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.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 WordSizeWordCount {

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

//Defining a local variable count of type IntWritable

private static IntWritable count ;

//Defining a local variable word of type Text

private Text word = new Text();

**//Mapper**

/\*\*

\* @method map

\* <p>This method takes the input as text data type and splits the input into words.

\* Now the length of each word in the input is determined and key value pair is made.

\* This key value pair is passed to reducer.

\* @method\_arguments key, value, output, reporter

\* @return void

\*/

/\*

\* (non-Javadoc)

\* @see org.apache.hadoop.mapred.Mapper#map(java.lang.Object, java.lang.Object, org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)

\*/

@Override

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) into words

StringTokenizer tokenizer = new StringTokenizer(line);

//iterating through all the words available in that line and forming the key value pair

while (tokenizer.hasMoreTokens()) {

String thisH = tokenizer.nextToken();

//finding the length of each token(word)

count= new IntWritable(thisH.length());

word.set(thisH);

//Sending to output collector which in turn passes the same to reducer

//So in this case the output from mapper will be the length of a word and that word

context.write(count,word);

}

}

}

**//Reducer**

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

/\*\*

\* @method reduce

\* <p>This method takes the input as key and list of values pair from mapper, it does aggregation

\* based on keys and produces the final output.

\* @method\_arguments key, values, output, reporter

\* @return void

\*/

/\*

\* (non-Javadoc)

\* @see org.apache.hadoop.mapred.Reducer#reduce(java.lang.Object, java.util.Iterator, org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)

\*/

@Override

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

throws IOException, InterruptedException {

//Defining a local variable sum of type int

int sum = 0;

/\*

\* Iterates through all the values available with a key and add them together and give the final

\* result as the key and sum of its values.

\*/

for(Text x : values)

{

sum++;

}

//Dumping the output

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

}

}

**//Driver**

/\*\*

\* @method main

\* <p>This method is used for setting all the configuration properties.

\* It acts as a driver for map reduce code.

\* @return void

\* @method\_arguments args

\* @throws Exception

\*/

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, "Wordsize");

//Assigning the driver class name

job.setJarByClass(WordSizeWordCount.class);

//Defining the mapper class name

job.setMapperClass(Map.class);

//Defining the reducer class name

job.setReducerClass(Reduce.class);

//Defining the output key class for the mapper

job.setMapOutputKeyClass(IntWritable.class);

//Defining the output value class for the mapper

job.setMapOutputValueClass(Text.class);

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

job.setOutputKeyClass(IntWritable.class);

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

job.setOutputValueClass(IntWritable.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);

}

}