

# NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY



## BIG DATA ANALYTICS LAB FILE (CICPC19)

**PREPARED BY:**  
Sudharma Jain  
2021UCI8066  
CSIOT (6<sup>TH</sup> SEM)

**SUBMITTED TO:**  
Mr. R.S. Raw

## INDEX

| S.NO | TOPIC   | DATE | SIGN |
|------|---|------|------|
| 1.   | Installation of VMWare to setup the Hadoop environment and its ecosystems   |      |      |
| 2.   | Perform setting up and installing of Hadoop in 3 modes: standalone, distributed and pseudo distributed modes.                           |      |      |
| 3.   | Implement the following file management tasks in Hadoop:<br>1. Adding files and directories<br>2. Retrieving files<br>3. Deleting Files |      |      |
| 4.   | Run a basic Word Count Map Reduce Program to understand Map Reduce Paradigm   |      |      |
| 5.   | Write a Map Reduce Program that mines Weather Data  |      |      |
| 6.   | Write a Map Reduce program that implements Matrix Multiplication.   |      |      |
| 7.   | Install PIG. Write Pig Latin scripts sort, group, join, project, and filter your data. Run the Pig Latin Scripts to find Word Count.    |      |      |
| 8.   | Install HIVE. Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.                                     |      |      |
| 9.   | Install MongoDB. Create database in MongoDB.  |      |      |
| 10.  | Write R programs for various functions of Array, Matrix and Factor  |      |      |

## EXPERIMENT-1

### AIM:

Installation of VMWare to setup the Hadoop environment and its ecosystems.

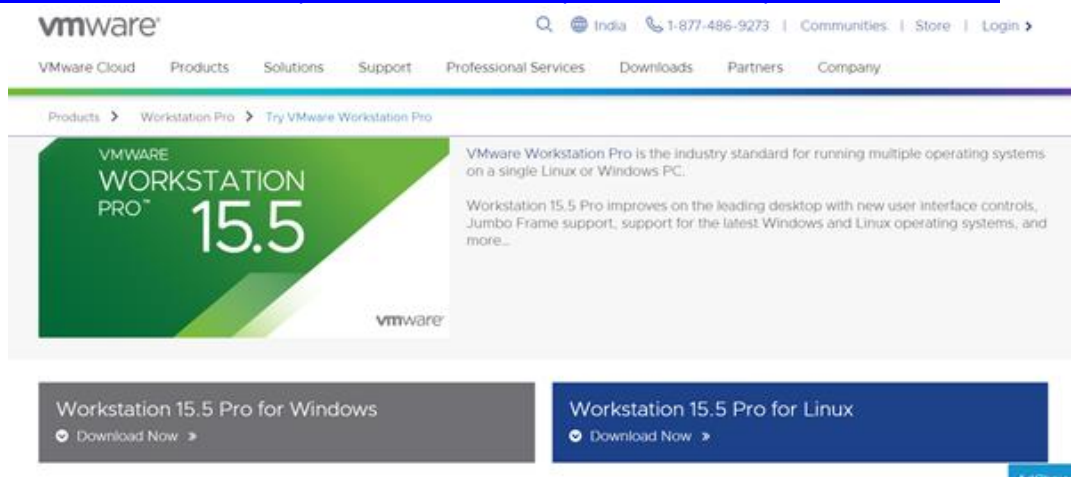
### THEORY:

VMware develops virtualization Software. Virtualization software creates an abstraction layer over computer hardware that allows the hardware elements of a single computer — processors, memory, storage, and more — to be divided into multiple virtual computers, commonly called virtual machines (VMs). Each virtual machine runs its own operating system (OS) and behaves like an independent computer, even though it is running on a portion of the actual underlying computer hardware. A VM is a software-based representation of a physical computer. An operating system (OS) running in a VM is called a guest OS.

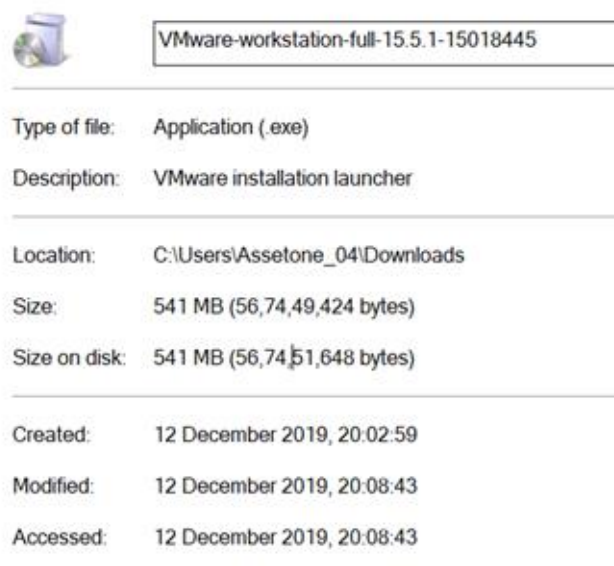
### Setup and Installation:

1. Installing VMware Workstation from given below link. There are two options for downloading one is Windows and other for Linux. My Base Operating System is Windows8, So I choose for VMware for Windows. If Your Base OS is Linux go and choose VMware for Linux Link.

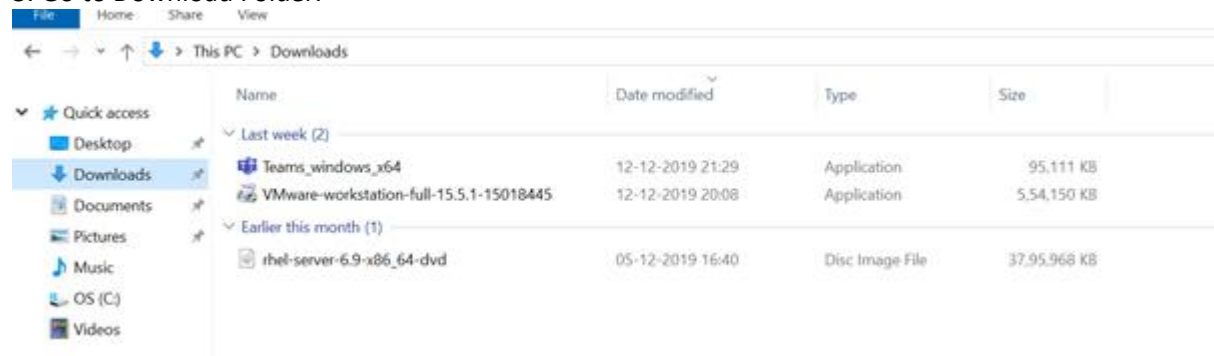
<https://www.vmware.com/in/products/workstation-pro/workstation-pro-evaluation.html>



