**Practical 6 : Implementing Matrix Multiplication with Hadoop Map Reduce.**

file1: matrixmultiply.java

package com.lendap.hadoop;

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 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("deprecation")

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.class);

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

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

job.waitForCompletion(true);

}

}

file2: map.java

package com.lendap.hadoop;

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);

}

}

}

}

file3: reduce.java

package com.lendap.hadoop;

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)));

}

}

}

**Input:**

M,0,0,1.0

M,0,1,2.0

M,0,2,3.0

M,1,0,4.0

M,1,1,5.0

M,1,2,6.0

M,2,0,7.0

M,2,1,8.0

M,2,2,9.0

N,0,0,9.0

N,0,1,8.0

N,1,0,7.0

N,1,1,6.0

N,2,0,5.0

N,2,1,4.0

**Output:**

0,0,38.0

0,1,32.0

1,0,101.0

1,1,86.0

2,0,164.0

2,1,140.0

**Result Matrix:**

38.0 32.0

101.0 86.0

164.0 140.0