

- It is not fault tolerant
- Data movement over a network depends on bandwidth.

45. What is Sequence file input format?

Hadoop uses a specific file format which is known as Sequence file. The sequence file stores data in a serialized key-value pair. Sequence file input format is an input format to read sequence files.

References:

TEXT & REFERENCE BOOKS

1. Michael Minelli, Michelle Chambers, and AmbigaDhiraj, "Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses", Wiley, 2013.
2. P. J. Sadalage and M. Fowler, "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence", Addison-Wesley Professional, 2012.
3. Tom White, "Hadoop: The Definitive Guide", Third Edition, O'Reilley, 2012.
4. Eric Sammer, "Hadoop Operations", O'Reilley, 2012.
5. E. Capriolo, D. Wampler, and J. Rutherglen, "Programming Hive", O'Reilley, 2012.
6. Lars George, "HBase: The Definitive Guide", O'Reilley, 2011.
7. Eben Hewitt, "Cassandra: The Definitive Guide", O'Reilley, 2010.
8. Alan Gates, "Programming Pig", O'Reilley, 2011.

VISION (SKIT)

"To Promote Higher Learning in Advanced Technology and Industrial Research to make our Country a Global Player"

MISSION (SKIT)

"To Promote Quality Education, Training and Research in the field of Engineering by establishing effective interface with Industry and to encourage Faculty to undertake Industry Sponsored Projects for Students"

QUALITY POLICY (SKIT)

"We are committed to 'achievement of quality' as an integral part of our institutional policy by continuous self-evaluation and striving to improve ourselves.

Institute would pursue quality in:

- All its endeavors like admissions, teaching-learning processes, examinations, extra and co-curricular activities, industry-institution interaction, research & development, continuing education, and consultancy.
- Functional areas like teaching departments, training & placement cell, library, administrative office, accounts office, hostels, canteen, security services, transport, maintenance section and all other services."

Vision and Mission of CSE Department

VISION (CSE Department)

"Producing quality graduates trained in the latest tools and technologies and to be a leading department in the state and region by imparting in-depth knowledge to the students in an emerging technologies in computer science & engineering."

MISSION (CSE Department)

"Delivering Student's resources in IT enabled domain through:

- Effective Industry interaction and project based learning
- Motivating our graduates for Employability, entrepreneurship, research and higher education
- Providing excellent engineering skills in a state-of-the-art infrastructure"

DEPARTMENT OF COMPUTER SCIENCE& ENGINEERING

Program Educational Objectives (PEOs)

Graduate of the CSE Program will be:

- Prepared to be employed in IT industries and be engaged in learning, understanding, and applying new ideas.
- Able to apply their technical knowledge as practicing professionals or engage in graduate education.
- Prepared to be responsible computing professionals in their own area of interest.
- Working successfully in their chosen career individually and within a professional team environment.

Program Outcomes for CSE Department

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes for CSE Department

PSO1: Graduates will possess strong fundamental concepts on database technologies, Operating systems, advanced programming, Software engineering and other core subjects.

PSO2: Graduates will demonstrate an ability to design, develop, test, debug, deploy, analyze, troubleshoot, maintain, manage and secure the software.

PSO3: Graduates will possess the knowledge of institutions / organizations / companies related to computer science & engineering.