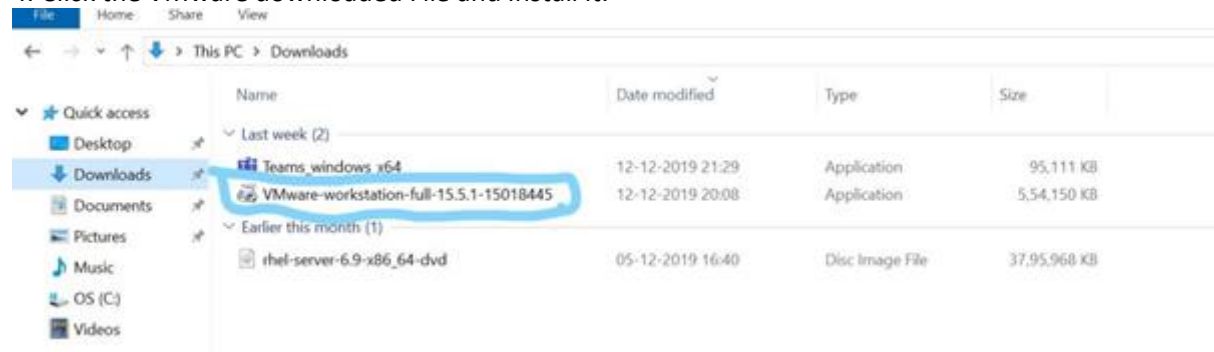
2. Check your VMware Properties.



3. Go to Download Folder.



4. Click the VMware downloaded File and Install it.



5. Click on VMware Software and click and choose “Pin to Taskbar”.

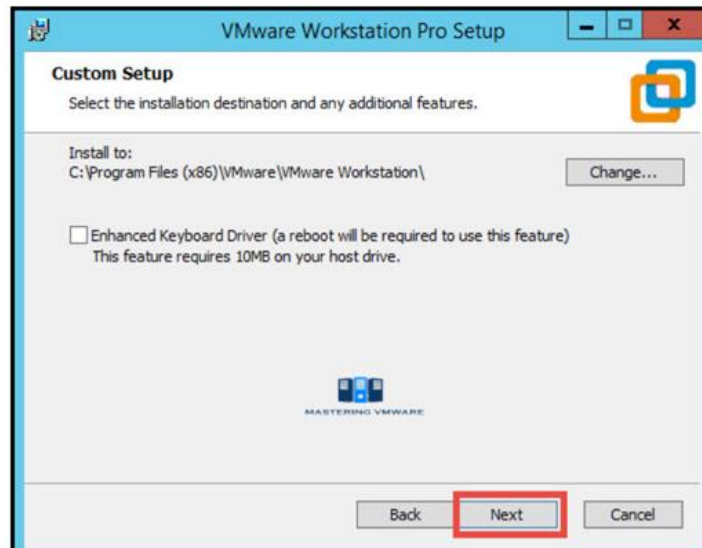
6. Click on VMware Software and Click on Next to the Installation wizard.



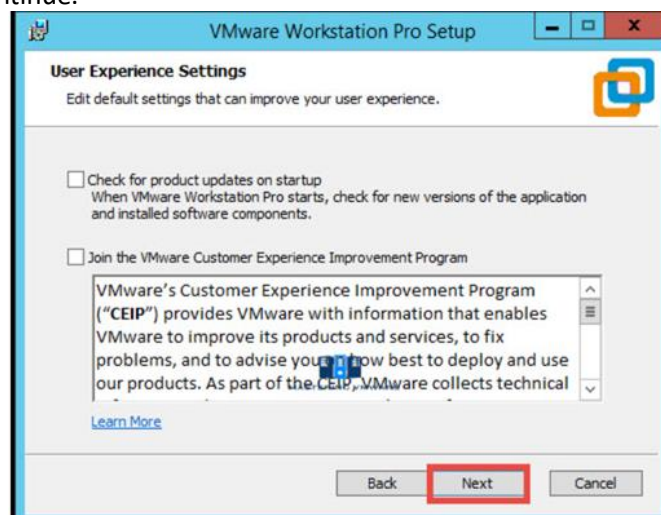
7. Read and Accept the VMware End User license agreement. Click Next to Continue.



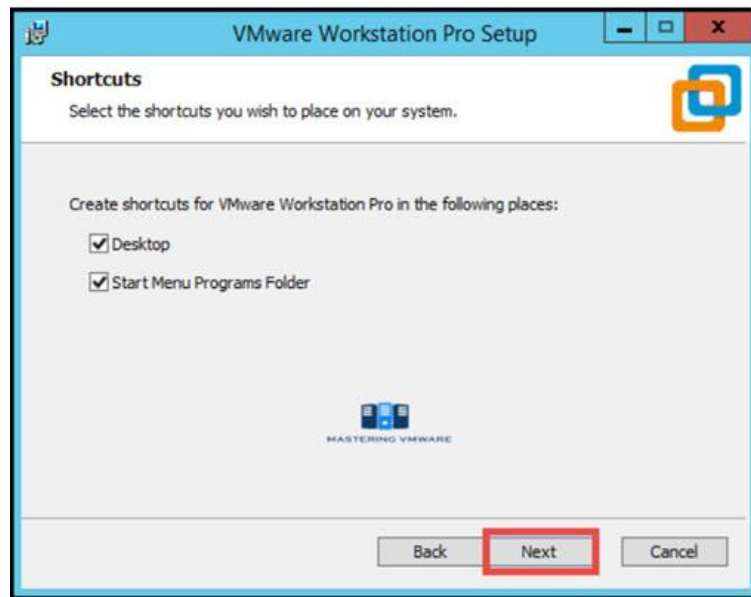
8. Specify the Installation directory. You can also enable Enhance keyboard driver here. Click Next to continue.



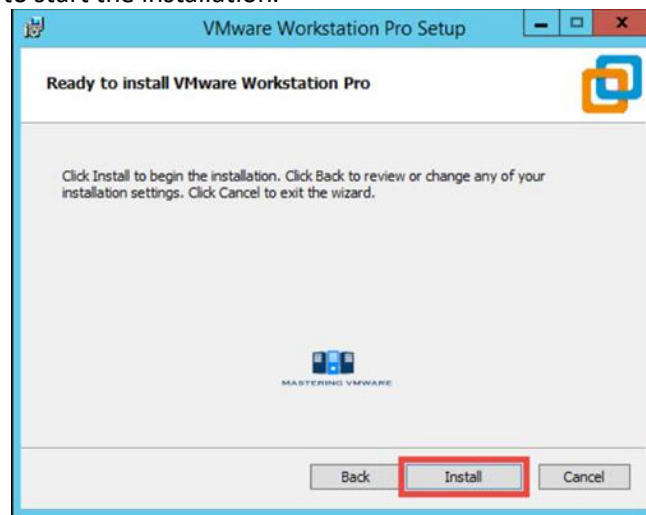
9. You can enable product startup and join the VMware Customer experience Improvement program here. Click Next to Continue.



