**Module 3, Exercise 1:** Implement MapReduce Jobs.

Note:

* The programs can be executed in Eclipse, too.
* The highlighted text needs to be filled by a valid path.
* Delete the output files if they already exist.

**Note: Please find the java files required for this exercise in this path: /home/mapr/Desktop/mapr\_training/java\_files**

**Solution 1:**

1. Save the below code as WordCount.java. In this case, its already saved.

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.mapred.\*;

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

public class WordCount {

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

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

output.collect(word, one);

}

}

}

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

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

int sum = 0;

while (values.hasNext()) {

sum += values.next().get();

}

output.collect(key, new IntWritable(sum));

}

}

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

JobConf conf = new JobConf(WordCount.class);

conf.setJobName("wordcount");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Map.class);

conf.setCombinerClass(Reduce.class);

conf.setReducerClass(Reduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, "maprfs:///user/mapr/input");

FileOutputFormat.setOutputPath(conf, new Path("maprfs:///user/mapr/output"));

JobClient.runJob(conf);

}

}

Compilation steps:

* Navigate to the folder where WordCount.java file exists. Compile and create jar

$**javac –cp $(hadoop classpath) WordCount.java**

$**jar cf wc.jar WordCount\*.class**

* Run the jar

$**hadoop jar wc.jar WordCount /user/mapr/input /user/mapr/output**

**Solution 2:**

1. Save the below code as LineDivider.java. In this case, its already saved.

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.mapred.\*;

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

public class LengthDivider {

public static class LengthDividerCountMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

public final static IntWritable countOne = new IntWritable(1);

private final Text fiveCharsOrMore = new Text("greaterOrEqualsToFiveChars");

private final Text lessThanFiveChars = new Text("lessThanFiveChars");

@Override

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

StringTokenizer tokenizer = new StringTokenizer(value.toString());

while (tokenizer.hasMoreTokens()) {

if (tokenizer.nextToken().length() >= 5){

output.collect(fiveCharsOrMore, countOne);

} else {

output.collect(lessThanFiveChars, countOne);

}

}

}

}

public static class LengthDividerCountReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

@Override

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

int sum = 0;

while (values.hasNext()) {

sum += values.next().get();

}

output.collect(key, new IntWritable(sum));

}

}

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

JobConf conf = new JobConf(LengthDivider.class);

conf.setJobName("length-divider");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(LengthDividerCountMapper.class);

conf.setCombinerClass(LengthDividerCountReducer.class);

conf.setReducerClass(LengthDividerCountReducer.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf,new Path("maprfs:///user/mapr/input"));

FileOutputFormat.setOutputPath(conf, new Path("maprfs:///user/mapr/out\_line"));

JobClient.runJob(conf);

}

}

1. Compilation steps:

* Navigate to the folder where LineDivider.java file exists. Compile and create jar

$**javac –cp $(hadoop classpath)** LineDivider**.java**

$**jar cf ld.jar LineDivider\*.class**

* Run the jar

$**hadoop jar ld.jar LineDivider /user/mapr/input user/mapr/out\_line**

**Solution 3:**

1. Save the below code as LineIndexer.java. In this case, its already saved.

import java.io.IOException;

import java.util.Iterator;

import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.FileSplit;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class LineIndexer {

public static class LineIndexMapper extends MapReduceBase implements Mapper<LongWritable,Text,OutputCollector,Reporter> {

private final static Text word = new Text();

private final static Text location = new Text();

public void map(LongWritable key, Text val,OutputCollector output, Reporter reporter) throws IOException {

FileSplit fileSplit = (FileSplit)reporter.getInputSplit();

String fileName = fileSplit.getPath().getName();

location.set(fileName);

String line = val.toString();

StringTokenizer itr = new StringTokenizer(line.toLowerCase());

while (itr.hasMoreTokens()) {

word.set(itr.nextToken());

output.collect(word, location);

}

}

}

public static class LineIndexReducer extends MapReduceBase implements Reducer<Text,Iterator,OutputCollector,Reporter> {

public void reduce(Text key, Iterator values,OutputCollector output, Reporter reporter) throws IOException {

boolean first = true;

StringBuilder toReturn = new StringBuilder();

while (values.hasNext()){

if (!first)

toReturn.append(", ");

first=false;

toReturn.append(values.next().toString());

}

output.collect(key, new Text(toReturn.toString()));

}

}

/\*\*\* The actual main() method for our program; this is the

\* "driver" for the MapReduce job.

\*/ public static void main(String[] args) {

JobClient client = new JobClient();

JobConf conf = new JobConf(LineIndexer.class);

conf.setJobName("LineIndexer");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(conf, new Path("/user/mapr/input/"));

FileOutputFormat.setOutputPath(conf, new Path("/maprfs:///user/mapr/out\_index"));

conf.setMapperClass(LineIndexMapper.class);

conf.setReducerClass(LineIndexReducer.class);

client.setConf(conf);

try {

JobClient.runJob(conf);

} catch (Exception e) {

e.printStackTrace();

}

}

}

1. Compilation steps:

* Navigate to the folder where LineIndexer.java file exists. Compile and create jar

$**javac –cp $(hadoop classpath)** LineIndexer**.java**

$**jar cf li.jar LineIndexer\*.class**

* Run the jar

$**hadoop jar li.jar LineIndexer /user/mapr/input user/mapr/out\_indexer**