1. Install Hadoop and Implement the following file management tasks in Hadoop: Adding files and directories, retrieving files, deleting files and directories.
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.**

1) Create a directory
$mkdir Hadoop

2) Create a file in a Hadoop directory of file name hello.txt
$cd Hadoop/
$vi hello.txt
(Type the file content and Press Esc: wq)

3) Display the content of the File
$cat hello.txt

4) Create directory in HDFS
$hdfs dfs -mkdir /today

5) Copy the file from Hadoop to hdfs
$hdfs dfs -put hello.txt /today/data.txt

6) Copy the file within hdfs
$hdfs dfs -mkdir /input
$hdfs dfs -cp /today/data.txt /input/data.txt
$hdfs dfs -cat /input/data.txt

7) Copy the file from hdfs to Hadoop
$ hdfs dfs -get /input/data.txt

8) Delete the files
$hdfs dfs -rm /today/data.txt

9) Remove directory
$hdfs dfs -rmdir /today

2. Hadoop Programming: Word Count MapReduce Program.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
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 java.io.IOException;
import java.util.StringTokenizer;

public class WordCount {


public static class TokenizerMapper extends Mapper<Object, Text, Text,
IntWritable> {
     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();

     public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
{
          StringTokenizer itr = new
     StringTokenizer(value.toString());
          while (itr.hasMoreTokens()) {
               word.set(itr.nextToken());
               context.write(word, one);
          }
     }
}
public static class IntSumReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {
     public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
          int sum = 0;
          for (IntWritable val : values) {
               sum += val.get();
          }
          context.write(key, new IntWritable(sum));
```

```
      }
}
public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "word count");
      job.setJarByClass(WordCount.class);
      job.setMapperClass(TokenizerMapper.class);
      job.setCombinerClass(IntSumReducer.class);
      job.setReducerClass(IntSumReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
}
```

Compile and Package the Java Code

javac -classpath `hadoop classpath` -d . WordCount.java jar cf wc.jar WordCount*.class

Run the Hadoop Word Count Job Upload the input file to HDFS:

hdfs dfs -mkdir -p /user/cloudera/input

hdfs dfs -put sample.txt /user/cloudera/input/

Run the Map Reduce job

hadoop jar wc.jar WordCount /user/cloudera/input /user/cloudera/output

View the Output:

hdfs dfs -cat /user/cloudera/output/part-r-00000

3. **Implementing Matrix Multiplication Using Map-Reduce.**

```
from functools import reduce

#Function to create key-value pairs for the Map step
def mapper(matrix1, matrix2):
      #List to store key-value pairs
      key_value_pairs = []

      #Iterate over the rows of the first matrix
      for i in range(len(matrix1)):
            #Iterate over the columns of the second matrix
            for j in range(len(matrix2[0])):
                  for k in range(len(matrix2)):
                        #Create key-value pair (i, j) -> matrix1[i][k]
                        * matrix2[k][j] key_value_pairs.append(((i, j),
                        matrix1[i][k] * matrix2[k][j]))
      return key_value_pairs

#Function to sum values for the Reduce step
def reducer(key, values):
      return sum(values)

# Function to perform MapReduce
def map_reduce(matrix1, matrix2):
      #Apply the mapper function to create key-value pairs
      key_value_pairs = mapper(matrix1, matrix2)

      # Group values by key
      grouped_values = {}
      for key, value in key_value_pairs:
            if key not in grouped_values:
                  grouped_values[key] = []
            grouped_values[key].append(value)

      # Apply the reducer function to get final results
      result = {}
      for key in grouped_values:
            result[key] = reducer(key, grouped_values[key])
      return result

# Function to convert result dictionary to matrix
def result_to_matrix(result, rows, cols):
```

```
    matrix = [[0 for _ in range(cols)] for _ in range(rows)]
    for key, value in result.items():
        matrix[key[0]][key[1]] = value
    return matrix

# Example matrices
matrix1 = [
    [1, 2, 3],
    [4, 5, 6]
]

matrix2 = [
    [7, 8],
    [9, 10],
    [11, 12]
]

# Perform MapReduce matrix multiplication
result = map_reduce(matrix1, matrix2)

# Convert result dictionary to matrix
result_matrix = result_to_matrix(result, len(matrix1),
len(matrix2[0]))

# Print result matrix
for row in result_matrix:
    print(row)
```

4. Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB

Install MongoDB on Fedora using below commands

$sudo dnf install -y mongodb mongodb-server
$sudo systemctl start mongod
$sudo systemctl enable mongod
$sudo systemctl status mongod
$sudo dnf install mongodb-mongosh Start the MongoDB shell by running:
$mangosh
Select a database
$use testDB

Sample Data:

