**Pranay Takkallapelly**

**700645489**

**Ranjitha Gade**

**700639359**

**Bigdata:Storage,Analytc &visl**

**MidTerm-1**

**Java program to implement a map reduce version of the two job matrix multiplication**

**FirstMapper.java**

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.fs.Path;

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

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

import org.apache.hadoop.io.Text;

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 FirstMapper extends Mapper<LongWritable, Text, Text, Text> {

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

String line = value.toString();

String[] indicesAndValue = line.split(",");

//int length = indicesAndValue.length();

//Text outputKey = new Text();

//Text outputValue = new Text();

if(indicesAndValue[0].equals("A"))

{

String toBuildValue= indicesAndValue[0]+","+indicesAndValue[1]+","+indicesAndValue[3];

context.write(new Text(indicesAndValue[2]), new Text(toBuildValue));

}

else

{

String toBuildValue= indicesAndValue[0]+","+indicesAndValue[2]+","+indicesAndValue[3];

context.write(new Text(indicesAndValue[1]), new Text(toBuildValue));

}

}

}

**FirstReducer.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 FirstReducer

extends Reducer<Text, Text, Text, Text> {

@Override

public void reduce(Text key, Iterable<Text> values,

Context context)

throws IOException, InterruptedException {

String firstMatrix[] = new String[100];

String secondMatrix[] = new String[100];

int firstIndex=0,

secondIndex=0;

for (Text val : values) {

String str = val.toString();

if (str.contains("A"))

{

firstMatrix[firstIndex] = val.toString();

firstIndex++;

}

else

{

secondMatrix[secondIndex] = val.toString();

secondIndex++;

} }

for (int k=0; k<firstIndex; k++)

{

String[] insideFirst=firstMatrix[k].split(",");

for (int l=0; l<secondIndex; l++)

{

String[] insideSecond=secondMatrix[l].split(",");

int firstMatrixVal = Integer.parseInt(insideFirst[2]);

int secondMatrixVal = Integer.parseInt(insideSecond[2]);

String tempValue = 1+","+insideFirst[1]+","+insideSecond[1]+","+firstMatrixVal\*secondMatrixVal;

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

}

}

}

}

**SecondMapper.java**

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.fs.Path;

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

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

import org.apache.hadoop.io.Text;

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 SecondMapper extends Mapper<LongWritable, Text, Text, Text> {

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

// String strKey=key.toString();

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

String nKey=keyValues[1]+","+keyValues[2];

String nValue=keyValues[3]+"";

context.write(new Text(nKey), new Text(nValue));

}

}

**SecondReducer.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 SecondReducer

extends Reducer<Text, Text, Text, Text> {

String var="";

@Override

public void reduce(Text key, Iterable<Text> values,

Context context)

throws IOException, InterruptedException {

int sum=0;

for (Text val : values) {

int value = Integer.parseInt(val.toString());

sum =sum+value;

}

var = sum+"";

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

}

}

**MatrixMultiplication.java**

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.io.IntWritable;

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

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

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

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

public class MatrixMultiplication extends Configured implements Tool {

private static final String OUTPUT\_PATH = "Matrix\_Intermediateoutput10";

@Override

public int run(String[] args) throws Exception {

Configuration conf = getConf();

FileSystem fs = FileSystem.get(conf);

Job job = new Job(conf,"job 1");

job.setJarByClass(MatrixMultiplication.class);

//job.setJobName("Matrix Multiplication");

job.setMapperClass(FirstMapper.class);

job.setReducerClass(FirstReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

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

TextOutputFormat.setOutputPath(job, new Path(OUTPUT\_PATH));

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

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.waitForCompletion(true);

Job job1 = new Job(conf,"job 2");

job1.setJarByClass(MatrixMultiplication.class);

// job.setJobName("Matrix Multiplication");

job1.setMapperClass(SecondMapper.class);

job1.setReducerClass(SecondReducer.class);

job1.setInputFormatClass(TextInputFormat.class);

job1.setOutputFormatClass(TextOutputFormat.class);

TextInputFormat.addInputPath(job1, new Path(OUTPUT\_PATH));

TextOutputFormat.setOutputPath(job1, new Path(args[1]));

job1.setOutputKeyClass(Text.class);

job1.setOutputValueClass(Text.class);

return job1.waitForCompletion(true) ? 0 : 1;

