Output -

HDFS 1
Hadoop 2
MapReduce 1
a 2

processing 1
Storage 1
tool 1
unit 1

Experiment - 4

Objective - Run a basic word count MapReduce Program to understand MapReduce Paradigm.

→ **Steps to follow** -

a) Create a directory in HDFS, where to kept text file

$ HDFS dfs - mkdir /test

b) Upload the data.txt file on HDFS in the specific directory

$ Hdfs dfs - put / home / codegyan / data.txt / test

c) Write the MapReduce Program using eclipse

→ **Code** -

```
package com
import java.io.*;
import java.util.*;
import org.apache.hadoop.*;
public class Mapper extends MapReduceBase
  implement Mapper < LongWritable, Text, Text,
    IntWritable>
{ private final static IntWritable one = new
                                  IntWritable();
private Text word = new Text();
```

```
public void map(LongWritable key, Text value,
                                    output collector)
{ String line = value.toString();
  StringTokenizer tokenizer = new StringTokenizer
                                    (line);
while (tokenizer.hasMoreToken.s())
{
 word.set(tokenizer.nextToken());
 output.collect(word, one);
}
}
}
```

Output -

The Day is Cold Day : 20200101 - 21.8

The Day is Cold Day : 20201102 - 23.4

The Day is Cold Day : 20201103 - 25.4

The Day is Cold Day : 20201104 - 20.0

The Day is Cold Day : 20201105 - 24.8

EXPERIMENT-5

POORNIMA

Objective:- Write a Map Reduce program that mines weather data. Weather sensors collecting data everyhour 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 ~~text~~ semi structured and record-oriented.

Mapper Code:-

```
package com.projectxyz.wc;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.io.LongWritable;
public class WordCountMapper extends Mapper <LongWritable,
    Text, Text, IntWritable>
{
  private Text wordToken = new Text();
public void map (LongWritable key, Text value, Context context) throws
  IOException, InterruptedException
  {
StringTokenizer tokens = new StringTokenizer (value.toString());
while (tokens.hasMoreToken())
  {
wordToken.set(tokin.nextToken());
context.write(wordToken, new IntWritable(1));
}}
}
```

POORNIMA

Reducer Code-

```
package com.projectgurukul.wc;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class WordCountReducer extends Reducer <Text, IntWritable,
  Text, IntWritable> {
private IntWritable count= new IntWritable();
public void reduce (Text key, Iterable <IntWritable> values,
  Context context) throws IOException, InterruptedException
{
  int valueSum= 0;
for (IntWritable val: values) {
    valueSum += val.get();
  }
  count.set (valueSum);
  context.write (key, count);
}
}
```

Main Code-

```
package com.projectgurukul.wc;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
```

POORNIMA

```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInput-
                    Format;
import org.apache.hadoop.mapreduce.lib.output.FileOutput-
                    Format;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
 public static void main (String[] args) throws Exception
 {
Configuration conf = new Configuration();
String[] pathArgs = new GenericOptionsParser(conf, args).get-
                    RemainingArgs();
if (pathArgs.length < 2) {
 System.err.println("MR Project Usage: wordcount <input-
   path> [....] <output-path>");
 System.exit(2);
}
Job wcJob = Job.getInstance(conf, "MapReduce WordCount");
wcJob.setJarByClass(WordCount.class);
wcJob.setMapperClass(WordCountMapper.class);
wcJob.setCombinerClass(WordCountReducer.class);
wcJob.setReducerClass(WordCountReducer.class);
wcJob.setOutputKeyClass(Text.class);
wcJob.setOutputValueClass(IntWritable.class);
for (int i = 0; i < pathArgs.length - 1; ++i)
 {
 FileInputFormat.addInputPath(wcJob, new Path(pathArgs[i]));
}
```

Page No. 23

POORNIMA

```
FileOutputFormat.setOutputPath(wcJob, new Path(pathArgs
                [pathArgs.length-1]));
System.exit(wcJob.waitForCompletion(true)?0:1);
}
}
```

EXPERIMENT - 6

POORNIMA

Objective:- Implement Matrix Multiplication with Hadoop Map Reduce.