10. Select the shortcuts you want to create for easy access to VMware Workstation. Click Next to Continue.



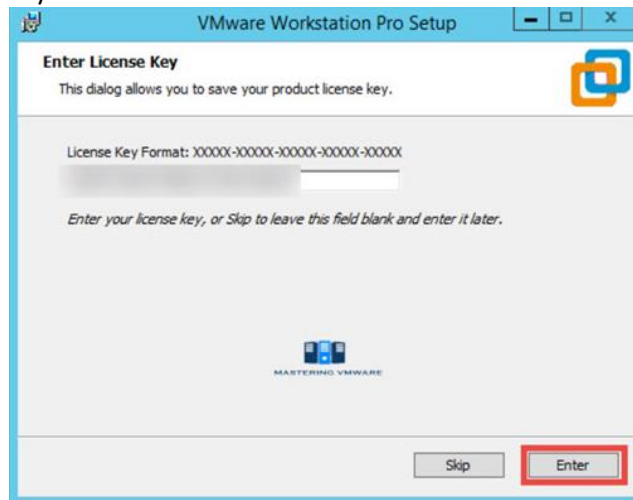
11. Click Install button to start the installation.



12. Installation will take just few seconds to complete. If you have license-key then click on License to enter the license or you can also click Finish to exit the Installer.



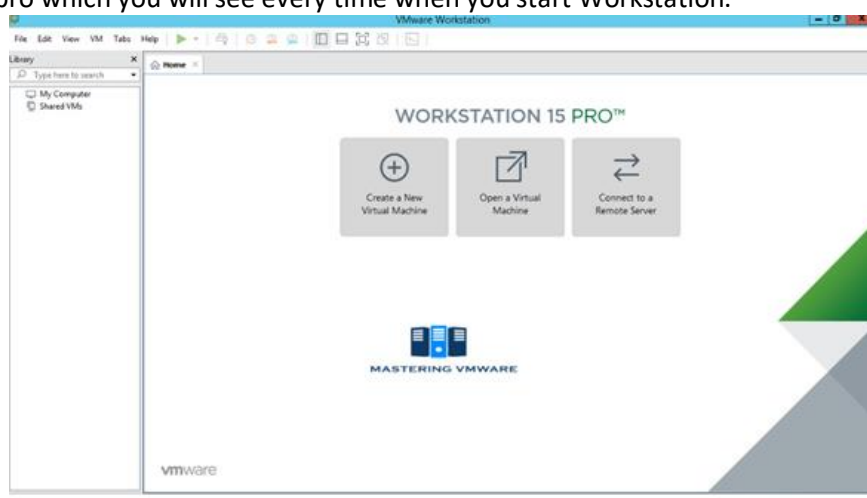
13. Provide the License Key for VMware Workstation Pro. Press Enter to continue.



14. Click Finish to exit the wizard.



15. That's it we have successfully installed VMware Workstation Pro. Now you can start the VMware Workstation Pro by clicking on the shortcut on Desktop. Below is the home screen of the VMware Workstation pro which you will see every time when you start Workstation.



## EXPERIMENT-2

### AIM:

Perform setting up and installing of Hadoop in 3 modes: standalone, distributed and pseudo distributed modes.

### DESCRIPTION:

Hadoop is written in java, so you will need to have java installed on your system version 6 or later. Sun's is the one most widely used with Hadoop, although others have been reported to work. Hadoop runs on unix and on windows. Linux is the only supported production platform, but other flavours of unix (including mac-OS x) can be used to run Hadoop for development. Windows is only supported as a development platform and requires Cygwin to run during the Cygwin installation process. You should include the, openSSH package if you plan to run Hadoop in pseudo distributed mode.

### ALGORITHM:

#### STEPS INVOLVED IN STANDALONE MODE:

1. Command for installing SSH is  
***sudo apt-get install ssh.***
2. Command for key generation is  
***ssh-keygen -t rsa -P ''***
3. Store the key into ***rsa.pub*** by using the command  
***cat \$HOME/.ssh/id\_rsa.pub >>> \$HOME/.ssh/authorized\_keys***
4. Extract the java by using the command  
***tar xvfz jdk-8u60-linux-i586.tar.gz***
5. Extract the eclipse by using the command  
***tar xvfz eclipse-jee-mars-R-linux-gtk.tar.gz***
6. Extract the Hadoop by using the command  
***tar xvfz hadoop-2.7.1.tar.gz***
7. Move the java to ***/usr/lib/jvm/*** and eclipse to ***/opt/path***. Configure the java path in ***eclipse.ini*** file
8. Export the java path in ***/bashrc*** by checking the java version and Hadoop version
9. Check the Hadoop instance in standalone mode working correctly or not by using an implicit Hadoop jar file.
10. If the path is displayed correctly in ***part-r-00000*** this means that standalone mode is installed successfully

### ALGORITHM:

#### STEPS FOR PSEUDO DISTRIBUTED MODE:

1. In order install pseudo distributed mode we need to configure the hadoop configuration files resides in the directory ***/home/lendi/hadoop-2.7.1/etc/hadoop***.
2. First configure the ***hadoop-env.sh*** file by changing the java path.
3. Configure the ***core-site.xml*** which contains a property tag, it contains name and value. Name as ***fs.defaultFS*** and value as ***hdfs://localhost:9000***
4. Configure ***hdfs-site.xml***.
5. Configure ***yarn-site.xml***.
6. Configure ***mapred-site.xml*** before configure the copy mapred-site.xml.template to mapred-site.xml.
7. Now format the name node by using command



***hdfs namenode -format.***

8. Type the command ***start-dfs.sh, start-yarn.sh*** means that starts the daemons like NameNode, DataNode, SecondaryNameNode, ResourceManager, NodeManager.
9. Run JPS which views all daemons. Create a directory in the hadoop by using command  
***hdfs dfs -mkdir /csedir***  
and enter some data into ***lendi.txt*** using command  
***nano lendi.txt***  
and copy from local directory to hadoop using command  
***hdfs dfs -copyFromLocal lendi.txt /csedir/***  
and run sample jar file wordcount to check whether pseudo distributed mode is working or not.
10. Display the contents of file by using command  
***hdfs dfs -cat /newdir/part-r-00000.***

