Date :

EXPERIMENT- 1

* AIM: Understanding and using basic HDFS commands

1. Make directory Command

This is no different from the UNIX mkdir command and is used to create a directory or a HDFS environment. The option -P is used to mention not to fail if the directory already exists

syntax:

```
$ hadoop fs-mkdir [-P] </source path>
```

usage:

```
$ hadoop fs-mkdir/user/hadoop/
$ hadoop fs-mkdir/user/data/
```

2. List Command : LS

This is no different from the UNIX ls ~~cmd~~ command and it is used for listing the directories present under a specific directory in or HDFS system The -lsr command may be used for the receiver listing of the directories and files under a specific folder.

→ The -d option is used to list the directories as plain files

→ The -h option is used to format the size of files into a human-readable manner.

Syntax:

```
$ hadoop fs - [-d] [-h] [-R] </source path>
```

usage:

```
$ hadoop fs -ls /
$ hadoop fs -lsr /
```

3. Put command: put

The put command feeds the data into the HDFS

This command is used to copy files from the local file system to the HDFS file system. This is very much similar to the - copy from local command.

The -p flag presource the access modification time

Syntax:

```
$ hadoop fs - put [-f] [-p] </ source path>
```

usage:

```
$ hadoop fs - put sample.txt /user/ data/
```

4. Get Command get:

This command copies files from HDFS file system to the local file system just the opposite of the command that we have seen just now.

Syntax:

```
hadoop dfs -get <source path> </destination
                 path>
```

Usage:
```
$ hadoop fs-get/user/ data / sample.txt
            workspace/
```

5. View Contents of particular file: CAT
This command is similar to the UNIX cat command and is used for displaying the contents of a file on the console
Syntax:
```
hadoop dfs -cat </path [filename]>
$ hadoop fs-cat /user/data/sample.txt
```
Usage:
```
$ hadoop fs-cat /user/data/Sample.txt
```

6. Copy Command: cp
This command is similar to the UNIX cp command, and it is used for copying files from one directory to another directory within the HDFS filed system.
Usage:
```
$ hadoop fs-cp/user/data/Sample1.txt/
          user/hadoop1
$ hadoop fs-cp/user/data/Sample2.txt/
          user/text/in1
```

7. Move Command : mv
This command is similar to the UNIX mv command, and it is used for moving a file

from one directory to another directory within the HDFS file system
Usage:

```
$ hadoop fs -mv /user/hadoop/sample1.text /
        user/text/
```

8. Removing the file: rm
This command is similar to the UNIX rm command, and it is used for removing a file from the HDFS file system
The command -rmr can be used to delete files recursively
The -skiptrash option is used to bypass the trash, if enabled and then it immedialely delete
The -f option is used to mention that if there is no file existing
Syntax:

```
$ hadoop fs -rm [-f][-r][-R] [-skipTrash]
        <path [filename]>
```

Usage:

```
$ hadoop fs -rm -r /user/test/sample.txt
```

9. Duplicating a complete file inside the HDFS
The 'copy from local' command will copy file from the local file system to the HDFS
Syntax:

```
= hadoop dfs -copy from local </source path>
        </destination path>
```

10. Run a cluster Balancing utility:
The command 'balance' will check for work load on nodes in cluster and balance it
Syntax:
```
hadoop balance
```

11. Touchz:
This command can be used to create a file of zero bytes size in HDFS file system
usage:
```
$hadoop fs-touchz URL
```

12. Expunge:
This command is used to empty the trash available in a HDFS system
Syntax:
```
$ hadoop fs-expunge
```
usage
```
user @ubuntu1: $ hadoop fs-expunge

17/10/15 10:15:22 INFO fs.TrashPolicyDefault:
Namenode trash Configuration: Deletion
```

13. Append To file:
This command appends the contents of all the given local files to the provided destination file on the HDFS file system

Syntax:
```
$ hadoop fs -append to file < local files
    separated by space> <hdfs destination files>
```
usage:
```
user @ ubuntu 1: - $ hadoop fs-appendtofile
    derby. log. data. tsv / in / append file
```

14. df:

This command is used to show the capacity free and the used space available on the HDFS file system

Syntax:
```
$ hadoop fs -[-h] [<path>----]
```

15. du:

This command is used to show the amount of space in bytes that has been used by files that match the specified file pattern

Syntax:
```
$ hadoop fs-du [-s] [-h] <paths>
```

16. Count:

This command is used to count the no. of directories, files & bytes under the path that match the provided file pattern

Syntax:
```
$ hadoop fs. count [-q] < path>
```

Date :

17. Chmod:

This command is used to change the permissions of a file, this command works as similar to that of LINUX's shell command chmod

Syntax:

```
$ hadoop fs -chmod [-R] <MODE [,MODE] ...
               /OCTALMODE> PATH
```

$\boxed{5/5}$

[illegible]

1/10/24

Date :

EXPERIMENT - 2

* **AIM**: For the given input file, will perform word count (space and new line as delimiter