Mapper File:-

```
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
public class Map extends org.apache.hadoop.mapreduce.
Mapper<LongWritable, Text, Text, Text>{
 public void map (LongWritable key, Text value, Context context)
     throws IOException, InterruptedException {
 Configuration conf= context.getConfiguration();
 int m = Integer.parseInt(conf.get("m"));
 int p=Integer.parseInt(conf.get("p"));
 String line = value.toString();
 String[] indicesAndValue = line.split(",");
 Text outputkey = new Text();
 Test outputkey= new Text();
 if (indicesAndValue[0].equals("M")){
     for (int k=0; k<p; k++){
       outputkey.set(indicesAndValue[1]+","+k);
   outputValue.set(indicesAndValue[0]+","+indicesAndVal
       [2]+","+indicesAndValue[3]);
   context.write(outputkey, outputValue);
 }
 }else{
   for(int i=0;i<m;i++){
```

```
        outputkey.set(i+","+indicesAndValue[2]);
        outputValue.set("N",+indicesAndValue[1]+","+indicesAndValue
                                                [3]);
        context.write(outputkey, outputValue);
      }
    }
  }
```

Reducer file-

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.IOException;
import java.util.HashMap;
public class Reduce extends org.apache.hadoop.mapreduce.Reducer
        <Text, Text, Text, Text> {
public void reduce (Text key, Iterable <Text> values, Context context)
        throws IOException, InterruptedException {
  String [] value;
HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();
  for (Text val: values) {
    value = val.toString().split(",");
      if (value[0].equals("M")) {
hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]
                                                    ));
} else {
hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]))
  }
}
```

POORNIMA

```
int n = Integer.parseInt(context.getConfiguration().get("n"));
float result = 0.0f;
float m_ij;
float n_jk;
for (int j=0; j<n; j++) {
    m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
    n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
    result += m_ij * n_jk;
}
if (result != 0.0f) {
    context.write(null, new Text(key.toString() + "," + Float.
            toString(result)));
} } }
```

MatrixMultiply.java

```
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
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;
public class MatrixMultiply {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MatrixMultiply <in_dir> <out_dir
                    >");
            System.exit(2);
        }
```