#### **ALGORITHM:**

##### **FULLY DISTRIBUTED MODE INSTALLATION:**

1. Stop all single node clusters  
***\$stop-all.sh***
2. Decide one as NameNode (Master) and remaining as DataNodes (Slaves).
3. Copy public key to all three hosts to get a password less SSH access  
***\$ssh-copy-id -I \$HOME/.ssh/id\_rsa.pub lendi@l5sys24***
4. Configure all Configuration files, to name Master and Slave Nodes.  
***\$cd \$HADOOP\_HOME/etc/hadoop***  
***\$nano core-site.xml***  
***\$ nano hdfs-site.xml***
5. Add hostnames to file slaves and save it.  
***\$ nano slaves***
6. Configure ***\$ nano yarn-site.xml***
7. Do in Master Node  
***\$ hdfs namenode -format***  
***\$ start-dfs.sh***  
***\$start-yarn.sh***
8. Format NameNode
9. Daemons Starting in Master and Slave Nodes
10. **END**

#### **INPUT**

ubuntu @localhost> jps

#### **OUTPUT:**

Data node, name node, Secondary name node,  
NodeManager, Resource Manager

## EXPERIMENT-3

### AIM:

Implement the following file management tasks in Hadoop:

1. Adding files and directories
2. Retrieving files
3. Deleting Files

### DESCRIPTION:

HDFS is a scalable distributed filesystem designed to scale to petabytes of data while running on top of the underlying filesystem of the operating system. HDFS keeps track of where the data resides in a network by associating the name of its rack (or network switch) with the dataset. This allows Hadoop to efficiently schedule tasks to those nodes that contain data, or which are nearest to it, optimizing bandwidth utilization. Hadoop provides a set of command line utilities that work similarly to the Linux file commands, and serve as your primary interface with HDFS. We are going to have a look into HDFS by interacting with it from the command line. We will take a look at the most common file management tasks in Hadoop, which include:

1. Adding files and directories to HDFS
2. Retrieving files from HDFS to local filesystem
3. Deleting files from HDFS

### ALGORITHM:

#### SYNTAX AND COMMANDS TO ADD, RETRIEVE AND DELETE DATA FROM HDFS

#### STEP-1

##### Adding Files and Directories to HDFS

Before you can run Hadoop programs on data stored in HDFS, you 'll need to put the data into HDFS first. Let's create a directory and put a file in it. HDFS has a default working directory of */user/\$USER*, where *\$USER* is your login user name. This directory is not automatically created for you, though, so let 's create it with the *mkdir* command. For the purpose of illustration, we use chuck. You should substitute your user name in the example commands.

```
hadoop fs -mkdir /user/chuck  
hadoop fs -put example.txt  
hadoop fs -put example.txt /user/chuck
```

#### STEP-2

##### Retrieving Files from HDFS

The Hadoop command *get* copies files from HDFS back to the local filesystem. To retrieve *example.txt*, we can run the following command:

```
hadoop fs -cat example.txt
```

#### STEP-3

##### Deleting Files from HDFS

```
hadoop fs -rm example.txt
```

Command for creating a directory in hdfs is "*hdfs dfs -mkdir /lendicse*".

Adding directory is done through the command "*hdfs dfs -put lendi\_english /*".

#### STEP-4

#### Copying Data from NFS to HDFS

Copying from directory command is

*"hdfs dfs -copyFromLocal/home/lendi/Desktop/shakes/glossary /lendicse/"*

View the file by using the command *"hdfs dfs -cat /lendi\_english/glossary"*

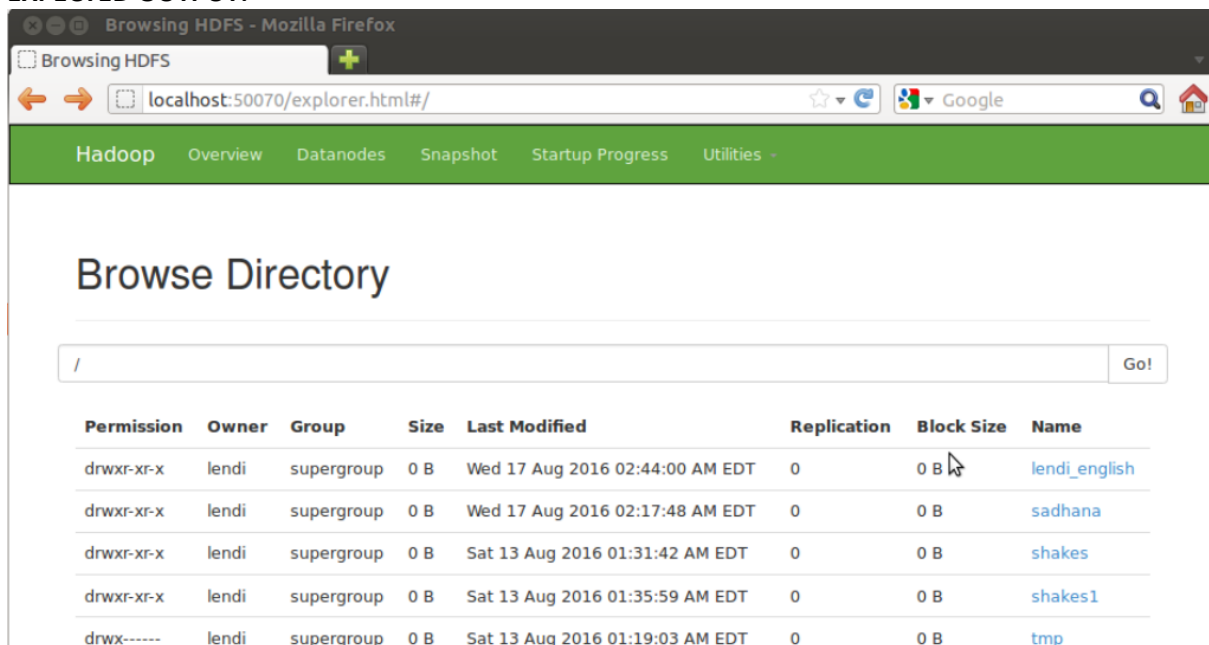
Command for listing of items in Hadoop is *"hdfs dfs -ls hdfs://localhost:9000/"*.

Command for Deleting files is *"hdfs dfs -rm r /kartheek"*.