}

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

ToolRunner.run(new Configuration(), new MatrixMultiplication(), args);

}

}

**Input.txt**

A,1,1,1

A,1,2,2

A,2,1,3

A,2,2,4

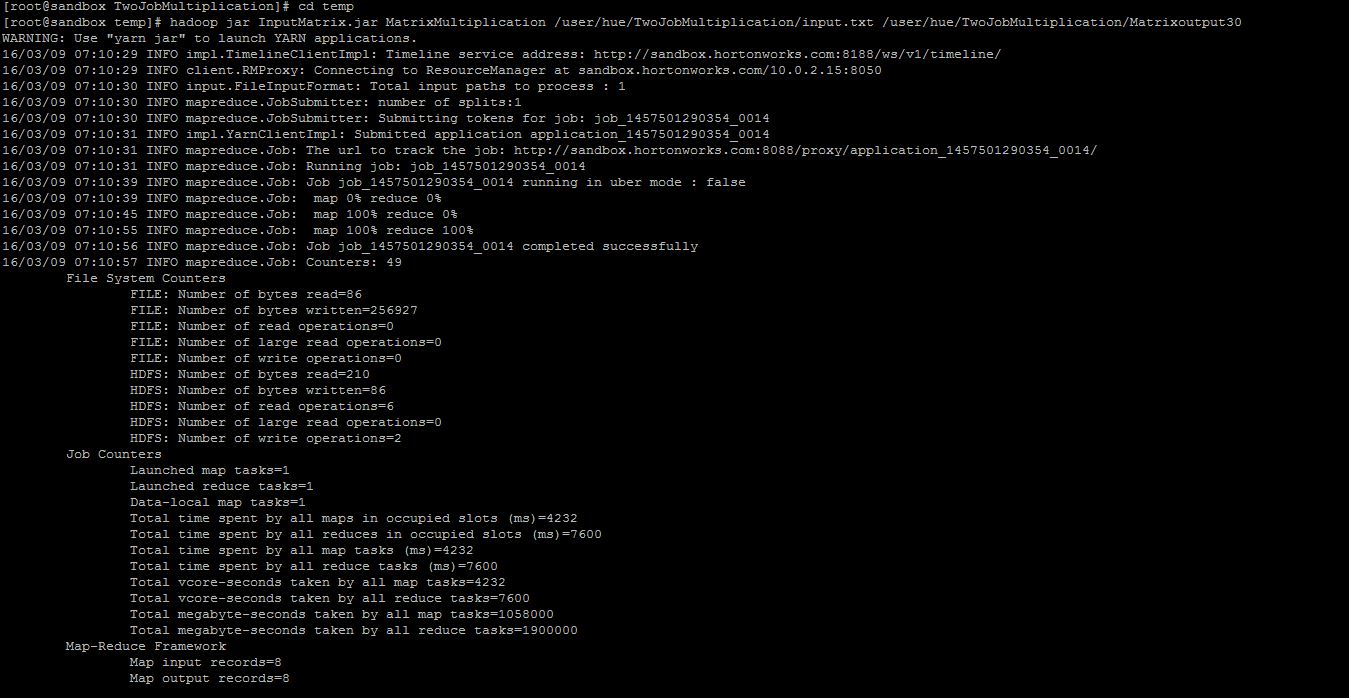
B,1,1,5

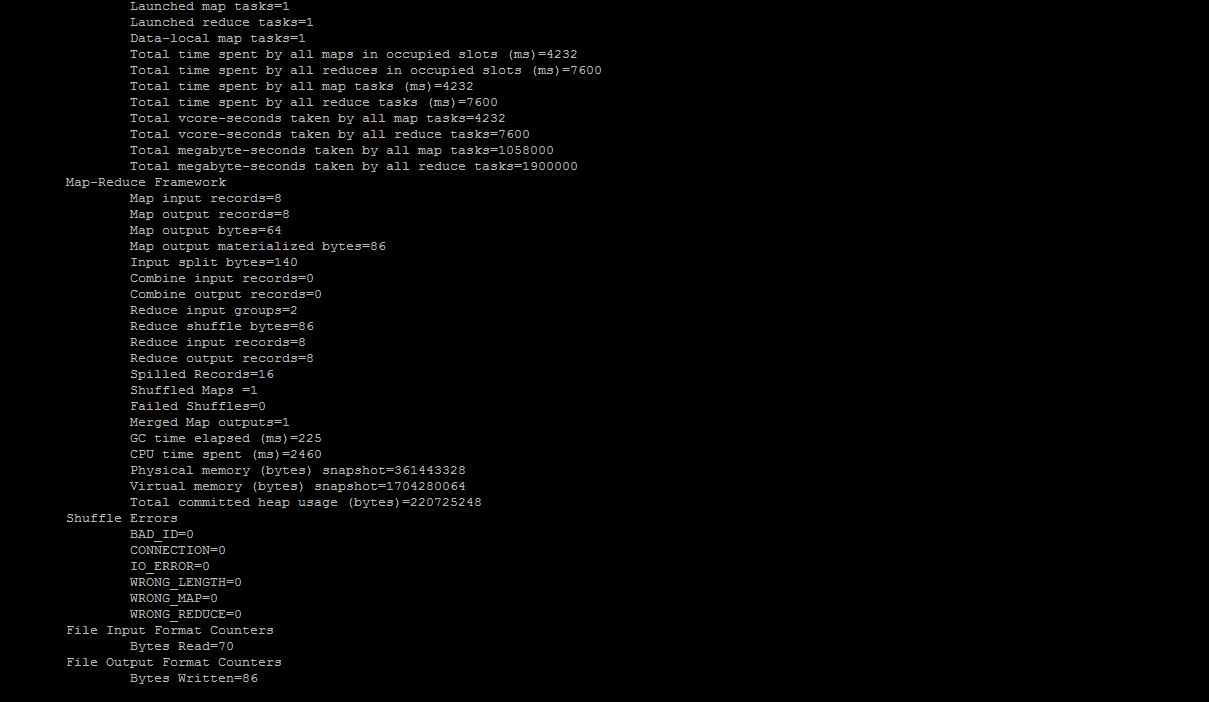
B,1,2,6

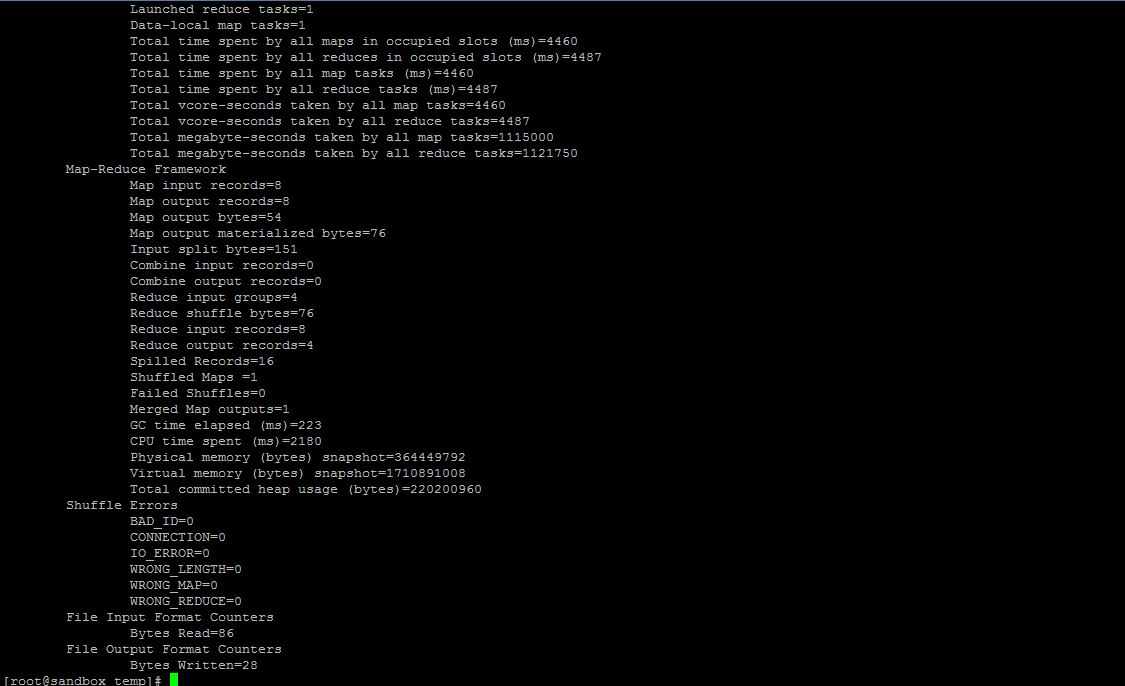
