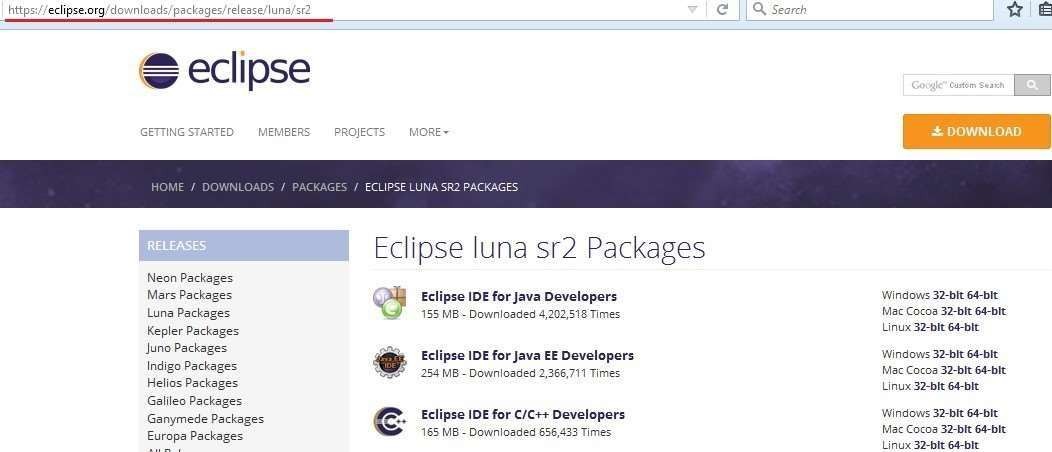
# Practical 5

## AIM: Implementing simple algorithms in Map-Reduce.

1. Configuration Wordcount Program with Eclipse IDE and Run Program in Hadoop2.x

Steps to configure and Run Wordcount Program Step: Download Eclipse according to 32 bit or 64 bit.

https://eclipse.org/downloads/packages/release/luna/sr2

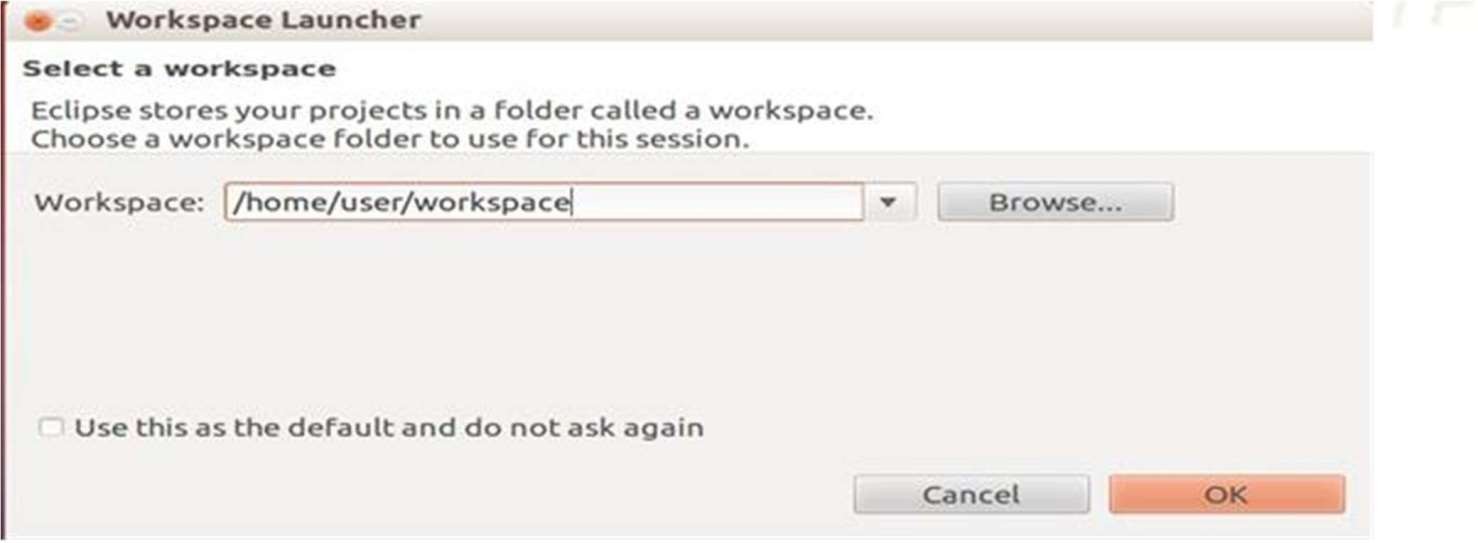


### Step 1: Extract Eclipse and Click on eclipse icon

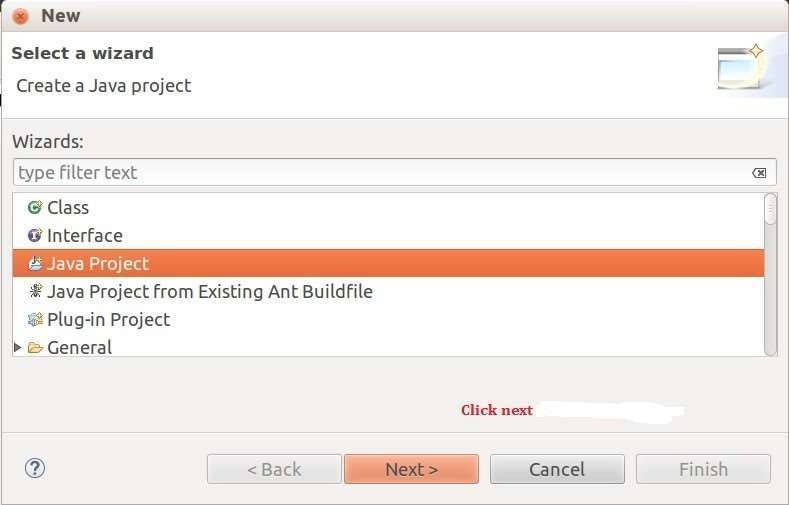




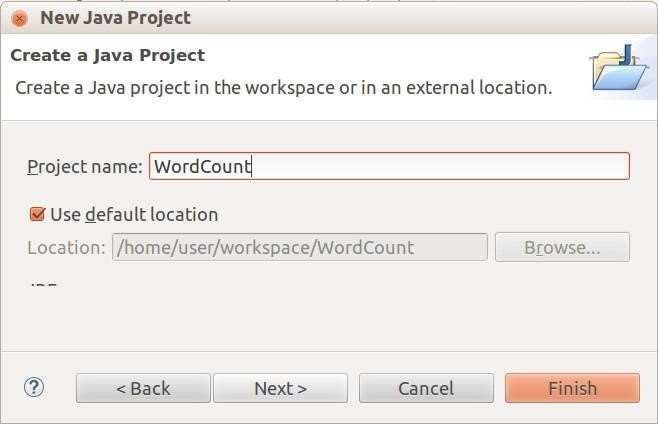
**Step 2: Create workspace in /home/use/workspace if want to change then give location or browse.**



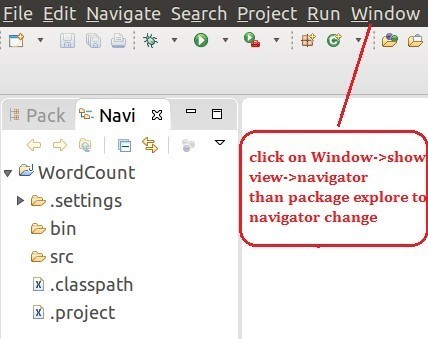
### Step 3: Create project file->new->other->java->javaproject