```
[
{ "_id": 1, "name": "Alice", "age": 22, "marks": 85 },
{ "_id": 2, "name": "Bob", "age": 21, "marks": 78 },
{ "_id": 3, "name": "Charlie", "age": 23, "marks": 92 },
{ "_id": 4, "name": "David", "age": 20, "marks": 88 },
{ "_id": 5, "name": "Eve", "age": 22, "marks": 76 }
]
```

Create students collection using above sample data

```
db.createCollection('Students');
db.student.insertOne({ "_id": 1, "name": "Alice", "age": 22, "marks": 85 })
```

i. Count

Count the number of students age is greater than 20

```
$ db.student.count({ age: { $gt: 20 } })
```

ii. Sort

Sort documents by age in descending order

```
$ db.students.find().sort({ marks: -1 })
```

iii. Limit

Limit the number of returned documents to 5:

```
$db.students.find().sort({ marks: -1 }).limit(3)
```

iv. Skip

$db.students.find().sort({ marks: -1 }).skip(2)

v. Aggregate Data

db.students.aggregate([{ $group: { _id: "$age", avgMarks: { $avg: "$marks" } } }])

5. **Develop Pig Latin scripts to sort, group, join, project, and filter the data.**

Sample Data

Let's assume we have two datasets:

- employees.txt (Employee ID, Name, Age, Department ID, Salary)
- departments.txt (Department ID, Department Name)

employees.txt (stored in HDFS at /user/cloudera/employees.txt)

101,John,30,1,50000

102,Sam,28,2,60000

103,Anna,32,1,75000

104,David,29,3,62000

105,Lily,27,2,58000

departments.txt (stored in HDFS at /user/cloudera/departments.txt)

1. HR
2. Finance
3. IT

Pig Latin Script

Save the script as employee_analysis.pig and execute it in Cloudera.

```
-- Load the employees dataset
employees = LOAD 'hdfs://localhost:9000/user/cloudera/employees.txt'
USING PigStorage(',')
AS (emp_id:int, name:chararray, age:int, dept_id:int, salary:int);

-- Load the departments dataset
departments = LOAD 'hdfs://localhost:9000/user/cloudera/departments.txt'
USING PigStorage(',')
AS (dept_id:int, dept_name:chararray);
```

```
-- 1. FILTER: Select employees with age greater than 28
filtered_employees = FILTER employees BY age > 28;

-- 2. PROJECT: Select only emp_id, name, and salary columns
projected_employees = FOREACH filtered_employees GENERATE emp_id, name, salary;

-- 3. SORT: Order employees by salary in descending order
sorted_employees = ORDER projected_employees BY salary DESC;

-- 4. GROUP: Group employees by department ID
grouped_by_department = GROUP employees BY dept_id;

-- 5. JOIN: Join employees with department names using dept_id
joined_data = JOIN employees BY dept_id, departments BY dept_id;

-- STORE results in HDFS
STORE sorted_employees
INTO 'hdfs://localhost:9000/user/cloudera/output/sorted_employees'
USING PigStorage(',');

STORE grouped_by_department
INTO 'hdfs://localhost:9000/user/cloudera/output/grouped_by_department'
USING PigStorage(',');
STORE joined_data
INTO 'hdfs://localhost:9000/user/cloudera/output/joined_data'
USING PigStorage(',');

-- DISPLAY the results on the screen
DUMP sorted_employees;
DUMP grouped_by_department;
DUMP joined_data;
```

Upload the data in HDFS

hdfs dfs -mkdir -p /user/cloudera

hdfs dfs -put employees.txt /user/cloudera/

hdfs dfs -put departments.txt /user/cloudera/

Run the Script

pig -x mapreduce employee_analysis.pig

ouput commands

hdfs dfs -cat /user/cloudera/output/sorted_employees/part-r-00000

hdfs dfs -cat /user/cloudera/output/grouped_by_department/part-r-00000

hdfs dfs -cat /user/cloudera/output/joined_data/part-r-00000

6. **Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.**

Create a Database:

```
CREATE DATABASE employee_db;
```

Use the database:

```
USE employee_db;
```

Alter a Database:

```
ALTER DATABASE employee_db SET DBPROPERTIES ('Owner'='Admin');
```

Drop a Data Base