```
Configuration conf = new Configuration();
conf.set("m", "1000");
conf.set("n", "100");
conf.set("p", "1000");
Job job = new Job(conf, "MatrixMultiply");
job.setJarByClass(MatrixMultiply.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(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);
}
}
```

POORNIMA

EXPERIMENT-7

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

Apache Pig:- is a data manipulation tool that is built over Hadoop's MapReduce. Pig provides us with a scripting language for easier and faster data manipulation. This scripting language to called Pig Latin.

Apache Pig scripts can be executed in 3 ways as follows:-

i) Using Grunt Shell (Interactive Mode):- Write the commands in the grunt shell and get the output there itself using the DUMP commands.

ii) Using Pig Scripts (Batch Mode):- Write the pig latin commands in a single file with .pig extension and execute the script on the prompt.

iii) Using User-Defined Functions (Embedded Mode):- Write your own functions on languages like Java and then use them in the scripts.

Pig Installation:- ~~Before~~ Before proceeding you need to make sure that you have all these pre-requisites as follows.

Hadoop Ecosystem installed on your system and all the four components i.e., DataNode, NameNode, ResourceManager, TaskManager, are working. If any one of them randomly shuts down then you need to fix that before proceeding.

7-zip is required to extract the .tar.gz files in windows.

Step 1:- Download the Pig version 0.17.0 tar file from the official Apache pig site.

Download the file 'pig-0.17.0.tar.gz' from the website and Install

Step 2:- Add the path variables of PIG-HOME and PIG-HOME\bin

Click the windows Button and ~~search~~ in search bar, type "Environment Variables". Then click on the "Edit the system environment variables".

Variable Name = PIG_HOME
Variable Value = C:\pig-0.17.0

Step-3:- Correcting the Pig Command File
Find file 'pig.cmd' in the bin folder of the pig file (C:\pig-0.17.0\bin)
set HADOOP_BIN_PATH = %HADOOP_HOME%\bin
Find the line:-
set HADOOP_BIN_PATH = %HADOOP_HOME%\bin
Replace the line:-
set HADOOP_BIN_PATH = %HADOOP_HOME%\libexec.
And save this file. We are finally here. Now you are all set to start exploring Pig and it's environment.
There are 2 ways of Invoking the grunt shell:-
Local mode:- All the files are installed, accessed, and run in the local machine itself. No need to use HDFS. The command for running Pig in local mode is as follows.

pig -x local.

Mapreduce Mode:- The files are all present on the HDFS. We need to load this data to process it. The command for running Pig in MapReduce / HDFS Mode is as follows.

pig -x mapreduce.

POORNIMA

EXPERIMENT-8

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

Hive is a data warehouse software project built on top of Hadoop, that facilitate reading, writing, and managing large datasets residing in distributed storage using SQL.

Download Hive 2.1.0

https://archive.apache.org/dist/hive/hive-2.1.0/

Download hive-site.xml

https://github.com/apache/hive/blob/master/data/conf/hive-site.xml

Step-1:- Extract the Hive file.

Extract file apache-hive-2.1.0-bin.tar.gz and place under "D:\Hive", you can use any preferred location.

i) You will get again a tar file post extraction:-

apache-hive-2.1.0-bin.tar

ii) Go inside of apache-hive-2.1.0-bin.tar folder and extract again

iii) apache-hive-2.1.0-bin

iv) Copy the leaf folder "apache-hive-2.1.0-bin" and move to the root folder "D:\Hive" and removed all other files and folders:-

bin

Step-2:- Extract the Derby file.

Similar to Hive, extract file db-derby-10.12.1.1-bin.tar.gz and place under "D:\Derby", you can use any preferred location:-

db-derby-10.12.1.1-bin

Step-3:- Moving hive-site.xml file

Drop the downloaded file "hive-site.xml" to hive configuration location "D:\Hive\apache-hive-2.1.0-bin\conf".

POORNIMA

Step-3:- Moving Derby libraries.

Next, need to drop all derby library to hive library location:-

i) Move to library folder under derby location D:\Derby\db-derby-10.12.1.1-bin\lib.

ii) Select all and copy all libraries.

iii) Move to library folder under hive location D:\Hive\apache-hive-2.1.0-bin\lib.

Step-4:- Drop all selected libraries here.

Step-5:- Configure Environment variables.

Set the path for the following Environment variables (User Variable) on windows 10 -

a) HIVE_HOME=D:\Hive\apache-hive-2.1.0-bin

b) HIVE-BIN-D:\Hive\apache-hive-2.1.0-bin\bin

c) HIVE-LIB-D:\Hive\apache-hive-2.1.0-bin\lib.

d) DERBY_HOME-D:\Derby\db-derby-10.12.1.1-bin

e) HADOOP_USER_CLASSPATH_FIRST.-true

Step-6:- Configure System variables.

Next onward need to set System variable, including Hive bin directory path:-

a) HADOOP-USER-CLASSPATH-FIRST-true

b) Variable: Path

c) Value:

d) D:\Hive\apache-hive-2.1.0-bin\bin

e) D:\Derby\db-derby-10.12.1.1-bin\bin

Edit file D:/Hive/apache-hive-2.1.0-bin/conf/hive-site.xml, paste below xml paragraph and save this file.

POORNIMA

hive-site.xml

<configuration>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:derby://localhost:1527/metastore_db;create=true</value>
<description>JDBC connect string for a JDBC metastore</description>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>org.apache.derby.jdbc.ClientDriver</value>
~~<description>JDBC connect string for a JDBC metastore</description>~~
<description>Driver class name for a JDBC metastore</description>
</property>
<property>
<name>hive.server2.enable.impersonation</name>
<description>Enable user impersonation for HiveServer2</description>
<value>true</value>
</property>
<property>
<name>hive.server2.authentication</name>
<value>NONE</value>
<description>Client authentication types. NONE: no authentication check
LDAP. LDAP/AD based authentication KERBEROS. Kerberos/GSSAPI
authentication CUSTOM: Custom authentication provider (Use with property
hive.server2.custom.authentication.class)</description>
</property>
<property>
<name>datanucleus.autoCreateTables</name>

<value>True</value>
</property>
</configuration>

Open command prompt and change directory to "D:\Hadoop\hadoop-2.8.0\sbin" and type "start-all.cmd" to start apache

D:\Hadoop\hadoop-2.8.0\sbin>start-all.cmd.

It will open four instances of cmd for following tasks:-

i) Hadoop Datanode
ii) Hadoop Namenode
iii) Yarn Nodemanager
iv) Yarn Resourcemanager.

It can be verified via browser also as-

Namenode (hdfs)- http://localhost:50070

Datanode - http://localhost:50075

All Application (cluster) - http://localhost:8088 etc.

Since the "start-all.cmd" command has been deprecated so you can use below command in order wise:-

i) "start-dfs.cmd" and
ii) "start-yarn.cmd"

Step-7: Start Derby Server

Post successful execution of Hadoop, change directory to "D:\Derby\db-derby-10.12.1.1-bin\bin" and type "startNetworkServer -h 0.0.0.0" to start derby server.

Step-8:- Start the Hive.

Derby server has been started and ready to accept connection so open

POORNIMA

a new command prompt under administrator privileges and move to hive directory "D:\Hive\apache-hive-2.1.0-bin\bin"-

i) Type "jps-m" to check Network Server Control.

ii) Type "hive" to execute hive server.

Step-II:- Some hands on activities.

i) Create database in Hive:-