**Step 4: Give Project name WordCount**



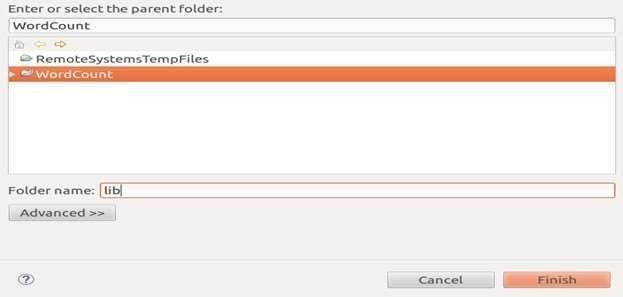
### Step 5: change the default view (Project Explore to



**Navigator) Window->show view->navigator)**

### Step 6: create lib folder inside Wordcount project. Right click on Wordcount-

**>New>Folder**

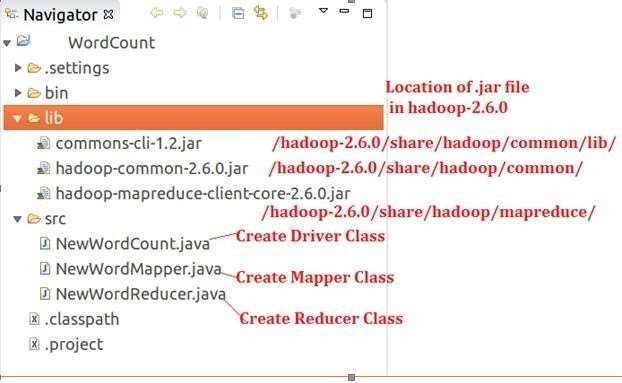


### Step 7: Copy the Three jar file in lib folder and Create Three java class.Jar file name and location.

1. /hadoop-2.6.0/share/hadoop/common/lib : commons-cli-1.2.jar
2. /hadoop-2.6.0/share/hadoop/common : hadoop-common-2.6.0.jar
3. /hadoop-2.6.0/share/hadoop/mapreduce/ : hadoop-mapredure-client-core-2.6.0.jar

Three java file for Drivercode,Mapper code,Reducer code. NewWordCount.java : main class

NewWordMapper.j ava NewWordReducer.j ava



### Step 8: set java build path (class path) by Right Click on

->WordCountProject-> Properties->JavaBuildPath->Libaries->Click on Add jar and find three jar file in lib folder of WordCount project.



### NewWordCount.java

import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.mapreduce.Job; import

org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.input.TextInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputForm at; import org.apache.hadoop.mapreduce.lib.output.TextOutputForm at; public class NewWordCount

{

public static void main(String[] args) throws Exception

{

//Creating an object of Configuration class, which loads the configuration parameters Configuration conf = new Configuration();

//Creating the object of Job class and passing the conf object and Job name as arguments. The Job class allows the user to configure the job, submit it and control its execution.

Job job = new ~~Job~~(conf, "wordcount");

//Setting the jar by finding where a given class came from job.setJarByClass(NewWordCount.class); //Setting the key class for job output data job.setOutputKeyClass(Text.class);

//Setting the value class for job output data job.setOutputValueClass(IntWritable.class);

//Setting the mapper for the job job.setMapperClass(NewWordMapper.class);

//Setting the reducer for the job job.setReducerClass(NewWordReducer.class);

//Setting the Input Format for the job

job.setInputFormatClass(TextInputFormat.class); //Setting the Output Format for the job job.setOutputFormatClass(TextOutputFormat.class);

//Adding a path which will act as a input for MR job. args[0] means it will use the first argument written on terminal as input path FileInputFormat.*addInputPath*(job, new Path(args[0])); //Setting the path to a directory where MR job will dump the output. args[1] means it will use the second argument written on terminal as output path FileOutputFormat.*setOutputPath*(job,new Path(args[1])); //Submitting

the job to the cluster and waiting for its completion job.waitForCompletion(true);

}}

### NewWordMapper.java

import java.io.IOException; import java.util.StringTokenizer; import

org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Mapp er;

public class NewWordMapper extends Mapper<LongWritable, Text, Text, IntWritable>

{

private final static IntWritable one = new IntWritable(1); private Text word = new Text();

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

IOException, InterruptedException

{

String line = value.toString(); StringTokenizer tokenizer = new StringTokenizer(line); while(tokenizer.hasMoreTokens())

{

word.set(tokenizer.nextToken()); context.write(word,one);

}}}

### NewWordReducer.java

import java.io.IOException; import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class NewWordReducer extends Reducer<Text, IntWritable, Text, IntWritable>

{

public void reduce(Text key, Iterable<IntWritable>values, Contextcontext) throws IOException, InterruptedException

{

int count = 0; for(IntWritable val :

values)

{

count += val.get();

}

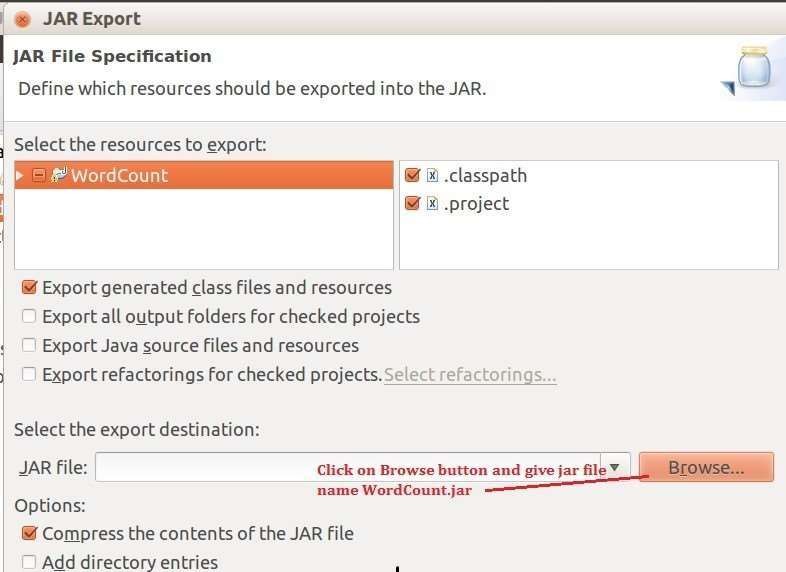
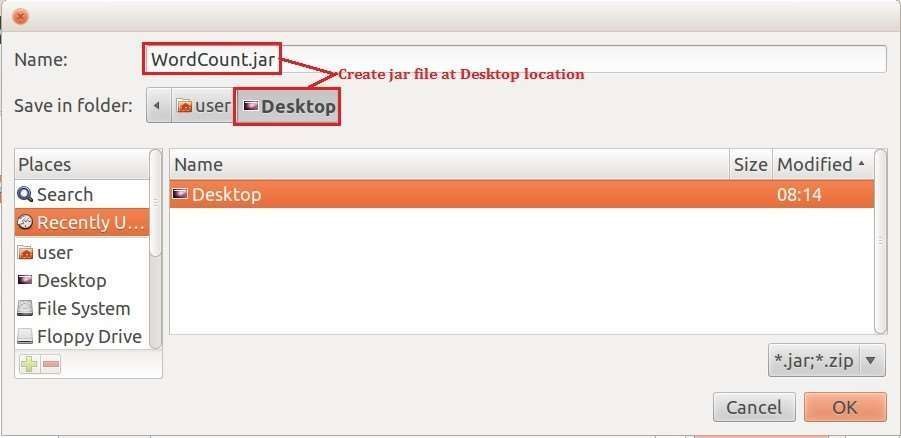
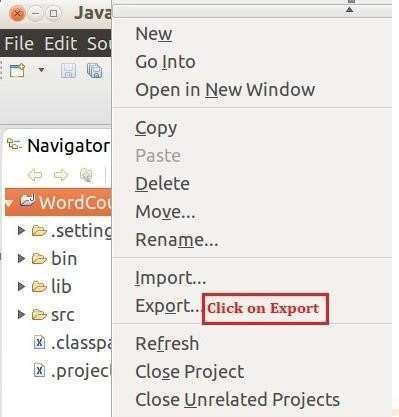
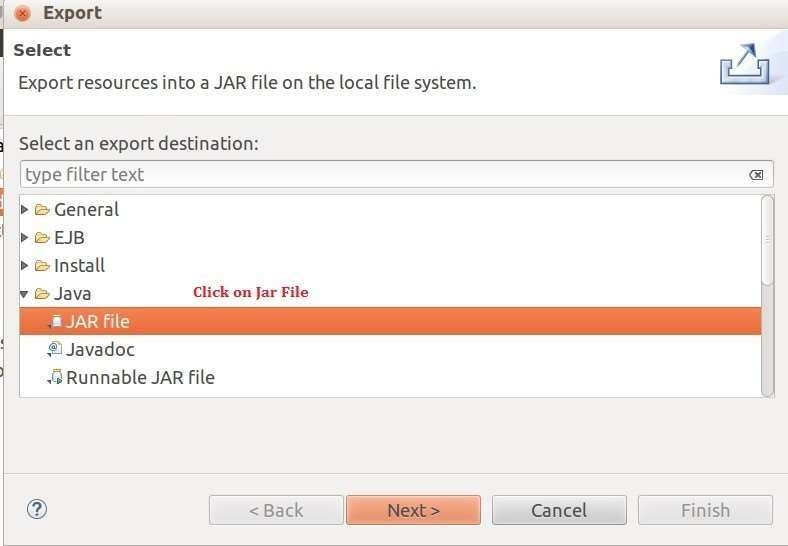
context.write(key, new IntWritable(count));