```
DROP DATABASE employee_db CASCADE;
```

Create a Table

```
CREATE TABLE employees (
        emp_id INT,
        name STRING,
        age INT,
        dept_id INT,
        salary FLOAT
        )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

LOAD DATA INPATH '/user/cloudera/employees.txt' INTO TABLE employees;
```

Alter a Table

Add a New Column

```
ALTER TABLE employees ADD COLUMNS (email STRING);
```

Rename:

```
ALTER TABLE employees RENAME TO employees_new;
```

Drop Table:

DROP TABLE employees_new;

Create a View

CREATE VIEW high_salary_employees AS
SELECT emp_id, name, salary
FROM employees
WHERE salary > 50000;

Alter a View:

ALTER VIEW high_salary_employees AS
SELECT emp_id, name, age, salary
FROM employees
WHERE salary > 60000;

Drop View:

DROP VIEW high_salary_employees;

Create a Function

Add a JAR file containing a Java-based UDF:
ADD JAR /user/cloudera/custom_udf.jar;
CREATE FUNCTION to_upper AS 'com.example.hiveudf.ToUpperUDF';

Use the Function

SELECT to_upper(name) FROM employees;

Drop Function

DROP FUNCTION to_upper;

Create an Index

CREATE INDEX emp_dept_idx
ON TABLE employees (dept_id)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
WITH DEFERRED REBUILD;

Rebuild the Index:

ALTER INDEX emp_dept_idx ON employees REBUILD;

Drop Index

DROP INDEX emp_dept_idx ON employees;

Check all tables in the current database:

SHOW TABLES;

DESCRIBE employees;

Display the table data

SELECT * FROM employees LIMIT 5;

7. **Implementing Frequent Item set algorithm using Map-Reduce.**

fequent_itemset_reducer.py

```
#!/usr/bin/env python

import sys
item1 = None
item2 = None
current_count = 0

# input comes from STDIN
for line in sys.stdin:
      # remove leading and trailing whitespace
      line = line.strip()

      # parse the input we got from mapper.py
      key1, key2, count = line.split('\t', 2)
      #print(key1, key2, count)

      # convert count (currently a string) to int
      try:
            count = int(count)
      except ValueError:
            # count was not a number, ignore
            continue

      if item1 == key1 and item2 == key2:
            current_count += count
      else:
            if item1!=None and item2!=None and current_count>=2:
                  # output as a frequent 2 itemset if support>2
                  print('%s\t%s' % (item1, item2))
            current_count = count
            item1 = key1
            item2 = key2

# last itemset
if item1 == key1 and item2 == key2 and current_count>=2:
      print('%s\t%s' % (item1, item2))
```

frequent_itemset_mapper.py

```
#!/usr/bin/env python

import sys
import re

# input comes from STDIN (standard input)
for line in sys.stdin:
      # remove leading and trailing whitespace
      line = line.strip()
      # split the line into digits
      words = re.findall(r'\w+', line)

      lst = []
      for i in words:
            lst.append(i)    #lst contains items in each transaction
      for item1 in words:
            for item2 in lst:
                  if item1!=item2 and lst:
                        print('%s\t%s\t%s'   %   (item1,   item2,   1))
#outputs frequent 2itemsets with value 1 and key=item1, item2
            lst.pop(0)
```

frequent_itemset_data.txt

```
apple      orange     grapes
apple      mango
orange     apple      chickoo
apple      grapes     orange
milk       butter
butter     cheese
bread      milk       butter
cheese     bread      buttermilk
cheese     apple      orange
```

Step 1: Create a directory on HDFS.

```
sudo -u hdfs hadoop fs -mkdir /frequent_itemset
hdfs dfs -ls /
```

Step 2: Copy input file to HDFS.

```
sudo -u hdfs hadoop fs -put /home/cloudera/frequent_itemset_data.txt /frequent_itemset
hdfs dfs -ls /frequent_itemset
```

Step 3: Configure permissions to run MapReduce for Frequent Itemset Mining on Hadoop.

```
chmod 777 frequent_itemset_mapper.py frequent_itemset_reducer.py
sudo -u hdfs hadoop fs -chown cloudera /frequent_itemset
```

Step 4: Run MapReduce on Hadoop.

Run the following command on terminal.

```
hadoop jar /home/cloudera/hadoop-streaming-2.7.3.jar \
> -input /frequent_itemset/frequent_data.txt
> -output /frequent_itemset/output \
> -mapper /home/cloudera/frequent_map.py \
> -reducer /home/cloudera/frequent_reduce.py
```

Step 5: Read MapReduce output.

