**Write a program in Map Reduce for Matrix Multiplication**

Step 1: Open virtual box and then start cloudera quickstart

Step 2: Open eclipse present on the cloudera desktop

Step 3: Create java project

File->New-> Java Project

Give project name: MatrixMultiplication

Click Next

Step 4: Add Hadoop libraries to project

Select Libraries tab->click on Add Externa Jars

File System->user->lib->Hadoop

Select all library(jar) files->ok

Again click on Add Externa Jars

File System->user->lib->Hadoop->client

Select all library(jar) files from client->ok->finish

Step 5: Write java code for matrix multiplication using mapreduce

Right click on src folder of project MatrixMultiplication

New->class

Write class name MatrixMultiplicationDriver

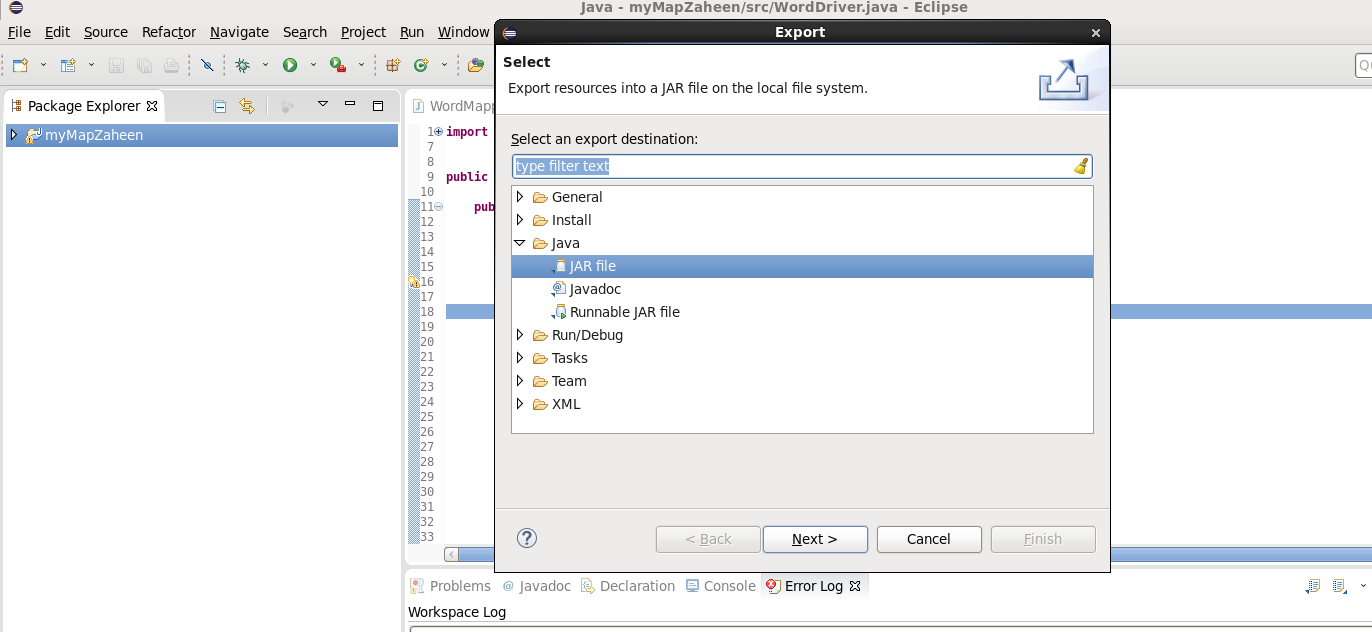
Click Finish

Write the code for MatrixMultiplicationDriver

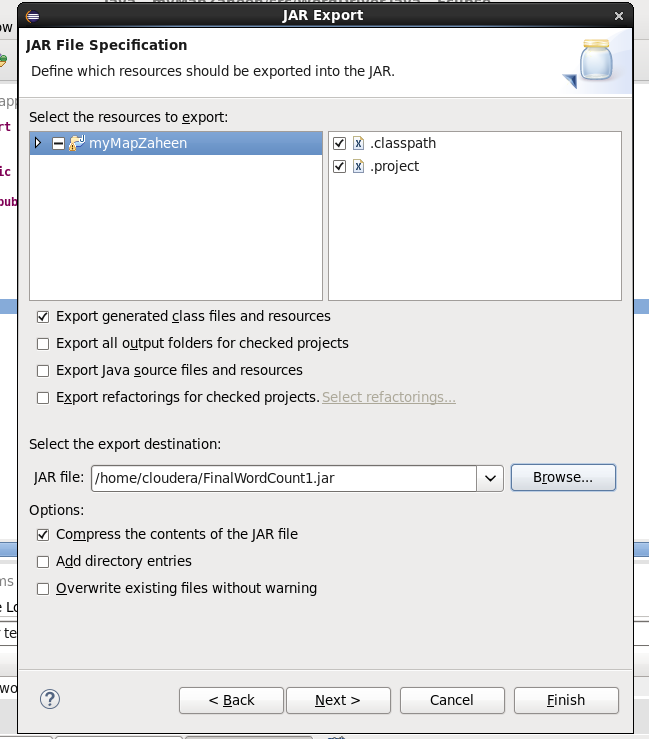
Same way create classes MatrixMultiplicationMapper and MatrixMultiplicationReducer

