**Big Data Analytics**

**BAD601**

**Experiment 2:**

Develop a MapReduce program to implement Matrix Multiplication

**Open Notepad and write**

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.HashMap;

public class MatrixMultiplication {

// Mapper Class

public static class MatrixMapper extends Mapper<Object, Text, Text, Text> {

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

String[] tokens = value.toString().split(",");

String matrix = tokens[0]; // "A" or "B"

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

int col = Integer.parseInt(tokens[2]);

int val = Integer.parseInt(tokens[3]);

if (matrix.equals("A")) {

for (int k = 0; k < 2; k++) { // Assuming B has 2 columns

context.write(new Text(row + "," + k), new Text("A," + col + "," + val));

}

} else {

for (int i = 0; i < 2; i++) { // Assuming A has 2 rows

context.write(new Text(i + "," + col), new Text("B," + row + "," + val));

}

}

}

}

// Reducer Class

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

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

HashMap<Integer, Integer> A = new HashMap<>();

HashMap<Integer, Integer> B = new HashMap<>();

for (Text val : values) {

String[] parts = val.toString().split(",");

int index = Integer.parseInt(parts[1]);

int value = Integer.parseInt(parts[2]);

if (parts[0].equals("A")) {

A.put(index, value);

} else {

B.put(index, value);

}

}

int sum = 0;

for (int j : A.keySet()) {

if (B.containsKey(j)) {

sum += A.get(j) \* B.get(j);

}

}

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

}

}

// Driver Class

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Matrix Multiplication");

job.setJarByClass(MatrixMultiplication.class);

job.setMapperClass(MatrixMapper.class);

job.setReducerClass(MatrixReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

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

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

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

}

}

**Save the Program:**

Click of Save and save the program with the name MatrixMultiplication.java

**Make two text file and save as matrixA.txt, matrixB.txt:**

A,0,0,1

A,0,1,2

A,1,0,3

A,1,1,4

B,0,0,5

B,0,1,6

B,1,0,7

B,1,1,8

**Change the directory from cmd:**

Go to the directory where the code is saved using cd

**Get the Hadoop class path:**

hadoop classpath

**Compile the Java File**

Now, compile your Java program using the copied classpath:

javac -classpath "PASTE\_HADOOP\_CLASSPATH\_HERE" -d . MatrixMultiplication.java

**Create a JAR File**

Once compilation is successful, create the JAR file:

jar cf matrixmultiplication.jar \*.class

**Upload Input Files to HDFS**

start-dfs.cmd

start-yarn.cmd

hdfs dfs -mkdir /matrix

hdfs dfs -put matrixA.txt /matrix/

hdfs dfs -put matrixB.txt /matrix/

**Run the Hadoop Job**

hadoop jar matrixmultiplication.jar MatrixMultiplication /matrix /output

**View the Output**

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