```
hdfs dfs -cat /frequent_itemset/output/part-00000
```

8. Implementing Clustering algorithm using Map-Reduce

```
import java.io.IOException;
import java.util.*;
import java.io.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.Reducer;

@SuppressWarnings("deprecation")
public class kMeansClustering {
public static String OUT = "outfile";
public static String IN = "inputlarger";
public static String CENTROID_FILE_NAME = "/centroid.txt";
public static String OUTPUT_FILE_NAME = "/part-00000";
public static String DATA_FILE_NAME = "/data.txt";
public static String JOB_NAME = "KMeans";
public static String SPLITTER = "\t| ";
public static List<Double> mCenters = new ArrayList<Double>();

/*
 * In Mapper class we are overriding configure function. In this
we are reading file from Distributed Cache and then storing that
into instance variable "mCenters"
 */
public static class Map extends MapReduceBase implements
          Mapper<LongWritable,      Text,      DoubleWritable,
DoubleWritable> {
     @Override
     public void configure(JobConf job) {
          try {
               // Fetch the file from Distributed Cache Read
it and store the
               // centroid in the ArrayList
               Path[]              cacheFiles              =
DistributedCache.getLocalCacheFiles(job);
               if (cacheFiles != null && cacheFiles.length >
0) {
                    String line;
                    mCenters.clear();
```

```
                        BufferedReader    cacheReader    =    new
BufferedReader(
                                    new
FileReader(cacheFiles[0].toString()));
                        try {
                              //  Read  the  file  split  by  the
splitter and store it in
                              // the list
                              while            ((line            =
cacheReader.readLine()) != null) {
                                    String[]         temp         =
line.split(SPLITTER);

mCenters.add(Double.parseDouble(temp[0]));
                              }
                        } finally {
                              cacheReader.close();
                        }
                  }
            } catch (IOException e) {
                  System.err.println("Exception           reading
DistribtuedCache: " + e);
            }
      }

      /*
       * Map function will find the minimum center of the point
and emit it to the reducer
       */
      @Override
      public void map(LongWritable key, Text value,
                  OutputCollector<DoubleWritable,
DoubleWritable> output,
                  Reporter reporter) throws IOException {
            String line = value.toString();
            double point = Double.parseDouble(line);
            double min1, min2 = Double.MAX_VALUE, nearest_center
= mCenters
                        .get(0);
            // Find the minimum center from a point
            for (double c : mCenters) {
                  min1 = c - point;
                  if (Math.abs(min1) < Math.abs(min2)) {
                        nearest_center = c;
                        min2 = min1;
                  }
```

```
            }
            // Emit the nearest center and the point
            output.collect(new DoubleWritable(nearest_center),
                        new DoubleWritable(point));
      }
}

public static class Reduce extends MapReduceBase implements
            Reducer<DoubleWritable,             DoubleWritable,
DoubleWritable, Text> {

      /*
       * Reduce function will emit all the points to that center
and calculate
       * the next center for these points
       */
      @Override
      public       void       reduce(DoubleWritable       key,
Iterator<DoubleWritable> values,
                  OutputCollector<DoubleWritable,  Text>  output,
Reporter reporter)
                  throws IOException {
            double newCenter;
            double sum = 0;
            int no_elements = 0;
            String points = "";
            while (values.hasNext()) {
                  double d = values.next().get();
                  points = points + " " + Double.toString(d);
                  sum = sum + d;
                  ++no_elements;
            }

            // We have new center now
            newCenter = sum / no_elements;

            // Emit new center and point
            output.collect(new  DoubleWritable(newCenter),  new
Text(points));
      }
}

public static void main(String[] args) throws Exception {
      run(args);
}
```

```
public static void run(String[] args) throws Exception {
      IN = args[0];
      OUT = args[1];
      String input = IN;
      String output = OUT + System.nanoTime();
      String again_input = output;

      // Reiterating till the convergence
      int iteration = 0;
      boolean isdone = false;
      while (isdone == false) {
            JobConf conf = new JobConf(kMeansClustering.class);
            if (iteration == 0) {
                  Path    hdfsPath    =    new    Path(input    +
CENTROID_FILE_NAME);
                  //  upload  the  file  to  hdfs.  Overwrite  any
existing copy.