```
CREATE DATABASE IF NOT EXISTS TRAINING
```

ii) Show Database:-

```
SHOW DATABASES;
```

iii) Creating Hive Tables:-

```
CREATE TABLE IF NOT EXISTS testhive (col1 char(10), col2 char(20));
```

output (Linked List)

Linked list : [Cat, Dog, Horse]
first Element : Cat
Last Element : Horse

output (stack)

stack : [Dog, Horse, Cat]
stack after pop : [Dog, Horse]

POORNIMA

Experiment : 1

Objective:- Implement the following Data structure in Java.

i) Linked list :- A linked list is a data structure whe the objects are arranged in a linear order.

Program:-

```
import java.util.Linkedlist;
class Main {
  public Static Void main (String [] args) {
  Linkedlist <String> animals = new Linkedlist <>();
  animals.add ("dog");
  animals.addfirst ("Cat");
  animals.addlast ("Horse");
  System.out.println ("Linkedlist:" + animals);
  System.out.println ("first Element:" + animals.getfirst());
  System.out.println ("last Element:" + animals.getlast());
  }
}
```

ii) Stack - A stack is a linear data structure that follows the principle of last in first out [LIFO]

Program:-

```
import java.util.stack;
class Main {
  public static Void main ( String [] args) {
    stack <String> animals = new stack <>();
```

```
Stack <String> animals = new Stack<>();
animals.push ("Dog");
animals.push ("Horse");
animals.push ("Cat");
System.out.println ("stack:" + animals);
animals.pop();

System.out.println ("stack after pop:" + animals);
}
}
```

iii) Queues : A Queue is a linear structure which follows a particular order in which the operation are performed

The order of first In first out (FIFO)

Program :-

```
import java.util.Queue;
import java.util.LinkedList;

class Main {
  public Static Void main (String [] args) {
    Queue < Integer > number = new LinkedList <> ();
    numbers.offer (1);
    numbers.offer (2);
    numbers.offer (3);
```

```
System.out.println (" Queue" + numbers);
int removed Number = number.poll();
System.out.println (" Removed Elements." + removed Number);
System.out.println (" Queue after deletion:" + number);
}
}
```

iv) **Set:** A set data structure allows to add data to a container.

Program:

```
import java.util.*;
public class Main {
  public Static Void main (String[] args)
  {
set <String> Data = new LinkedHashSet < String > ();
  data.add ("Dog");
  data.add ("Cat");
  data.add ("Horse");
  data.add ("Cat");
  System.out.println (Data);
}
}
```

v) **Map:** Map is a type of Data structure to multiple dictionaries together as one unit.

Program:

POORNIMA