#### SAMPLE INPUT:

Input as any data format of type structured, Unstructured or Semi Structured

#### EXPECTED OUTPUT:



| Permission | Owner | Group      | Size | Last Modified                   | Replication | Block Size | Name                          |
|------------|-------|------------|------|---------------------------------|-------------|------------|-------------------------------|
| drwxr-xr-x | lendi | supergroup | 0 B  | Wed 17 Aug 2016 02:44:00 AM EDT | 0           | 0 B        | <a href="#">lendi_english</a> |
| drwxr-xr-x | lendi | supergroup | 0 B  | Wed 17 Aug 2016 02:17:48 AM EDT | 0           | 0 B        | <a href="#">sadhana</a>       |
| drwxr-xr-x | lendi | supergroup | 0 B  | Sat 13 Aug 2016 01:31:42 AM EDT | 0           | 0 B        | <a href="#">shakes</a>        |
| drwxr-xr-x | lendi | supergroup | 0 B  | Sat 13 Aug 2016 01:35:59 AM EDT | 0           | 0 B        | <a href="#">shakes1</a>       |
| drwx-----  | lendi | supergroup | 0 B  | Sat 13 Aug 2016 01:19:03 AM EDT | 0           | 0 B        | <a href="#">tmp</a>           |

## EXPERIMENT 4

### AIM:

Run a basic Word Count Map Reduce Program to understand Map Reduce Paradigm

### DESCRIPTION:

MapReduce is the heart of Hadoop. It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster.

The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions. The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

### ALGORITHM:

#### MAPREDUCE PROGRAM

WordCount is a simple program which counts the number of occurrences of each word in a given text input data set. WordCount fits very well with the MapReduce programming model making it a great example to understand the Hadoop Map/Reduce programming style. Our implementation consists of three main parts:

1. Mapper
2. Reducer
3. Driver

#### Step-1. Write a Mapper

A Mapper overrides the "map" function from the Class:

```
"org.apache.hadoop.mapreduce.Mapper"
```

which provides <key, value> pairs as the input.

A Mapper implementation may output <key,value> pairs using the provided Context .

Input value of the WordCount Map task will be a line of text from the input data file and the key would be the line number <line\_number, line\_of\_text> . Map task outputs <word, one> for each word in the line of text.

#### Pseudo-code:

```
void Map (key, value) {  
    for each word x in value:  
        output.collect(x, 1);  
}
```

#### Step-2. Write a Reducer

A Reducer collects the intermediate <key,value> output from multiple map tasks and assemble a single result. Here, the WordCount program will sum up the occurrence of each word to pairs as <word, occurrence>.

#### Pseudo-code:

```
void Reduce (keyword, <list of value>) {  
    for each x in <list of value>:  
        sum+=x;
```

```

    final_output.collect(keyword, sum);
}

```

### Step-3. Write Driver

The Driver program configures and run the MapReduce job. We use the main program to perform basic configurations such as:

- Job Name: name of this Job
- Executable (Jar) Class: the main executable class. For here, WordCount.
- Mapper Class: class which overrides the "map" function. For here, Map.
- Reducer: class which override the "reduce" function. For here, Reduce.
- Output Key: type of output key. For here, Text.
- Output Value: type of output value. For here, IntWritable.
- File Input Path
- File Output Path

### INPUT:

Set of Data Related Shakespeare Comedies, Glossary, Poems

### OUTPUT:

```

lendi@ubuntu: ~/Desktop
16/08/17 01:17:45 INFO impl.YarnClientImpl: Submitted application application_1471410736896_0001
16/08/17 01:17:45 INFO mapreduce.Job: The url to track the job: http://ubuntu.ubuntu-domain:8088/proxy/application_1471410736896_0001/
16/08/17 01:17:45 INFO mapreduce.Job: Running job: job_1471410736896_0001
16/08/17 01:17:52 INFO mapreduce.Job: Job job_1471410736896_0001 running in uber mode : false
16/08/17 01:17:52 INFO mapreduce.Job:  map 0% reduce 0%
16/08/17 01:17:59 INFO mapreduce.Job:  map 100% reduce 0%
16/08/17 01:18:06 INFO mapreduce.Job:  map 100% reduce 100%
16/08/17 01:18:06 INFO mapreduce.Job: Job job_1471410736896_0001 completed successfully
16/08/17 01:18:06 INFO mapreduce.Job: Counters: 49
File System Counters
  FILE: Number of bytes read=3772644
  FILE: Number of bytes written=7775215
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=1744718
  HDFS: Number of bytes written=510970
  HDFS: Number of read operations=6
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2

```

```

part-r-00000(3) (~/.Downloads) - gedit
2.      1
3.     28
3.      1
4.      1
5.      1
6.      1
7.      1
8.      1
9.      1
A      1012
A'      2
ADRIAN, 2
AEdiles, 1
AEsculapius? 1
ALARBUS, 1
ALENCON, 2
ALL'S 25
ANDRONICUS, 1
ANGELO, 2

```

## EXPERIMENT - 5

### AIM:

Write a Map Reduce Program that mines Weather Data

### DESCRIPTION:

Climate change has been seeking a lot of attention since long time. The antagonistic effect of this climate is being felt in every part of the earth. There are many examples for these, such as sea levels are rising, less rainfall, increase in humidity. The propose system overcomes some issues that occurred by using other techniques. In this project we use the concept of big data Hadoop. In the proposed architecture we are able to process offline data, which is stored in the National Climatic Data Centre (NCDC). Through this we are able to find out the maximum temperature and minimum temperature of year, and able to predict the future weather forecast. Finally, we plot the graph for the obtained MAX and MIN temperature for each moth of the particular year to visualize the temperature. Based on the previous year data weather data of coming year is predicted.

### ALGORITHM:

#### MAPREDUCE PROGRAM

WordCount is a simple program which counts the number of occurrences of each word in a given text input data set. WordCount fits very well with the MapReduce programming model making it a great example to understand the Hadoop/ MapReduce programming style. Our implementation consists of three main parts:

- Mapper
- Reducer
- Main program

#### Step-1. Write a Mapper

A Mapper overrides the `—map()` function from the Class "[org.apache.hadoop.mapreduce.Mapper](#)" which provides <key, value> pairs as the input. A Mapper implementation may output <key,value> pairs using the provided Context .

Input value of the WordCount Map task will be a line of text from the input data file and the key would be the line number <line\_number, line\_of\_text>. Map task outputs <word, one> for each word in the line of text.

#### Pseudo-code

```
void Map (key, value) {
    for each max_temp x in value:
        output.collect(x, 1);
}
void Map (key, value) {
    for each min_temp x in value:
        output.collect(x, 1);
}
```

#### Step-2 Write a Reducer

A Reducer collects the intermediate <key, value> output from multiple map tasks and assemble a single result. Here, the WordCount program will sum up the occurrence of each word to pairs as <word, occurrence>.