DistributedCache.addCacheFile(hdfsPath.toUri(), conf);
            } else {
                  Path   hdfsPath   =   new   Path(again_input   +
OUTPUT_FILE_NAME);
                  //  upload  the  file  to  hdfs.  Overwrite  any
existing copy.

DistributedCache.addCacheFile(hdfsPath.toUri(), conf);
            }

            conf.setJobName(JOB_NAME);
            conf.setMapOutputKeyClass(DoubleWritable.class);
            conf.setMapOutputValueClass(DoubleWritable.class);
            conf.setOutputKeyClass(DoubleWritable.class);
            conf.setOutputValueClass(Text.class);
            conf.setMapperClass(Map.class);
            conf.setReducerClass(Reduce.class);
            conf.setInputFormat(TextInputFormat.class);
            conf.setOutputFormat(TextOutputFormat.class);

            FileInputFormat.setInputPaths(conf,
                        new Path(input + DATA_FILE_NAME));
            FileOutputFormat.setOutputPath(conf,             new
Path(output));

            JobClient.runJob(conf);

            Path ofile = new Path(output + OUTPUT_FILE_NAME);
```

```
            FileSystem fs = FileSystem.get(new Configuration());
            BufferedReader br = new BufferedReader(new
      InputStreamReader(
                        fs.open(ofile)));
            List<Double> centers_next = new ArrayList<Double>();
            String line = br.readLine();
            while (line != null) {
                  String[] sp = line.split("\t| ");
                  double c = Double.parseDouble(sp[0]);
                  centers_next.add(c);
                  line = br.readLine();
            }
            br.close();

            String prev;
            if (iteration == 0) {
                  prev = input + CENTROID_FILE_NAME;
            } else {
                  prev = again_input + OUTPUT_FILE_NAME;
            }
            Path prevfile = new Path(prev);
            FileSystem fs1 = FileSystem.get(new Configuration());
            BufferedReader br1 = new BufferedReader(new
      InputStreamReader(
                        fs1.open(prevfile)));
            List<Double> centers_prev = new ArrayList<Double>();
            String l = br1.readLine();
            while (l != null) {
                  String[] sp1 = l.split(SPLITTER);
                  double d = Double.parseDouble(sp1[0]);
                  centers_prev.add(d);
                  l = br1.readLine();
            }
            br1.close();

            // Sort the old centroid and new centroid and check
      for convergence
            // condition
            Collections.sort(centers_next);
            Collections.sort(centers_prev);

            Iterator<Double> it = centers_prev.iterator();
            for (double d : centers_next) {
                  double temp = it.next();
                  if (Math.abs(temp - d) <= 0.1) { //convergence
      factor
```

```
                        isdone = true;
                    } else {
                        isdone = false;
                        break;
                    }
                }
                ++iteration;
                again_input = output;
                output = OUT + System.nanoTime();
            }
        }
        }
```

Input:

data.txt

```
20
23
19
29
33
29
43
35
18
25
27
47
55
63
59
69
15
25
54
89
```

centroid.txt

```
20.0
30.0
40.0
60.0
```

EXECUTION STEPS:

1. start-all.sh
2. jps
3. gedit kMeansClustering.java
4. hadoop com.sun.tools.javac.Main kMeansClustering.java
5. jar cf km.jar kMeansClustering *.class
6. create input file- data.txt & centroid.txt
7. hadoop fs –mkdir /kmm
8. hadoop fs –put data.txt centroid.txt /kmm
9. hadoop jar km.jar kMeansClustering /kmm /kmm/output
10. hadoop fs –cat /kmm/output*/part-00000

9. Implementing Page Rank algorithm using Map-Reduce

pagerank_data.txt

```
1    2
1    3
2    4
3    1
3    2
3    4
4    3
```

pagerank_mapper1.py

```
#!/usr/bin/python
import sys

for line in sys.stdin:
   if line.startswith('#'):
      continue
   else:
      print("%s" % (",".join(line.strip().split())))
```

pagerank_mapper2.py

```
#!/usr/bin/python

import sys

pageranks = {'1': '1', '2':'1', '3':'1', '4':'1'}

for line in sys.stdin:
   node, adj_list = line.strip().split('\t')
   adj_list = adj_list.split(',')
   out_num = len(adj_list)

   print("%s,%s" % (node, '0.0'))
   #print("Adjaceny list:", adj_list)

   for out_link in adj_list:
      out_link_contrib = float(pageranks[node]) / out_num
      print("%s,%s" % (out_link, out_link_contrib))
```

pagerank_reducer1.py