}}

### Step 9: Create Jar file Right click on

WordCountProject->Export->java->jar file>Browse->give jar WordCount.jar filename ->OK-

>finish.

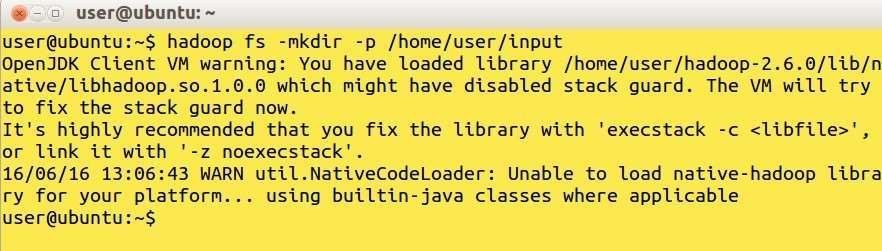


### Step 10: Create txt file name is input file

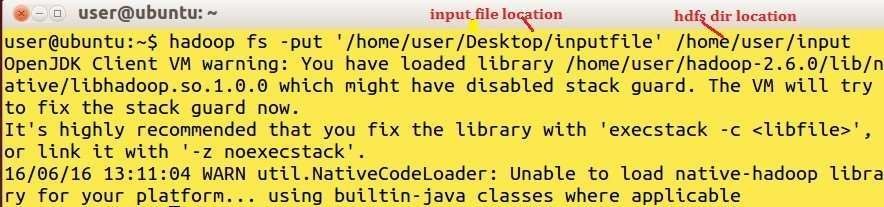
hi how are you how is your job how is your family how is your brother how is your sister what is the time now what is the strength