### Pseudo-code

```
void Reduce (max_temp, <list of value>) {  
    for each x in <list of value>:  
        sum+=x;  
        final_output.collect(max_temp, sum);  
}  
  
void Reduce (min_temp, <list of value>) {  
    for each x in <list of value>:  
        sum+=x;  
        final_output.collect (min_temp, sum);  
}
```

### Step -3 Write Driver

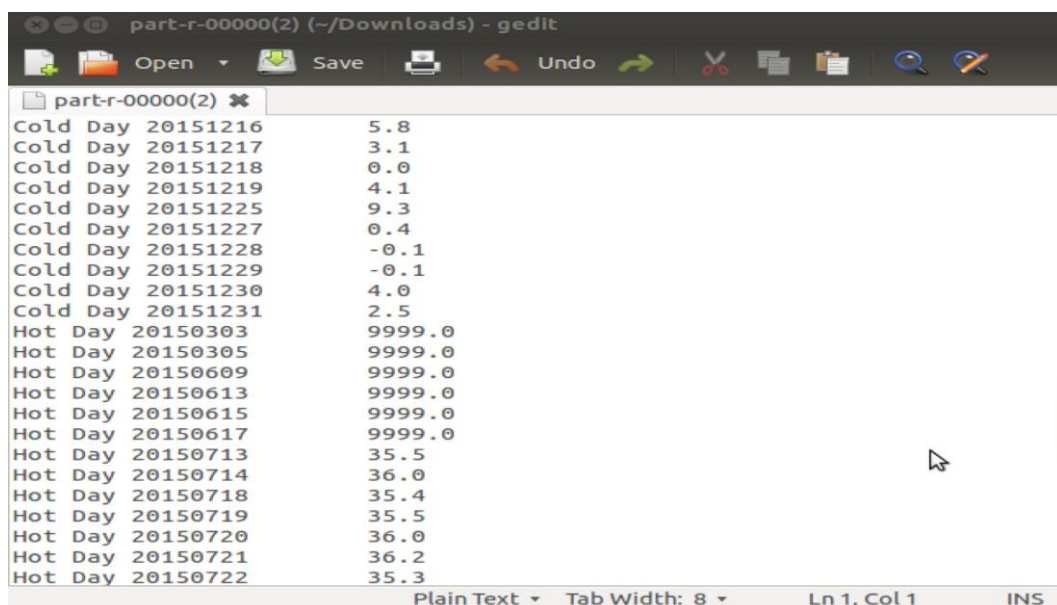
The Driver program configures and run the MapReduce job. We use the main program to perform basic configurations such as:

- Job Name: name of this Job
- Executable (Jar) Class: the main executable class. For here, WordCount.
- Mapper Class: class which overrides the "map" function. For here, Map.
- Reducer: class which override the "reduce" function. For here, Reduce.
- Output Key: type of output key. For here, Text.
- Output Value: type of output value. For here, IntWritable.
- File Input Path
- File Output Path

### INPUT:

Set of Weather Data over the years

### OUTPUT:



## EXPERIMENT - 6

### AIM:

Write a Map Reduce Program that implements Matrix Multiplication.

### DESCRIPTION:

We can represent a matrix as a relation (table) in RDBMS where each cell in the matrix can be represented as a record (i, j, value). As an example, let us consider the following matrix and its representation. It is important to understand that this relation is a very inefficient Relation if the matrix is dense. Let us say we have 5 Rows and 6 Columns, then we need to store only 30 values. But if you consider above relation, we are storing 30 rowid, 30 col\_id and 30 values in other sense we are tripling the data. So, a natural question arises why we need to store in this format? In practice most of the matrices are sparse matrices. In sparse matrices not all cells used to have any values, so we do not have to store those cells in DB. So, this turns out to be very efficient in storing such matrices.

### MapReduceLogic:

Logic is to send the calculation part of each output cell of the result matrix to a reducer. So in matrix multiplication the first cell of output (0,0) has multiplication and summation of elements from row 0 of the matrix A and elements from col 0 of matrix B. To do the computation of value in the output cell (0,0) of resultant matrix in a separate reducer we need to use (0,0) as output key of map phase and value should have array of values from row 0 of matrix A and column 0 of matrix B. Hopefully this picture will explain the point. So in this algorithm output from map phase should be having a <key,value> , where key represents the output cell location (0,0) , (0,1) etc.. and value will be list of all values required for reducer to do computation. Let us take an example for calculating value at output cell (0,0). Here we need to collect values from row 0 of matrix A and col 0 of matrix B in the map phase and pass (0,0) as key. So a single reducer can do the calculation.

### ALGORITHM:

We assume that the input files for A and B are streams of (key, value) pairs in sparse matrix format, where each key is a pair of indices (i, j) and each value is the corresponding matrix element value. The output files for matrix  $C=A*B$  are in the same format.

We have the following input parameters:

- The path of the input file or directory for matrix A.
- The path of the input file or directory for matrix B.
- The path of the directory for the output files for matrix C. Strategy = 1, 2, 3
- R = the number of reducers.
- I = the number of rows in A and C.
- K = the number of columns in A and rows in B.
- J = the number of columns in B and C.
- IB = the number of rows per A block and C block.
- KB = the number of columns per A block and rows per B block.
- JB = the number of columns per B block and C block.

In the pseudo-code for the individual strategies below, we have intentionally avoided factoring common code for the purposes of clarity.

Note that in all the strategies the memory footprint of both the mappers and the reducers is flat at scale.



Note that the strategies all work reasonably well with both dense and sparse matrices. For sparse matrices we do not emit zero elements. That said, the simple pseudo-code for multiplying the individual blocks shown here is certainly not optimal for sparse matrices. As a learning exercise, our focus here is on mastering the MapReduce complexities, not on optimizing the sequential matrix multiplication algorithm for the individual blocks.

### Steps

1. setup ()
  2. var NIB = (I-1)/IB+1
  3. var NKB = (K-1)/KB+1
  4. var NJB = (J-1)/JB+1
  5. map (key, value)
  6. if from matrix A with key=(i,k) and value=a(i,k)
  7. for 0 <= jb < NJB
  8. emit (i/IB, k/KB, jb, 0), (i mod IB, k mod KB, a(i,k))
  9. if from matrix B with key=(k,j) and value=b(k,j)
  10. for 0 <= ib < NIB
  11. emit (ib, k/KB, j/JB, 1), (k mod KB, j mod JB, b(k,j))
- Intermediate keys (ib, kb, jb, m) sort in increasing order first by ib, then by kb, then by jb, then by m. Note that m = 0 for A data and m = 1 for B data.