* **PROGRAM**:

```
Package wordpack;
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.path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat
import org.apache.hadoop.util.GenericOptionsParser;

Public class WordCount
 {
 public static class MapForWordCount extends Mapper
         <LongWritable, Text, Text, IntWritable>

  {
```

```
public void map(LongWritable key, Text value,
        Context con) throws IOException, Interrupted
        Exception

  {
    string line = value.toString();
    StringTokenizer token = new StringTokenizer(line);
    While(token.hasmoreTokens())
    {
      string status = new String();
      String word = token.nextToken();
      Text outputKey = new Text(word);
      IntWritable outputValue = new IntWritable(1);
      con.write(outputKey, outputValue);
    }
  }
}

Public static class ReduceforWordCount extends
Reducer<Text, IntWritable, Text, IntWritable>

  {
    public void reduce(Text word, Iterable <IntWrita
        values, Context con) throws IOException,
      InterruptedException

        {
```

```
        int sum=0;
        for (IntWritable value : values)
          {
              sum+ = value.get();
          }
          con.write (word, new IntWritable(sum));
        }

  }

Public static void main( String[] args) throws
            Exception
  {

        Configuration c= new Configuration();
        String[] files = new GenericOptionsParser (C, args)
                  . getRemainingArgs(),
        path input = new path (files[0]);
        path output = new path (files[1]);
        Job j = new job (c, "wordCount");

        j.setMapperClass (MapForWordCount.class);
        j.setReducer class(ReduceforWordCount.class);
        j.setOutputKeyClass (Text.class);
        j.setOutputValueClass (IntWritable.class);
        FileInputFormat.addInputPath ( j, input);
        System.exit(j.waitForCompletion(true)? 0 : 1);
    }
}
```

Date :

EXPERIMENT - 3

*** AIM:** Analysis of weather Dataset on multi node cluster

*** PROGRAM:**

```
import java.io.IOException;
import java.util.stringTokenizer;
import org.apache.hadoop.io.text;
import org.apache.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.path;
import org.apache.hadoop.mapreduce.lib.input.fileInputformat;
import org.apache.hadoop.mapreduce.lib.output.TextOutput;

public class CalculateMaxAndMinTemperatureWithTime {
    public static String calOutputName = "california";
    Public static String nyOutputName = "Newyork";
    Public static String njOutputName = "Newjersy";
    Public static ausOutputName = "Austin";
    Public static String bosOutputName = "Boston";

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 currentTemp = null;
    float minTemp = Float.MAX_VALUE;
    float maxTemp = Float.MIN_VALUE;
    string date = null;
    string currentTime = null;
    string minTempANDTime = null;

   while (strTokens.hasMoreElements()){
      if (Counter == 0){
       date = strTokens.nextToken();
     } else {
        if (counter % 2 == 1){
            currentTime = strTokens.nextToken()
     }else {
        currentTemp = Float.parseFloat(strTokens.nextToken());
     }
     if (maxTemp <currnetTemp){
         maxTemp = currentTemp;
         maxTempANDTime = maxTemp + "AND" + currentTime;
    }
  }
 }
```

```
  counter++;
 }

Temp temp = new Text();
temp.set (maxTempANDTime);
Text dateText = new Text();
try {
  con.write (dataText, temp);
 }
 catch (Exception e) {
     e.printStackTrace();
  }

  temp.set (minTempANDTime);
  dataText.set (data);
  con.write ( dataText, 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
throws IOException, InterruptedException {
 int counter=0;
 String reduceInputStr[]=null;
  String f1Time="";
  String f2Time="";
  Text result=new Text();
   for (Text value :values){
  if (counter==0){
       f1 = reduceInputStr[0];
    }
 else{
      f2 = reduceInputStr[0];
      f2Time= reduceInputStr[1];
  }
 Counter = Counter + 1

result=new Text("Time:"+"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 = CalculateMaxAndMinTemperatureTime.calOutputName;
}
else if (Key.toString().subString(0, 2).equals ("NY")){
fileName = CalculateMaxAndMinTemperatureTime.nyOutputName
}
else if (Key.toString().substring (0, 3).equals ("BAL")){
fileName = CalculateMaxAndMinTemperatureTime.balOutputNa

}
 String strArr [] = Key.toString().split("_").
 Key.set(setArr [1]);
}

@ 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.setMapperClass (WhetherForcastMapper.class);
job.setMapOutputKeyClass(Text.class);
job.setOutputValueClass( Text.class);

MultipleOutputs.addNamedOutput(job, calOutputName,
      TextOutputFormat.class, Text.class, Text.class);

TextOutputFormat.class, Text.class, Text.class);
 MultipleOutputs.addNamedOutput (job, rjOutputName,
    TextOutputsFormat.class, Text.class, Text.class);

path pathInput = new Path(
   "hdfs://192.168.213.133:54310/weatherInputData/input
            temp.txt");

 fileinputFormat.addInputPath(job, pathInput);

try {
   System.exit(job waitForCompletion(true)? 0 : 1);
 }
 catch (Exception e){
   e.printStackTrace();
 }
}
}
```

EXPERIMENT- 4

*AIM: Program to read the data from file and write data into file

*PROGRAM:

```
import java.io.File;
import java.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FSDataOutputStream;

public class HelloDFS{
    public static final String thefilename = "hello.txt";

    public static final String message = "Hello HDFS! \n";
    public static void main (String [] args) throws
                    IOException{

        Configuration conf = new Configuration();
        File System hdfs = fileSystem.get(conf);
        path filename Path = new Path(thefilename);
        try{

        if(hdfs.exists(filenamePath)){
```

```
        hdfs.delete(filenamePath, true);

    }

    FSDataOutputStream out = hdfs.create(filenamePath);
        out.writeUTF(message);
        out.close();

    FSDataInputStream in = hdfs.open(filenamePath);
      String messageIn = in.readUTF();
      System.out.print(messageIn);
      in.close();
    }
     catch (IOException ioe){
         System.err.println("IOException during operation"
           + ioe.toString());

       System.exit(1);

    }
  }
}
```