of Hadoop

### Step 11: create directory inside hdfs name is /home/user/input

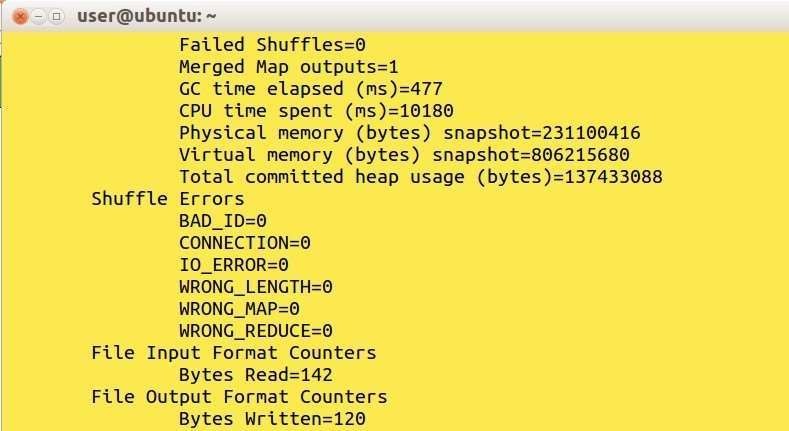


**Step 12: move inputfile.txt in hdfs /home/user/input director.**

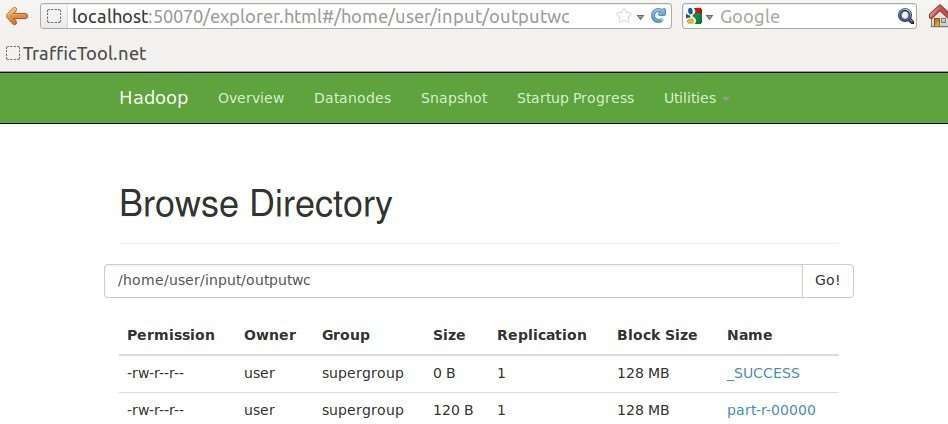


### Step 13: Run WordCount.jar file in HDFS .

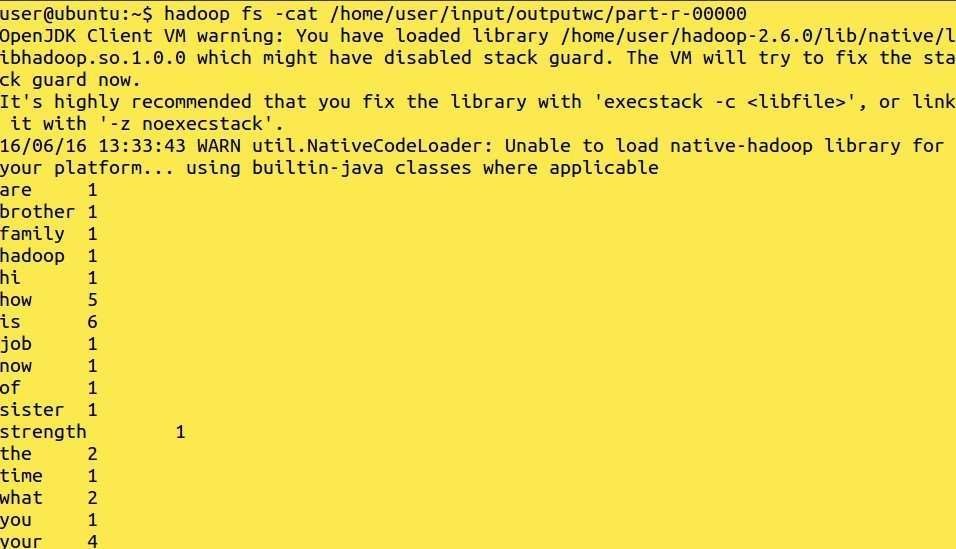
**Step 14: Console final output**



### Step 15: http://localhost:500070 Browser output.

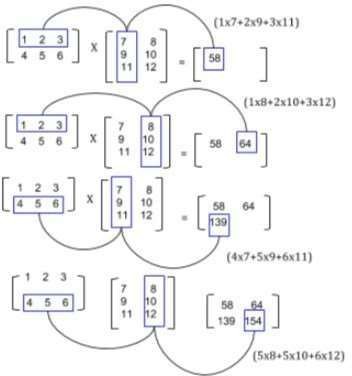


**Step 16: out file part-r-00000**



### Configuration Matrix Multiplication using MapReduce Programming

In mathematics, matrix multiplication or the matrix product is a binary operation that produces a matrix from two matrices. The definition is motivated by linear equations and linear transformations on vectors, which have numerous applications in applied

mathematics, physics, and engineering. In more detail, if A is an n × m matrix and B is an m × p matrix, their matrix product AB is an n × p matrix, in which the m entries across a row of A are multiplied with the m entries down a column of B and summed to produce an entry of AB. When two linear transformations are represented by matrices, then the matrix product represents the composition of the two transformations.

### Algorithm for Map Function.

for each element mij of M do produce (key, value) pairs as ((i, k), (M, j, mij), for k=1,2, 3, up to the number of columns of N

for each element njk of N do produce (key, value) pairs as ((i, k), (N, j, Njk), for I = 1,2, 3, Up to the number of rows of M.

return Set of (key, value) pairs that each key (i, k), has list with values (M, j, mij) and (N, j, njk) for all possible values of j.

### Algorithm for Reduce Function.

for each key (i, k) do

sort values begin with M by j in listM sort values begin with N by j in listN multiply mij and njk for jth value of each list sum up mij x njk return (i, k), Σj=1 mij x njk

### Step 1. Creating Mapper file for Matrix Multiplication.

package example;

**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**>**

### {

@Override

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**();**

// (M, i, j, Mij);

String**[]** indicesAndValue **=** line**.**split**(**","**);** Text outputKey **= new** Text**();**

Text outputValue **= new** Text**();**

**if (**indicesAndValue**[**0**].**equals**(**"M"**))**

### {

**for (**int k **=** 0**;** k **<** p**;** k**++)**

### {

outputKey**.**set**(**indicesAndValue**[**1**] +** "," **+** k**);**

// outputKey.set(i,k);

outputValue**.**set**(**indicesAndValue**[**0**] +** "," **+**indicesAndValue**[**2**]+** "," **+** indicesAndValue**[**3**]);**

// outputValue.set(M,j,Mij); context**.**write**(**outputKey**,** outputValue**);**

### }}

**else**

### {

// (N, j, k, Njk);

**for (**int i **=** 0**;** i **<** m**;** i**++)**

### {

outputKey**.**set**(**i **+** "," **+** indicesAndValue**[**2**]);**

outputValue**.**set**(**"N," **+** indicesAndValue**[**1**] +** ","**+** indicesAndValue**[**3**]);** context**.**write**(**outputKey**,** outputValue**);**

### }}}}

**Step 2. Creating Reducer.java file for Matrix Multiplication.**

package example;

**import** org**.**apache**.**hadoop**.**io**.**Text**; import** org**.**apache**.**hadoop**.**mapreduce**.**Reducer**; import** java**.**io**.**IOException**;**

**import** java**.**util**.**HashMap**;** public class Reduce

**extends** org**.**apache**.**hadoop**.**mapreduce**.**Reducer**<**Text**,** Text**,** Text**,** Text**>**

### {

@Override

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

IOException**,** InterruptedException

### {

String**[]** value**;**

//key=(i,k),

//Values = [(M/N,j,V/W),..]

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**]));**

### }}

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**)));**

### }}}

**Step 3. Creating MatrixMultiply.java file for**

package example;

**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**.**FileOutputForma t**; import** org**.**apache**.**hadoop**.**mapreduce**.**lib**.**output**.**TextOutputForm at**;** 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**();**

// M is an m-by-n matrix; N is an n-by-p matrix. conf**.**set**(**"m"**,** "1000"**);**

conf**.**set**(**"n"**,** "100"**);**

conf**.**set**(**"p"**,** "1000"**);** @SuppressWarnings**(**"deprecat ion"**)**

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**.**cla ss**);** FileInputFormat**.**addInputPath**(**job**, new** Path**(**args**[**0**]));** FileOutputFormat**.**setOutputPath**(**job**, new** Path**(**args**[**1**]));** job**.**waitForCompletion**(true);**

### }}

**Step 4. Compiling the program in particular folder named as operation**

$ javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce- client-core- 2.7.1.jar:operation/:. -d operation/ Map.java

$ javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce- client-core- 2.7.1.jar:operation/:. -d operation/ Reduce.java

$ javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce- client-core 2.7.1.jar:operation/:. -d operation/ MatrixMultiply.java

### Step 5. Let’s retrieve the directory after compilation.

$ ls -R operation/ operation/:

www operation/www:

ehadoopinfo operation/www/ehadoopinfo:

com operation/www/ehadoopinfo/com:

Map.class MatrixMultiply.class Reduce.class

### Step 6. Creating Jar file for the Matrix Multiplication.

$ jar -cvf MatrixMultiply.jar -C operation/ . added manifest

adding: www/(in = 0) (out= 0)(stored 0%) adding: www/ehadoopinfo/(in = 0) (out= 0)(stored 0%) adding: www/ehadoopinfo/com/(in = 0) (out= 0)(stored 0%) adding: www/ehadoopinfo/com/Reduce.class(in = 2919) (out= 1271)(deflated 56%) adding: www/ehadoopinfo/com/MatrixMultiply.class(in = 1815) (out= 932)(deflated 48%) adding: www/ehadoopinfo/com/Map.class(in = 2353) (out= 993)(deflated 57%)

### Step 7. Uploading the M, N file which contains the matrix multiplication data to HDFS.

$ cat M M,0,0, 1

M,0,1,2

M,1,0,3

M,1,1,4

$ cat N N,0,0, 5

N,0,1,6

N,1,0,7

N,1,1,8

$ hadoop fs -mkdir Matrix/

$ hadoop fs -copyFromLocal M Matrix/

$ hadoop fs -copyFromLocal N Matrix/

### Step 8. Executing the jar file using hadoop command and thus how fetching record from HDFS and storing output in HDFS.

$ hadoop jar MatrixMultiply.jar MatrixMultiply Matrix/\* result/ WARNING: Use "yarn jar" to launch YARN applications.