### The partitioner maps intermediate key (ib, kb, jb, m) to a reducer r as follows:

11.  $r = ((ib * JB + jb) * KB + kb) \bmod R$
11. These definitions for the sorting order and partitioner guarantee that each reducer R [ib, kb, jb] receives the data it needs for blocks A [ib, kb] and B [kb, jb], with the data for the A block immediately preceding the data for the B block.
12. var A = new matrix of dimension IB x KB
14. var B = new matrix of dimension KB x JB
13. KB x JB
14. var sib = -1
15. var skb = -1

### Reduce (key, valueList)

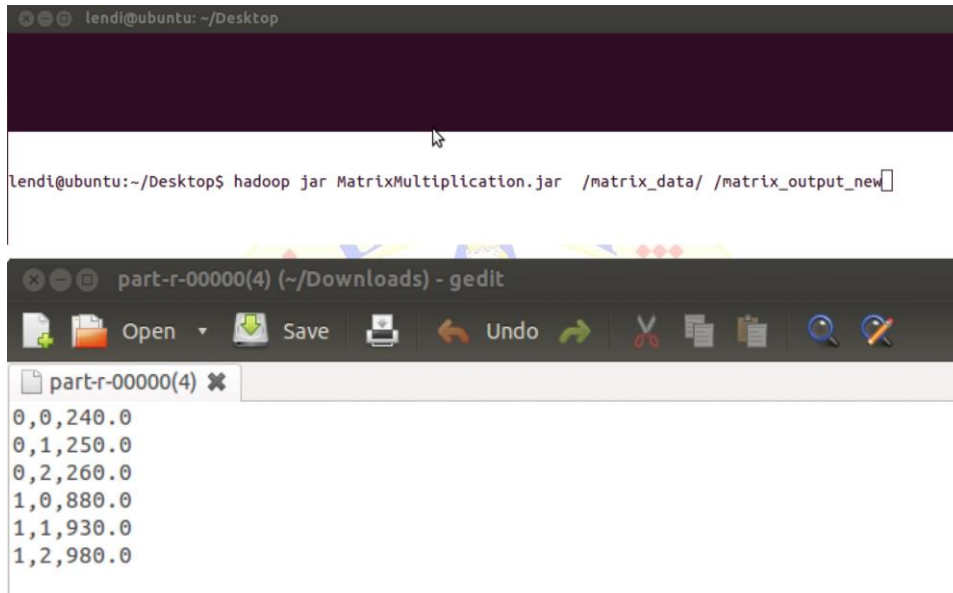
16. if key is (ib, kb, jb, 0)
17. // Save the A block.
18. sib = ib
19. skb = kb
20. Zero matrix A
21. for each value = (i, k, v) in valueList A (i, k) = v
22. if key is (ib, kb, jb, 1)
23. if ib != sib or kb != skb return // A [ib, kb] must be zero!
24. // Build the B block.
25. Zero matrix B
26. for each value = (k, j, v) in valueList B (k, j) = v
27. // Multiply the blocks and emit the result.
28. ibase = ib \* IB
29. jbase = jb \* JB
30. for 0 <= i < row dimension of A
31. for 0 <= j < column dimension of B
32. sum = 0
33. for 0 <= k < column dimension of A = row dimension of B

- a.  $\text{sum} += A(i, k) * B(k, j)$
34. if  $\text{sum} \neq 0$  emit ( $\text{ibase} + i, \text{jbase} + j$ ),  $\text{sum}$

**INPUT:**

Set of Data sets over different Clusters are taken as Rows and Columns

**OUTPUT:**



The image shows two overlapping windows. The top window is a terminal titled 'lendi@ubuntu: ~/Desktop'. It contains the command: `lendi@ubuntu:~/Desktop$ hadoop jar MatrixMultiplication.jar /matrix_data/ /matrix_output_new`. The bottom window is a gedit editor titled 'part-r-00000(4) (~/.Downloads) - gedit'. It displays the output of the Hadoop job, which consists of six lines of text: `0,0,240.0`, `0,1,250.0`, `0,2,260.0`, `1,0,880.0`, `1,1,930.0`, and `1,2,980.0`.

## Experiment 6:

**Aim:** Implement matrix multiplication with Hadoop Map Reduce

**Code:**

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
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.output.FileOutputFormat;

public class MatrixMultiplication {

    // Mapper function
    public static class MatrixMapper extends Mapper<LongWritable,
Text, Text, Text> {
        private Text outKey = new Text();
        private Text outValue = new Text();
```

```

    public void map(LongWritable key, Text value, Context
context)

        throws IOException, InterruptedException {
    Configuration conf = context.getConfiguration();
    int matrixSize = Integer.parseInt(conf.get("matrixSize"));
    String[] row = value.toString().split(",");
    String matrixType = row[0];
    int rowIndex = Integer.parseInt(row[1]);
    if (matrixType.equals("A")) {
        for (int k = 0; k < matrixSize; k++) {
            outKey.set(rowIndex + "," + k);
            outValue.set(matrixType + "," + row[2] + "," + row[3]);
            context.write(outKey, outValue);
        }
    } else {
        for (int i = 0; i < matrixSize; i++) {
            outKey.set(i + "," + rowIndex);
            outValue.set(matrixType + "," + row[2] + "," + row[3]);
            context.write(outKey, outValue);
        }
    }
}
}

```

```

    }

    }
    // Reducer function

    public static class MatrixReducer extends Reducer<Text, Text,
Text, DoubleWritable> {

        public void reduce(Text key, Iterable<Text> values, Context
context)

            throws IOException, InterruptedException {
        Configuration conf = context.getConfiguration();
        int matrixSize = Integer.parseInt(conf.get("matrixSize"));
        double[] a = new double[matrixSize];
        double[] b = new double[matrixSize];
        for (Text val : values) {
            String[] row = val.toString().split(",");
            int index = Integer.parseInt(row[1]);
            double value = Double.parseDouble(row[2]);
            if (row[0].equals("A")) {
                a[index] = value;
            } else {
                b[index] = value;
            }
        }
        double result = 0.0;
        for (int i = 0; i < matrixSize; i++) {
            result += a[i] * b[i];
        }
        context.write(key, new DoubleWritable(result));

    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        conf.set("matrixSize", args[2]);
        Job job = Job.getInstance(conf, "Matrix Multiplication");
    }

```

```

// Set mapper, combiner, and reducer classes
job.setJarByClass(MatrixMultiplication.class);
job.setMapperClass(MatrixMapper.class);
job.setReducerClass(MatrixReducer.class);