```
import java.util.Map;
import java.util.HashMap;

class Main{
    public static void main (String[] args){
    Map <String Integer> numbers = new HashMap <>();
    numbers.put ("one", 1);
    numbers.put ("Two", 2);
    System.out.println ("map" + numbers);
    System.out.println ("key:" + numbers.keyset());
    System.out.println ("Values" + numbes.Values());
    System.out.println ("Entries" + numbas.entrySet());
      int Value = numbers.remove ("two");
      System.out.println ("Removed Value:" + Value);
  }
}
```

POORNIMA

Experiment - 2

Obj:- Perform setting up and Installing Hadoop in its three operating modes. Standalone, Pseudo Distributed & full Distributed.

1. Standalone Mode:- In standalone mode none of the demon will run ie. Name Node, Data node, Secondary Name Node, Job Tracker, Task Tracker.

we use Job Tracker and Task Tracker for processing purpose in Hadoop for Hadoop we use Resource Manager and Node Manager.

↳ Standalone Mode also means that we are installing Hadoop only in a single system. By default Hadoop is mode to run in this standalone Mode are we can also call it as local mode

↳ We mainly use Hadoop in this mode for the purpose of learning, testing & debugging.

↳ Steps for Installation

i) Install Java
ii) Set Java - Home Variable
iii) Install spark
iv) Set spark - Home Variable
v) Download windows utilities for Hadoop

vi) Set hadoop home Variable
vii) Run spark as spark-shell in the Cmd

2 Pseudo Distributed Mode:- In the Pseudo-distributed mode we also only a single node, But the main Thing is that the cluster is Simulated, which means that all the process inside the cluster will run independently to each other. All the daemos that are NameNode, Datanode, Secondary Name Node, Resource Manager Node, nanager etc. will be running as a seperate process on seperate JVM. (Java Virtual Machine) or we can say that Run on Different java process that is why it is called as pseudo-distributed.

⇒ Steps for Installation:

i) Download the latest binary package from ("http:// hadoop
ii) unzip the binary package
iii) Create folders for datanode & namenode, Also set Hadoop environment Variable
iv) Make Java home path
v) Configure Hadoop Configuration files
vi) Format Name node
vii) Launch Hadoop on Cmd

POORNIMA

3) Fully-distributed Mode:- This is most important one in which nodes are used few of them run the master's daemon that are Namenode and resource manager and the rest of them run the slave daemons that are Datanode & node manager.

In fully Distributed NameNode run on the Master Node, the Daemons DataNode & Task Tracker run on the slave node

POORNIMA

Experiment-3

obj:- Implement the following file management Tasks in Hadoop:

i) Adding files and directories
ii) Retriving files
iii) Deleting files

Steps for files managment in Hadoop

i) Adding files and Directories

⇒ Listing files in HDFS

After loading the information in the Server, we can find the list of files in a Directory, status of a file using "ls"

Given below is the syntax of ls that you can pass to a Directory or filesname as an argument

$$ HADOOP_HOME/bin/hadoop fs -ls

→ Inserting Data into HDFS

step1 - Create an input directory

$$ HADOOP_HOME/bin/hadoop fs -Mkdir /user/input

step2:- Transfer & store a Data files from local system to Hadoop file System

using the "put" Command

$$ HADOOP_HOME /bin/hadoop fs -put /home/ file.txt /user/input

Stp3: you can verify the files using ls Command

$$ HADOOP_HOME/bin/ hadoop fs -ls / user/input.

ii) Retriving files:

Assume we have a file in HDFS called outfile. This file can be retrived from the hadoop file system as follows

Stp1: View the Data using Cat Command

$$ HADOOP_HOME/bin/hadoop fs -cat /user/output/outfile

Stp2: use "get" to bring the file to local file system

$$ HADOOP_HOME/bin/ hadoop fs -get /user/output/ hadoop fs

iii) Deleting files:

To remove an existing file

hdfs dfs -rm -r

hdfs: // path/to/ file