17/10/09 14:31:22 INFO impl.TimelineClientImpl: Timeline service address: http://sandbox.hortonworks.com:8188/ws/v1/timeline/

17/10/09 14:31:23 INFO client.RMProxy: Connecting to ResourceManager at sandbox.hortonworks.com/10.0.2.15:8050 17/10/09 14:31:23 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

17/10/09 14:31:24 INFO input.FileInputFormat: Total input paths to process : 2 17/10/09 14:31:24 INFO mapreduce.JobSubmitter: number of splits:2 17/10/09 14:31:24 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1507555978175\_0006

17/10/09 14:31:25 INFO impl.YarnClientImpl: Submitted application application\_1507555978175\_0006

17/10/09 14:31:25 INFO mapreduce.Job: The url to track the job: http://sandbox.hortonworks.com:8088/proxy/application\_1507555978175\_0 006/ 17/10/09 14:31:25 INFO mapreduce.Job: Running job: job\_1507555978175\_0006

17/10/09 14:31:35 INFO mapreduce.Job: Job job\_1507555978175\_0006 running in uber mode : false

17/10/09 14:31:35 INFO mapreduce.Job: map 0% reduce 0%

17/10/09 14:31:45 INFO mapreduce.Job: map 100% reduce 0%

17/10/09 14:31:53 INFO mapreduce.Job: map 100% reduce 100% 17/10/09 14:31:54 INFO mapreduce.Job: Job job\_1507555978175\_0006

completed successfully 17/10/09 14:31:55 INFO mapreduce.Job: Counters: 49 File System Counters

FILE: Number of bytes read=198 FILE: Number of bytes written=386063 FILE: Number of read operations=0

FILE: Number of large read operations=0 FILE: Number of write operations=0 HDFS: Number of bytes read=302 HDFS: Number of bytes written=36 HDFS:

Number of read operations=9 HDFS: Number of large read operations=0 HDFS: Number of write operations=2

Job Counters Launched map tasks=2 Launched reduce

tasks=1 Data-local map tasks=2

Total time spent by all maps in occupied slots (ms)=15088 Total time spent by all reduces in

occupied slots (ms)=6188 Total time spent by all map tasks (ms)=15088

Total time spent by all reduce tasks (ms)=6188 Total vcore-seconds taken by all map tasks=15088 Total vcore-seconds taken by all reduce tasks=6188

Total megabyte-seconds taken by all map tasks=3772000 Total megabyte-seconds taken by all reduce tasks=1547000 Map-Reduce Framework

Map input records=8 Map output records=16 Map output bytes=160

Map output materialized bytes=204 Input split bytes=238

Combine input records=0 Combine output records=0 Reduce input groups=4 Reduce shuffle bytes=204 Reduce input records=16 Reduce output records=4 Spilled Records=32 Shuffled Maps =2 Failed Shuffles=0 Merged Map outputs=2

GC time elapsed (ms)=196 CPU time spent (ms)=2720 Physical memory (bytes)

snapshot=536309760 Virtual memory (bytes) snapshot=2506076160 Total committed heap usage

(bytes)=360185856 Shuffle Errors BAD\_ID=0

CONNECTION=0 IO\_ERROR=0 WRONG\_LENGT H=0 WRONG\_MAP=0 WRONG\_REDUC E=0

File Input Format Counters Bytes Read=64

File Output Format Counters Bytes Written=36

### Step 10. Getting Output from part-r-00000 that was generated after the execution of the hadoop command.

$ hadoop fs -cat result/part-r- 00000 0,0,19.0

0,1,22.0

1,0,43.0

1,1,50.0

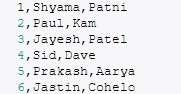
### Implementing simple algorithms in Map-Reduce.

1) **Join Algorithm**

UsersDetails.csv

This file contains the user related data, where data is present in following format,

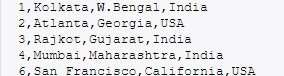
<UserID><FirstName><LastName>



AddressDetails.csv

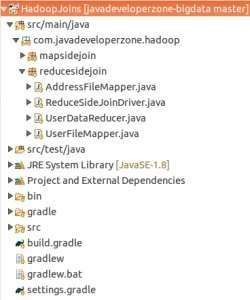
This file contains the User’s Address related data, where data is present in following format,

<UserID><City><State><Country>



We are going to use following 4 Java files for this algorithm,

1. AddressFileMapper.java
2. UserFileMapper.java
3. UserDataReducer.java
4. ReduceSideJoinDriver.java



### AddressFileMapper.java

package com.javadeveloperzone.hadoop.reducesidejoin; import java.io.IOException;

import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class AddressFileMapper extends Mapper<LongWritable, Text, LongWritable, Text>

{

private static final String fileTag = "AD~";

private static final String DATA\_SEPARATOR = ",";

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

{

String values[] = value.toString().split(DATA\_SEPARATOR); StringBuilder dataStringBuilder = new StringBuilder();

for (int index = 0; index < values.length; index++)

{

if (index != 0)

{

dataStringBuilder.append(values[index].toString().trim() + DATA\_SEPARATOR);

}

else

{

dataStringBuilder.append(fileTag);

}}

String dataString = dataStringBuilder.toString(); if (dataString != null && dataString.length()

> 1)

{

dataString = dataString.substring(0, dataString.length() - 1);

}

dataStringBuilder = null;

context.write(new LongWritable(Long.parseLong(values[0])), new Text(dataString));

}}

### UserFileMapper.java

package com.javadeveloperzone.hadoop.reducesidejoin; import java.io.IOException;

import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class UserFileMapper extends Mapper<LongWritable, Text, LongWritable, Text>

{

private static final String fileTag = "UD~";

private static final String DATA\_SEPARATOR = ",";

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

{

String values[] = value.toString().split(DATA\_SEPARATOR); StringBuilder dataStringBuilder = new StringBuilder();

for (int index = 0; index < values.length; index++)

{

if (index != 0)

{

dataStringBuilder.append(values[index].toString().trim() + DATA\_SEPARATOR);

}

else

{

dataStringBuilder.append(fileTag);

}}

String dataString = dataStringBuilder.toString(); if (dataString != null && dataString.length()

> 1)

{

dataString = dataString.substring(0, dataString.length() - 1);

}

dataStringBuilder = null;

context.write(new LongWritable(Long.parseLong(values[0])), new Text(dataString));

}}

### ReduceSideJoinDriver.java

package com.javadeveloperzone.hadoop.reducesidejoin; import org.apache.hadoop.conf.Configuration; import

org.apache.hadoop.conf.Configured; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.MultipleInputs; import org.apache.hadoop.mapreduce.lib.input.TextInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputForm

at; import org.apache.hadoop.mapreduce.lib.output.TextOutputForm at; import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class ReduceSideJoinDriver extends Configured implements Tool

{

private static final String DATA\_SEPARATOR = ","; public int run(String[] args) throws Exception

{

Configuration configuration = new Configuration(); configuration.set("mapreduce.output.textoutputformat.separator", DATA\_SEPARATOR);

Job job = Job.getInstance(configuration); job.setJobName("Reduce Side Join Mapreduce example using Java"); job.setJarByClass(ReduceSideJoinDriver.class);

// Map job.setMapOutputKeyClass(LongWritable

.class); job.setMapOutputValueClass(Text.class);

// Job job.setOutputKeyClass(LongWritable.clas s); job.setOutputValueClass(Text.class); job.setInputFormatClass(TextInputFormat

.class); job.setOutputFormatClass(TextOutputForma

t.class); job.setReducerClass(UserDataReducer.class)

;

MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, UserFileMapper.class); MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, AddressFileMapper.class); FileOutputFormat.setOutputPath(job, new Path(args[2]));

job.waitForCompletion(t rue); return 0;

}

public static void main(String[] args) throws Exception

{

if (args.length == 3)

{

int result = ToolRunner.run(new Configuration(), new

ReduceSideJoinDriver(), args); if (0 == result)

{

System.out.println("Reduce Side Join Mapreduce example using Java Job Completed Successfully...");

}

else

{

System.out.println("Reduce Side Join Mapreduce example using Java Job Failed...");