// Set input and output file formats
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(DoubleWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileInputFormat.addInputPath(job, new Path(args[1]));
FileOutputFormat.setOutputPath(job, new Path(args[3]));

// Run the job and wait for completion
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

### Output:

|         |         |           |
|---------|---------|-----------|
| A,0,0,1 | B,0,0,1 | 0,0 14.0  |
| A,0,1,2 | B,0,1,2 | 0,1 32.0  |
| A,0,2,3 | B,0,2,3 | 0,2 50.0  |
| A,1,0,4 | B,1,0,4 | 1,0 32.0  |
| A,1,1,5 | B,1,1,5 | 1,1 77.0  |
| A,1,2,6 | B,1,2,6 | 1,2 122.0 |
| A,2,0,7 | B,2,0,7 | 2,0 50.0  |
| A,2,1,8 | B,2,1,8 | 2,1 122.0 |
| A,2,2,9 | B,2,2,9 | 2,2 194.0 |

## Experiment 7:

**Aim:** Install PIG. Write Pig Latin scripts sort, group, join, project, and filter your data. Run the Pig Latin Scripts to find Word Count.

### Code (Sorting Data):

```
-- Load input data
input_data = LOAD 'input_data.txt' AS (id:int, name:chararray,
value:int);

-- Sort by value in descending order
sorted_data = ORDER input_data BY value DESC;

-- Output sorted data to file
STORE sorted_data INTO 'sorted_data.txt' USING PigStorage();
```

### Code (Grouping Data):

```
-- Load input data
input_data = LOAD 'input_data.txt' AS (id:int, name:chararray,
value:int);

-- Group data by name and calculate sum of values
grouped_data = GROUP input_data BY name;
summed_data = FOREACH grouped_data GENERATE group AS name,
SUM(input_data.value) AS total;

-- Output grouped data to file
STORE summed_data INTO 'grouped_data.txt' USING PigStorage();
```

### Code (Joining Data):

```
-- Load input data
users = LOAD 'users.txt' AS (id:int, name:chararray, age:int);
transactions = LOAD 'transactions.txt' AS (user_id:int,
item:chararray, price:double);

-- Join data on user_id
joined_data = JOIN users BY id, transactions BY user_id;

-- Project only the relevant fields
projected_data = FOREACH joined_data GENERATE users.name AS name,
transactions.item AS item, transactions.price AS price;

-- Output joined and projected data to file
STORE projected_data INTO 'joined_data.txt' USING PigStorage();
```

### **Code (Projecting Data):**

```
-- Load input data
input_data = LOAD 'input_data.txt' AS (id:int, name:chararray,
value:int);

-- Project only the name and value fields
projected_data = FOREACH input_data GENERATE name, value;

-- Output projected data to file
STORE projected_data INTO 'projected_data.txt' USING
PigStorage();
```

### **Code (Filtering Data):**

```
-- Load input data
input_data = LOAD 'input_data.txt' AS (id:int, name:chararray,
value:int);

-- Filter data to only include values greater than or equal to 10
filtered_data = FILTER input_data BY value >= 10;

-- Output filtered data to file
STORE filtered_data INTO 'filtered_data.txt' USING PigStorage();
```

### **Code (Word Count):**

```
-- Load input data
input_data = LOAD 'input_data.txt' AS (line:chararray);

-- Tokenize lines into words
tokenized_data = FOREACH input_data GENERATE
FLATTEN(TOKENIZE(line)) AS word;

-- Group words and count occurrences
grouped_data = GROUP tokenized_data BY word;
word_count = FOREACH grouped_data GENERATE group AS word,
COUNT(tokenized_data) AS count;

-- Output word count to file
STORE word_count INTO 'word_count.txt' USING PigStorage();
```



## Experiment 8:

**Aim:** Install HIVE. Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

### Code (Create Database):

```
CREATE DATABASE my_database;
```

### Code (Drop Database):

```
DROP DATABASE my_database;
```

### Code (Create Table):

```
CREATE TABLE my_table (  
    id INT,  
    name STRING,  
    value DOUBLE  
)  
  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
STORED AS TEXTFILE;
```

### Code (Drop Table):

```
DROP TABLE my_table;
```

### Code (Create View):

```
CREATE VIEW my_view AS  
SELECT name, SUM(value) AS total  
FROM my_table  
GROUP BY name;
```

### Code (Create Function):

```
CREATE FUNCTION my_function AS 'com.example.MyFunction' USING JAR  
'hdfs://my_hdfs_path/my_function.jar';
```

### Code (Create Index):

```
CREATE INDEX my_index ON TABLE my_table (name) AS 'COMPACT'  
WITH DEFERRED REBUILD;
```

## Experiment 9:

**Aim:** Install MongoDB. Create database in MongoDB

```
[root@layerstack ~]# mongo
MongoDB shell version v4.0.13
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("b7dcbd58-2750-41fa-acea-e049a596a5b5") }
MongoDB server version: 4.0.13
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
Server has startup warnings:
> _
```

**Code (Create a database):**

```
use new_database
```

## Experiment 10:

**Aim:** Write R programs for various functions of Array, Matrix and Factor

**Code:**

```
# Create an array
my_array <- array(1:24, dim = c(2, 3, 4))
print(my_array)

# Create a matrix
my_matrix <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
print(my_matrix)

# Create a factor
my_factor <- factor(c("red", "green", "blue", "red", "green"))
print(my_factor)

# Accessing Array Elements
print(my_array[1, 2, 3])

# Accessing Matrix Elements
print(my_matrix[1, 2])

# Subset of Array
print(my_array[, , 2])
```

```

# Subset of Matrix
print(my_matrix[, 2])
# Changing Array Elements
my_array[1, 2, 3] <- 0
print(my_array)
# Changing Matrix Elements
my_matrix[1, 2] <- 0
print(my_matrix)
# Length of Array
print(length(my_array))
# Dimensions of Array
print(dim(my_array))
# Number of Elements in Matrix
print(length(my_matrix))
# Dimensions of Matrix
print(dim(my_matrix))
# Levels of Factor
print(levels(my_factor))
# Number of Levels of Factor
print(nlevels(my_factor))
# Summary of Factor
summary(my_factor)

```

```

, , 1

      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

      [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

      [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

```

```

, , 4

      [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24

      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

[1] red  green blue  red  green
Levels: blue green red

[1] 11
[1] 2

```

```

      [,1] [,2]
[1,]    3    9
[2,]    4   10
[3,]    5   11
[4,]    6   12

[1] 2 3

, , 1

      [,1] [,2] [,3]
[1,]    1    0    5
[2,]    2    4    6

, , 2

```