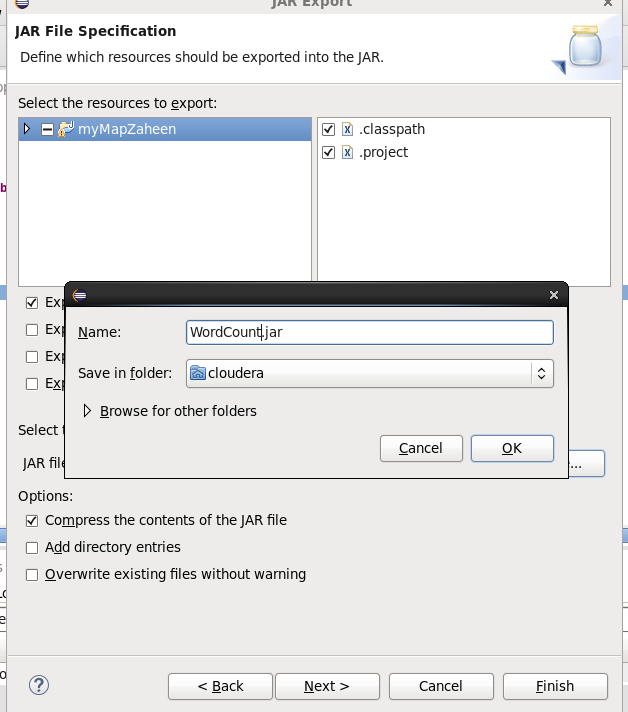
Step 6: Export the project as jar

Right click on project MatrixMultiplication and select Export>> Java>>JAR file>>Next



Select the export destination-Click browse-give file name matrixmultiplication.jar Click ok>>Finish>>ok





Verify the jar file

Verify the jar file through command line open terminal give command

**ls**

step 7: Move the jar file to the Hadoop file system

**hdfs dfs -put matrixmultiplication.jar /user/cloudera**

**hdfs dfs -ls**

Step 8: Create the input file for the MapReduce program

First input file

Command: **cat > myMMatrix.txt**

Enter data in input file

M,0,0,1

M,0,1,2

M,1,0,3

M,1,1,4

and press enter and ctrl z

Command: **cat myMMatrix.txt**

Second input file

Command: **cat > myNMatrix.txt**

Enter data in input file

N,0,0,3

N,0,1,6

N,1,0,4

N,1,1,2

and press enter and ctrl z

Command: **cat myNMatrix.txt**

Step 9: Create the input directory and move input files into it.

**hdfs dfs –mkdir /user/cloudera/matrixinput**

**hdfs dfs -put myMMatrix.txt /user/cloudera/matrixinput/**

**hdfs dfs -put myNMatrix.txt /user/cloudera/matrixinput/**

**hdfs dfs -ls**

step 10: Run mapreduce program on Hadoop

syntax: hadoop jar jarfilename.jar classname inputfoldername outputfoldername

command: **hadoop jar matrixmultiplication.jar MatrixMultiplicationDriver matrixinput matrixOutput**

step 11: view output directory

**hdfs dfs –ls**

**hdfs dfs -ls /user/cloudera/matrixOutput**

step 12: view the output file

**hdfs dfs -cat /user/cloudera/matrixOutput/part-r-00000**

Code

**MatrixMultiplicationDriver.java**

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

import org.apache.hadoop.fs.Path;

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

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

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MatrixMultiplicationDriver {

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

// TODO Auto-generated method stub

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");

Job job = new Job(conf, "MatrixMultiply");

job.setJarByClass(MMDriver.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(MMMapper.class);

job.setReducerClass(MMReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.waitForCompletion(true);

}

}

**MatrixMultiplicationMapper.java**

import java.io.IOException;

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

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class MatrixMultiplicationMapper extends 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);

}

}

}

}

**MatrixMultiplicationReducer.java**

import org.apache.hadoop.io.Text;

import java.io.IOException;

import java.util.HashMap;

public class MatrixMultiplicationReducer 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)));

}

}

}