```
#!/usr/bin/python

import sys

cur_node = None
prev_node = None
adj_list = []

for line in sys.stdin:
   cur_node, dest_node = line.strip().split(',')
   if cur_node == prev_node:
      adj_list.append(dest_node)
   else:
      if prev_node:
         print("%s\t%s" % (prev_node, ",".join(sorted(adj_list))))

      prev_node = cur_node
      del adj_list [:]
      adj_list.append(dest_node)

if cur_node == prev_node:
   print("%s\t%s" % (prev_node, ",".join(sorted(adj_list))))
```

pagerank_reducer2.py

```
#!/usr/bin/python

import sys

cur_node = None
prev_node = None
contrib_sum = 0
damping_factor = 0.85

for line in sys.stdin:
   cur_node, contrib = line.strip().split(',')
   print("\nCurrent node: %s\nContribution by this node: %s" %
(cur_node, contrib))

   if cur_node == prev_node:
```

```
      contrib_sum += float(contrib)
   else:
      if prev_node:
         new_pr = (1 - damping_factor) + (damping_factor * 
contrib_sum)
         new_pr = round(new_pr, 5)
         print("Previous node: %s\nNew page rank of node %s: %s" % 
(prev_node, cur_node, new_pr))

      prev_node = cur_node
      contrib_sum = float(contrib)

if cur_node == prev_node:
   new_pr = (1 - damping_factor) + (damping_factor * contrib_sum)
   new_pr = round(new_pr, 5)
   print("Previous node: %s\nNew page rank of node %s: %s" % 
(prev_node, cur_node, new_pr))
```

Step 4: Create a directory on HDFS.

sudo -u hdfs hadoop fs -mkdir /pagerank
hdfs dfs -ls /

Step 5: Copy input file on HDFS.

sudo -u hdfs hadoop fs -put /home/cloudera/pagerank_data.txt /pagerank
hdfs dfs -ls /pagerank

Step 6: Configure permissions to run MapReduce for PageRank algorithm on Hadoop.

chmod 777 pagerank_mapper1.py pagerank_reducer1.py pagerank_mapper2.py pagerank_reducer.2.py
sudo -u hdfs hadoop fs -chown cloudera /pagerank

Step 7: Run MapReduce on Hadoop.

Execute the first MapReduce job on Hadoop.
hadoop jar /home/cloudera/hadoop-streaming-2.7.3.jar \
> -input /pagerank/pagerank_data.txt \
> -output /pagerank/output \
> -mapper /home/cloudera/pagerank_mapper1.py \
> -reducer /home/cloudera/pagerank_reducer1.py

Let's read the output of the first MapReduce job.

hdfs dfs -cat /pagerank/output/part-00000

Execute the second MapReduce job on Hadoop.

```
hadoop jar /home/cloudera/hadoop-streaming-2.7.3.jar \
> -input /pagerank/output/part-00000 \
> -output /pagerank/final_output \
> -mapper /home/cloudera/pagerank_mapper2.py \
> -reducer /home/cloudera/pagerank_reducer2.py
```

To read the file, simply run the command below.

hdfs dfs -cat /pagerank/final_output/part-00000

10. Develop a Map Reduce program that mines weather data and displays appropriate messages indicating the weather conditions of the day.

```
def mapper(line):
      date, condition = line.strip().split(',')
      return (date, condition)

def reducer(date, conditions):
      # Assuming conditions is a list of weather conditions for the
given date condition_counts = {}
      for condition in conditions:
            if condition in condition_counts:
                  condition_counts[condition] += 1
            else:
                  condition_counts[condition] = 1

      # Determine the most frequent weather condition
      most_frequent_condition=max(condition_counts,key=condition_cou
      nts.get)
      return (date, most_frequent_condition)

def main(input_file):
      # Read input data
      with open(input_file, 'r') as f:
            lines = f.readlines()
      # Apply mapper function
      mapped_data = [mapper(line) for line in lines]

      # Group by date
      grouped_data = {}
      for date, condition in mapped_data:
            if date in grouped_data:
                  grouped_data[date].append(condition)
```

```
            else:
                  grouped_data[date] = [condition]

      # Apply reducer function
      reduced_data = {date: reducer(date, conditions) for date,
conditions in grouped_data.items()}

      # Display results
      for date, (date_key, condition) in reduced_data.items():
            print(f"Weather on {date_key}: {condition}") 5

      # Example usage
      if name == " main ":
            input_file = 'weather.txt'
            main(input_file)
```