B,2,1,7

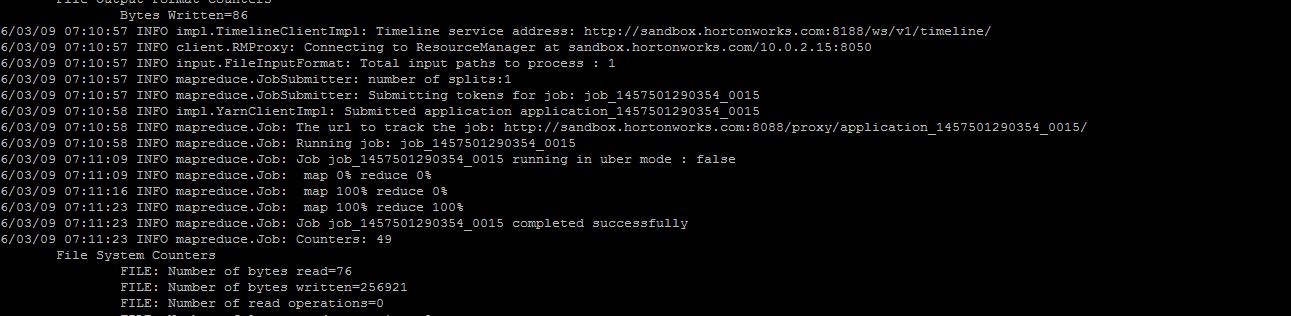
B,2,2,8

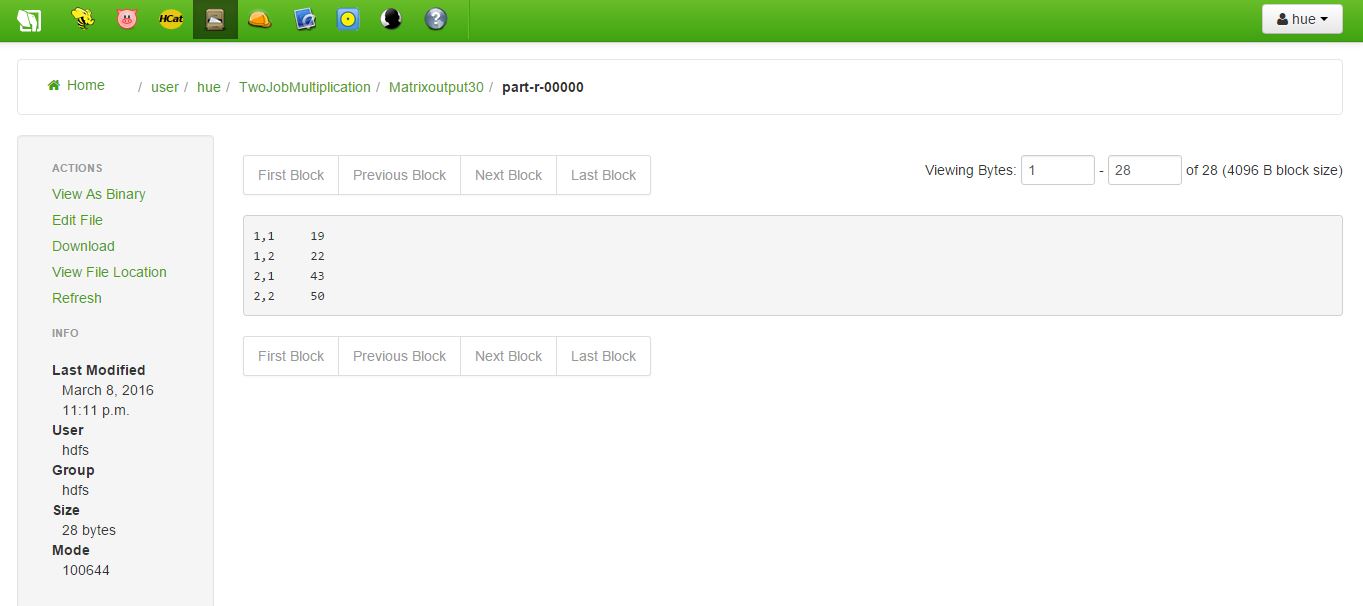
**Output**

****

****

****

****

****