}}

else

{

System.out.println("USAGE <InputPath1><InputPath2><OutputPath>");

}}}

### UserDataReducer.java

package com.javadeveloperzone.hadoop.reducesidejoin; import java.io.IOException;

import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class UserDataReducer extends Reducer<LongWritable, Text, LongWritable, Text>

{

public static final String TAG\_SEPARATOR = "~";

private static final String DATA\_SEPARATOR = ","; @Override

protected void reduce(LongWritable key, Iterable<Text> values,

Reducer<LongWritable, Text, LongWritable, Text>.Context context) throws IOException, InterruptedException

{

String value;

String[] spllitedValues; String tag;

String data = null, userDetails = null, addressDetails = null; for (Text txtValue : values)

{

value = txtValue.toString();

spllitedValues = value.split(TAG\_SEPARATOR); tag = spllitedValues[0];

if (tag.equalsIgnoreCase("UD"))

{

userDetails = spllitedValues[1];

}

else if (tag.equalsIgnoreCase("AD"))

{

addressDetails = spllitedValues[1];

}}

if (userDetails != null && addressDetails != null)

{

data = userDetails + DATA\_SEPARATOR + addressDetails;

}

else if (userDetails == null)

{

data = addressDetails;

}

else if (addressDetails == null)

{

data = userDetails;

}

context.write(key, new Text(data));

}}

### Copy files from local file system to HDFS

hdfs dfs -copyFromLocal 4-UserDetails.csv

/input/javadeveloperzone/reducesidejoin/ hdfs dfs -copyFromLocal 4- AddressDetails.csv /input/javadeveloperzone/reducesidejoin/

### Build & Run Application

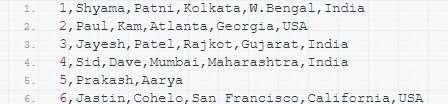
hadoop jar HadoopJoins.jar com.javadeveloperzone.hadoop.reducesidejoin.ReduceSideJoinDriver

/input/javadeveloperzone/reducesidejoin/4-UserDetails.csv

/input/javadeveloperzone/reducesidejoin/4- AddressDetails.csv

/output/javadeveloperzone/hadoop/reducesidejoin

**OUTPUT: -**



**2) Sort Algorithm**

### Input (test.txt)

1,50

2,20

3,30

4,10

5,15

6,25

7,55

8,35

9,70

### Sort.java

package com.my.cert.example; import java.nio.ByteBuffer; import

org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.IntWritable; import

org.apache.hadoop.io.IntWritable.Comparator

; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.WritableComparator; 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.input.TextInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputForm at;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFor

mat; public class ValueSortExp

{

public static void main(String[] args) throws Exception

{

Path inputPath = new Path("C:\\hadoop\\test\\test.txt"); Path outputDir = new Path("C:\\hadoop\\test\\test1");

// Path inputPath = new Path(args[0]);

// Path outputDir = new Path(args[1]);

// Create configuration

Configuration conf = new Configuration(true);

// Create job

Job job = new Job(conf, "Test HIVE commond"); job.setJarByClass(ValueSortExp.class);

// Setup MapReduce job.setMapperClass(ValueSortExp.MapTask.cl ass); job.setReducerClass(ValueSortExp.ReduceTas k.class); job.setNumReduceTasks(1);

// Specify key / value job.setMapOutputKeyClass(IntWritable.cla ss); job.setMapOutputValueClass(IntWritable.c lass); job.setOutputKeyClass(IntWritable.class); job.setOutputValueClass(IntWritable.class)

;

job.setSortComparatorClass(IntComparato r.class);

// Input

FileInputFormat.addInputPath(job, inputPath); job.setInputFormatClass(TextInputFormat

.class);

// Output FileOutputFormat.setOutputPath(job, outputDir); job.setOutputFormatClass(TextOutputForma t.class);

// Execute job

int code = job.waitForCompletion(true)

? 0 : 1; System.exit(code);

}

### MapTask.java

public static class MapTask extends Mapper<LongWritable, Text, IntWritable, IntWritable>

{

public void map(LongWritable key, Text value, Context context)throws java.io.IOException, InterruptedException

{

String line = value.toString();

String[] tokens = line.split(","); // This is the delimiter between int keypart = Integer.parseInt(tokens[0]);

int valuePart = Integer.parseInt(tokens[1]);

context.write(new IntWritable(valuePart), new IntWritable(keypart));

}}

### ReduceTask.java

public static class ReduceTask extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable>

{

public void reduce(IntWritable key, Iterable<IntWritable> list, Context context) throws java.io.IOException, InterruptedException

{

for (IntWritable value : list)

{

context.write(value,key);

}}}

### IntComparator.java

public static class IntComparator extends WritableComparator

{

public IntComparator()

{

super(IntWritable.class);

}

@Override

public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)

{

Integer v1 = ByteBuffer.wrap(b1, s1,

l1).getInt(); Integer v2 = ByteBuffer.wrap(b2, s2, l2).getInt(); return v1.compareTo(v2) \* (-1);

}}

**OUTPUT: -**

9 70

7 55

1 50

8 35

3 30

6 25

2 20

5 15

4 10

# Practical 6

## AIM: Implementing any one Frequent Itemset algorithm using Map- Reduce.

### CODE -

**Mapper Function**

**Map Phase** input: < k1, v1 >

k1 - Line no v1 - Transaction // get items from each transaction //item count set to 1 for each item k2-item v2 -1 End for

Output(k2, v2)

### Reducer Function

//Count the occurrences f or each item

// minimum support

**Reduce Phase** input: < k2, List < v2 >> sum the value for each item occurrence

if (occurrence of an item satisfy minsup)

Em it(Frequent item) k3 - Frequent item v3 - occurrences Output: < k3, v3 >

import java.io.BufferedReader; import java.io.\*; import java.io.IOException; import java.net.\*; import java.util.ArrayList; import java.util.\*; import model.HashTreeNode; import model.ItemSet;

import model.Transaction;

import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.conf.Configured; import

org.apache.hadoop.filecache.DistributedCache

; 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.FileOutputForm at; import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner; import utils.AprioriUtils;

import utils.HashTreeUtils; import org.apache.hadoop.fs.\*;

/\*\* A parallel hadoop-based Apriori algorithm \*/ public class MRApriori extends Configured implements Tool {

private static String jobPrefix = "MRApriori Algorithm Phase ";

// TODO : This is bad as I using a global shared variable between functions which should

// ideally be a function parameter. Need to fix this later. These parameters are required in

// reducer logic and have to be dynamica. How can I pass some initialisation parameters to

// reducer ?

public int run(String[] args) throws IOException, InterruptedException, ClassNotFoundException { if (args.length

!= 5) {

System.err.println("Incorrect number of command line args. Exiting!!");

return -1;

}

String hdfsInputDir = args[0];

String hdfsOutputDirPrefix = args[1];

int maxPasses = Integer.parseInt(args[2]);

Double MIN\_SUPPORT\_PERCENT = Double.parseDouble(args[3]); Integer MAX\_NUM\_TXNS = Integer.parseInt(args[4]); System.out.println("InputDir : " + hdfsInputDir);

System.out.println("OutputDir Prefix : " +

hdfsOutputDirPrefix); System.out.println("Number of Passes

: " + maxPasses); System.out.println("MinSupPercent : " + MIN\_SUPPORT\_PERCENT); System.out.println("Max Txns

