Q) Patent Problem

package com.mr;

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

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.Mapper.Context;

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 Patent {

public static class Map extendsMapper<LongWritable, Text, Text, Text> {

//Defining a local variable K of type Text

Text k= new Text();

//Defining a local variable v of type Text

Text v= new Text();

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

//Converting the record (single line) to String and storing it in a String variable line

String line = value.toString();

//StringTokenizer is breaking the record (line) according to the delimiter whitespace

StringTokenizer tokenizer = new StringTokenizer(line," ");

//Iterating through all the tokens and forming the key value pair

while (tokenizer.hasMoreTokens()) {

String patent= tokenizer.nextToken();

k.set(patent);

String patent1= tokenizer.nextToken();

v.set(patent1);

//Sending to output collector which inturn passes the same to reducer

context.write(k,v);

}

}

}

//End of Mapper

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

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

//Defining a local variable sum of type int

int sum = 0;

for(Text x : values)

{

sum++;

}

//Dumping the output in context object

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

}

}

//end of Reducer

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

//reads the default configuration of cluster from the configuration xml files

Configuration conf = new Configuration();

//Initializing the job with the default configuration of the cluster

Job = new Job(conf, "patent");

//Assigning the driver class name

job.setJarByClass(Patent.class);

//Defining the mapper class name

job.setMapperClass(Map.class);

//Defining the reducer class name

job.setReducerClass(Reduce.class);

//Explicitly setting the out key/value type from the mapper if it is not same as that of reducer

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

//Defining the output key class for the final output i.e. from reducer

job.setOutputKeyClass(Text.class);

//Defining the output value class for the final output i.e. from reducer

job.setOutputValueClass(IntWritable.class);

//Defining the output key class for the final output i.e. from reducer

job.setOutputKeyClass(Text.class);

//Defining the output value class for the final output i.e. from reducer

job.setOutputValueClass(Text.class);

//Defining input Format class which is responsible to parse the dataset into a key value pair

job.setInputFormatClass(TextInputFormat.class);

//Defining output Format class which is responsible to parse the final key-value output from MR framework to a text file into the hard disk

job.setOutputFormatClass(TextOutputFormat.class);

//setting the second argument as a path in a path variable

Path outputPath = new Path(args[1]);

//Configuring the input/output path from the filesystem into the job

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

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

//deleting the output path automatically from hdfs so that we don't have delete it explicitly

outputPath.getFileSystem(conf).delete(outputPath);

//exiting the job only if the flag value becomes false

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

}

}