package Matrixmulti;

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.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 Matrix{

public static class Map extends Mapper<LongWritable, Text, Text, Text> {

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();

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

Text outputKey = new Text();

Text outputValue = new Text();

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

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

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

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

context.write(outputKey, outputValue);

}

} else {

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

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

outputValue.set("B," + indicesAndValue[1] + "," + indicesAndValue[3]);

context.write(outputKey, outputValue);

}

}

}

}

public static class Reduce extends Reducer<Text, Text, Text, Text> {

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

String[] value;

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("A")) {

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 a\_ij;

float b\_jk;

for (int j = 0; j < n; j++) {

a\_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;

b\_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;

result += a\_ij \* b\_jk;

}

if (result != 0.0f) {

context.write(null, new Text(key.toString() + "," + Float.toString(result)));

}

}

}

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

Configuration conf = new Configuration();

// A is an m-by-n matrix; B is an n-by-p matrix.

conf.set("m", "3");

conf.set("n", "3");

conf.set("p", "3");

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

job.setJarByClass(Matrix.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(Map.class);

job.setReducerClass(Reduce.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);

}

}

matrixx

GNU nano 1.3.12 File: matip.txt

A,0,0,1.0

A,0,1,1.0

A,0,2,1.0

A,1,0,1.0

A,1,1,1.0

A,1,2,1.0

A,2,0,1.0

A,2,1,1.0

A,2,2,1.0

B,0,0,1.0

B,0,1,1.0

B,0,2,1.0

B,1,0,1.0

B,1,1,1.0

B,1,2,1.0

B,2,0,1.0

B,2,1,1.0

B,2,2,1.0