: " + MAX\_NUM\_TXNS);

long startTime = System.currentTimeMillis(); long endTime = System.currentTimeMillis();

for (int passNum = 1; passNum <= maxPasses; passNum++) { endTime = System.currentTimeMillis();

boolean isPassKMRJobDone = runPassKMRJob(hdfsInputDir,

hdfsOutputDirPrefix, passNum, MIN\_SUPPORT\_PERCENT, MAX\_NUM\_TXNS); if (!isPassKMRJobDone) {

System.err.println("Phase1 MapReduce job failed. Exiting!!");

return -1;

}

System.out.println("For pass " + passNum + " = " + (System.currentTimeMillis() - endTime));

}

endTime = System.currentTimeMillis();

System.out.println("Total time taken = " + (endTime - startTime));

return 1;

}

private static boolean runPassKMRJob(String hdfsInputDir, String hdfsOutputDirPrefix, int passNum, Double MIN\_SUPPORT\_PERCENT,

Integer MAX\_NUM\_TXNS)

throws IOException,

InterruptedException, ClassNotFoundException { boolean isMRJobSuccess = false;

Configuration passKMRConf = new Configuration(); passKMRConf.setInt("passNum", passNum); passKMRConf.set("minSup", Double.toString(MIN\_SUPPORT\_PERCENT)); passKMRConf.setInt("numTxns", MAX\_NUM\_TXNS); System.out.println("Starting AprioriPhase" + passNum + "Job"); if (passNum > 1) {

DistributedCache.addCacheFile(URI.create("hdfs://127.0.0.1

:54310" + hdfsOutputDirPrefix +

(passNum - 1) + "/part-r-00000"), passKMRConf);

System.out.println("Added to distributed cache the output of pass " + (passNum-1));

}

\*/

Job aprioriPassKMRJob = new Job(passKMRConf, jobPrefix + passNum);

if (passNum == 1) { configureAprioriJob(aprioriPassK MRJob, AprioriPass1Mapper.class);

} else { configureAprioriJob(aprioriPassK MRJob, AprioriPassKMapper.class);

}

FileInputFormat.addInputPath(aprioriPassKMRJob, new Path(hdfsInputDir)); System.out.println("saurabh " + new Path(hdfsInputDir)); FileOutputFormat.setOutputPath(aprioriPassKMRJob, new Path(hdfsOutputDirPrefix

+

passNum));

isMRJobSuccess = (aprioriPassKMRJob.waitForCompletion(true) ? true : false); System.out.println("Finished AprioriPhase" + passNum + "Job");

return isMRJobSuccess;

}

@SuppressWarnings({ "unchecked", "rawtypes"

})

private static void configureAprioriJob(Job aprioriJob, Class mapperClass) { aprioriJob.setJarByClass(MRApriori.class); aprioriJob.setMapperClass(mapperClass); aprioriJob.setReducerClass(AprioriReducer.class); aprioriJob.setOutputKeyClass(Text.class); aprioriJob.setOutputValueClass(IntWritable.class);

}

// Utility functions

// Phase1 - MapReduce

public static class AprioriPass1Mapper extends Mapper < Object, Text, Text, IntWritable > {

private final static IntWritable one = new IntWritable(1); private Text item = new Text(); public void map(Object key, Text txnRecord, Context context) throws IOException,

InterruptedExcepti on { Transaction txn =

AprioriUtils.getTransaction(txnRecord.toSt ring()); for (Integer itemId: txn.getItems()) { item.set(itemId.toString()); context.write(item, one);

}

}

}

public static class AprioriReducer extends Reducer < Text, IntWritable, Text, IntWritable > {

public void reduce(Text itemset, Iterable < IntWritable > values, Context context) throws IOException,

InterruptedExcepti on { int countItemId = 0;

for (IntWritable value: values) { countItemId += value.get();

}

// TODO : This can be improved. Creating too many strings.String itemsetIds = itemset.toString(); itemsetIds = itemsetIds.replace("[", "");

itemsetIds =

itemsetIds.replace("]", ""); itemsetIds = itemsetIds.replace(" ", ""); Double minSup =

Double.parseDouble(context.getConfiguration().get(" minSup")); Integer numTxns = context.getConfiguration().getInt("numTxns", 2);

//System.out.println("dsfsdfsdf: " + MIN\_SUPPORT\_PERCENT

+

" " + MAX\_NUM\_TXNS);

// If the item has minSupport, then it is a large itemset.

if (AprioriUtils.hasMinSupport(minSup, numTxns, countItemId)) { context.write(new Text(itemsetIds), new IntWritable(countItemId));

}

}

}

// Phase2 - MapReduce

public static class AprioriPassKMapper extends Mapper < Object, Text, Text, IntWritable > {

private final static IntWritable one = new IntWritable(1); private Text item = new Text(); private List < ItemSet >

largeItemsetsPrevPass = new ArrayList < ItemSet > ();

private List < ItemSet > candidateItemsets = null;private HashTreeNode hashTreeRootNode = null;

@Override

public void setup(Context context) throws IOException {

//Path[] uris = DistributedCache.getLocalCacheFiles(context.getConfig uration()); int passNum = context.getConfiguration().getInt("passNum",

2);

String opFileLastPass = context.getConfiguration().get("fs.default.name") +

"/user/hduser/mrapriori-out-" + (passNum - 1) + "/part-r-00000";

//System.out.println("ahsdkjdsgfjhgf"+opFileLastPass);

//System.out.println("Distributed cache file to search " + opFileLastPass);

try {

Path pt = new Path(opFileLastPass);

FileSystem fs = FileSystem.get(context.getConfiguration()); BufferedReader fis = new BufferedReader(new InputStreamReader(fs.open(pt))); String currLine = null;

//System.out.println("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaa aaaa ");

while ((currLine = fis.readLine()) != null) { currLine = currLine.trim(); String[] words = currLine.split("[\\s\\t]+"); if (words.length < 2) {

continue;

}

List < Integer > items = new ArrayList < Integer > (); for (int k = 0; k < words.length - 1; k++) {

String csvItemIds = words[k];

String[] itemIds = csvItemIds.split(","); for (String itemId: itemIds) {

items.add(Integer.parseInt(itemId));

}

}

String finalWord = words[words.length - 1];

int supportCount = Integer.parseInt(finalWord);

//System.out.println(items + " --> " + supportCount); largeItemsetsPrevPass.add(new ItemSet(items, supportCount));

}

}

catch (Exception e) {

}

candidateItemsets = AprioriUtils.getCandidateItemsets(largeItemsetsPrevPass, (passNum - 1)); hashTreeRootNode =

HashTreeUtils.buildHashTree(candidateItemsets, passNum); // This would be changed later

}

public void map(Object key, Text txnRecord, Context context) throws IOException, InterruptedException { Transaction txn =

AprioriUtils.getTransaction(txnRecord.toStr ing()); List < ItemSet > candidateItemsetsInTxn =

HashTreeUtils.findItemsets(hashTreeRootNode

, txn, 0); for (ItemSet itemset: candidateItemsetsInTxn) { item.set(itemset.getItems().toString()); context.write(item, one);

}

}

}

public static void main(String[] args) throws Exception { int exitCode = ToolRunner.run(new MRApriori(), args); System.exit(exitCode);

}

}

**OUTPUT:**

|  |  |  |
| --- | --- | --- |
| 1) OUTPUT1) |  | |
|  | 2 | 3 |
|  | 3 | 3 |
|  | 5 | 3 |

# Practical 7

## AIM: Implementing any one Clustering algorithm using Map‐

**Reduce**

### Mapper Class:

@Override

protected void map (Cluster Center key, Vector value, Context context) throws IO Exception, Interrupted Exception {

Cluster Center nearest = null;

double nearestDistance = Double.MAX\_VALUE; for (Cluster Center c : centers) {

double dist = DistanceMeasurer.measureDistance(c, value); if (nearest == null) {

nearest = c; nearestDistance = dist;

} else {

if (nearestDistance > dist) { nearest = c; nearestDistance = dist;

}

}

}

context.write(nearest, value);

### Reducer Class:

@Override

protected void reduce(ClusterCenter key, Iterable<Vector> values, Context context) throws IOException, InterruptedException { Vector newCenter = new Vector();

List<Vector> vectorList = new LinkedList<Vector>(); int vectorSize = key.getCenter().getVector().length; newCenter.setVector(new double[vectorSize]);

for (Vector value : values) { vectorList.add(new Vector(value));

for (int i = 0; i < value.getVector().length; i++) { newCenter.getVector()[i] += value.getVector()[i];

}

}

for (int i = 0; i < newCenter.getVector().length; i++) { newCenter.getVector()[i] = newCenter.getVector()[i]

/ vectorList.size();

}

ClusterCenter center = new ClusterCenter(newCenter); centers.add(center);

for (Vector vector :

vectorList) { context.write(center, vector);

}

if (center.converged(key)) context.getCounter(Counter.CONVERGED).incre ment(1);

}

### Vector Class:

public class Vector implements WritableComparable<Vector> { private double[] vector;

public Vector() {

super();

}

public Vector(Vector v) { super();

int l = v.vector.length; this.vector = new double[l];

System.arraycopy(v.vector, 0, this.vector, 0, l);

}

public Vector(double x, double y) { super();

this.vector = new double[] { x, y };

}

@Override

public void write(DataOutput out) throws IOException { out.writeInt(vector.length); for (int i = 0; i < vector.length;

i++)

out.writeDouble(vector[i]);

}

@Override

public void readFields(DataInput in) throws IOException { int size = in.readInt(); vector = new

double[size]; for (int i = 0; i < size; i++) vector[i] = in.readDouble();

}

@Override

public int compareTo(Vector o) {

boolean equals = true;

for (int i = 0; i < vector.length; i++) { int c = vector[i] - o.vector[i];

if (c != 0.0d) {

return c;

}

return 0;

}

}

### DistanceMeasurer.java

public static final double measureDistance(ClusterCenter center, Vector v) { double sum = 0;

int length = v.getVector().length; for (int i = 0; i < length; i++) {

sum += Math.abs(center.getCenter().getVector()[i]

- v.getVector()[i]);

}

return sum;

}

### Output:

**Input K-Centers:** (1,1);(5,5)

### Input:

Vector [vector=[16.0, 3.0]]

Vector [vector=[7.0, 6.0]]

Vector [vector=[6.0, 5.0]]

Vector [vector=[25.0, 1.0]]

Vector [vector=[1.0, 2.0]]

Vector [vector=[3.0, 3.0]]

Vector [vector=[2.0, 2.0]]

Vector [vector=[2.0, 3.0]]

Vector [vector=[-1.0, -23.0]]

### Output:

ClusterCenter [center=Vector [vector=[13.5, 3.75]]] / Vector [vector=[16.0, 3.0]] ClusterCenter [center=Vector [vector=[13.5, 3.75]]] / Vector [vector=[7.0, 6.0]] ClusterCenter [center=Vector [vector=[13.5, 3.75]]] / Vector [vector=[6.0, 5.0]] ClusterCenter [center=Vector [vector=[13.5, 3.75]]] / Vector [vector=[25.0, 1.0]] ClusterCenter [center=Vector [vector=[1.4, -2.6]]] / Vector [vector=[1.0, 2.0]] ClusterCenter [center=Vector [vector=[1.4, -2.6]]]

/ Vector [vector=[3.0, 3.0]] ClusterCenter [center=Vector [vector=[1.4, -2.6]]] / Vector [vector=[2.0, 2.0]] ClusterCenter [center=Vector [vector=[1.4, -2.6]]] / Vector [vector=[2.0, 3.0]] ClusterCenter [center=Vector [vector=[1.4, -2.6]]] / Vector [vector=[-1.0, -23.0]]

**Practical 8**

**AIM: Implementing any one data streaming**

**algorithm using Map‐Reduce.**

**PartitionerExample.java** package partitionerexample; import java.io.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*; import org.apache.hadoop.conf.\*; import org.apache.hadoop.conf.\*; import org.apache.hadoop.fs.\*;

import org.apache.hadoop.mapreduce.lib.input.\*; import org.apache.hadoop.mapreduce.lib.output.\*;

import org.apache.hadoop.util.\*;

public class PartitionerExample extends Configured implements Tool

{

//Map class

public static class MapClass extends Mapper<LongWritable,Text,Text,Text>

{

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

{

try{

String[] str = value.toString().split("\t", -3); String gender=str[3];

context.write(new Text(gender), new Text(value));

}

catch(Exception e)

{

System.out.println(e.getMessage());

}

}

}

//Reducer class

public static class ReduceClass extends Reducer<Text,Text,Text,IntWritable>

{

public int max = -1;

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

{

max = -1;

for (Text val : values)

{

String [] str = val.toString().split("\t", -3); if(Integer.parseInt(str[4])>max) max=Integer.parseInt(str[4]);

}

context.write(new Text(key), new IntWritable(max));

}

}

//Partitioner class

public static class CaderPartitioner extends Partitioner < Text, Text >

{

@Override

public int getPartition(Text key, Text value, int numReduceTasks)

{

String[] str = value.toString().split("\t"); int age = Integer.parseInt(str[2]);

if(numReduceTasks == 0)

{

return 0;

}

if(age<=20)

{

return 0;

}

else if(age>20 && age<=30)

{

return 1 % numReduceTasks;

}

else

{

return 2 % numReduceTasks;

}

}

}

@Override

public int run(String[] arg) throws Exception

{

Configuration conf = getConf();

Job job = new Job(conf, "topsal"); job.setJarByClass(PartitionerExample.class);

FileInputFormat.setInputPaths(job, new Path(arg[0])); FileOutputFormat.setOutputPath(job,new Path(arg[1]));

job.setMapperClass(MapClass.class);

job.setMapOutputKeyClass(Text.class); job.setMapOutputValueClass(Text.class);

//set partitioner statement

job.setPartitionerClass(CaderPartitioner.class); job.setReducerClass(ReduceClass.class); job.setNumReduceTasks(3); job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class); job.setOutputKeyClass(Text.class); job.setOutputValueClass(Text.class);

System.exit(job.waitForCompletion(true)? 0 : 1);

return 0;

}

public static void main(String ar[]) throws Exception

{

int res = ToolRunner.run(new Configuration(), new PartitionerExample(),ar);

System.exit(0);

}

}

**input.txt**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1201 | Gopal | 45 | Male | 50000 |
| 1202 | Manisha | 40 | Female | 51000 |
| 1203 | Khaleel | 34 | Male | 30000 |
| 1204 | Prasanth | 30 | Male | 31000 |
| 1205 | Kiran | 20 | Male | 40000 |
| 1206 | Laxmi | 25 | Female | 35000 |
| 1207 | Bhavya | 20 | Female | 15000 |
| 1208 | Reshma | 19 | Female | 14000 |
| 1209 | kantha | 22 | Male | 22000 |
| 1210 | Satish | 24 | Male | 25000 |
| 1211 | Krishna | 25 | Male | 26000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1212 | Arshad | 28 | Male | 20000 |
| 1213 | Lavanya | 18 | Female | 8000 |

**Output:**

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000

**Output in Part-00000**

Female 15000

Male 40000

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00001

**Output in Part-00001**

Female 35000

Male 31000

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00002

**Output in Part-00002**

Female 51000

Male 50000