JAVA PROGRAM

package org.myorg;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader; //Import classes for handling file I/O

import java.net.URI;

import java.util.HashSet;

import java.util.Set;

import java.io.IOException;

import java.util.regex.Pattern;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.util.StringUtils; //working with strings in Hadoop

import org.apache.log4j.Logger;

public class WordCount extends Configured implements Tool {

private static final Logger LOG = Logger.getLogger(WordCount.class);

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

int res = ToolRunner.run(new WordCount(), args);

System.exit(res);

}public int run(String[] args) throws Exception {

Job job = Job.getInstance(getConf(), "wordcount");

//Skip pattern configuration

for (int i = 0; i < args.length; i += 1) {

if ("-skip".equals(args[i])) {

job.getConfiguration().setBoolean("wordcount.skip.patterns", true);

i += 1;

job.addCacheFile(new Path(args[i]).toUri());

// this demonstrates logging

LOG.info("Added file to the distributed cache: " + args[i]);

}

}job.setJarByClass(this.getClass());

// Use TextInputFormat, the default unless job.setInputFormatClass is used

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

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

job.setMapperClass(Map.class);

job.setCombinerClass(Reduce.class);

job.setReducerClass(Reduce.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

return job.waitForCompletion(true) ? 0 : 1;

}

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

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

private Text word = new Text();

private boolean caseSensitive = false;

private long numRecords = 0;

private String input;

private Set<String> patternsToSkip = new HashSet<String>(); //stop words to be removed from the final result

private static final Pattern WORD\_BOUNDARY = Pattern.compile("\\s\*\\b\\s\*");

protected void setup(Mapper.Context context)

throws IOException,

InterruptedException {

if (context.getInputSplit() instanceof FileSplit) {

this.input = ((FileSplit) context.getInputSplit()).getPath().toString();

} else {

this.input = context.getInputSplit().toString();

}

Configuration config = context.getConfiguration();

this.caseSensitive = config.getBoolean("wordcount.case.sensitive", false);

//parseSkipFile method

if (config.getBoolean("wordcount.skip.patterns", false)) {

URI[] localPaths = context.getCacheFiles();

parseSkipFile(localPaths[0]);

}

}

//Getting file from the HDFS and to read until EOL

private void parseSkipFile(URI patternsURI) {

LOG.info("Added file to the distributed cache: " + patternsURI);

try {

BufferedReader fis = new BufferedReader(new FileReader(new File(patternsURI.getPath()).getName()));

String pattern;

while ((pattern = fis.readLine()) != null) {

patternsToSkip.add(pattern);

}

} catch (IOException ioe) {

System.err.println("Caught exception while parsing the cached file '"

+ patternsURI + "' : " + StringUtils.stringifyException(ioe));

}

}public void map(LongWritable offset, Text lineText, Context context)

throws IOException, InterruptedException {

String line = lineText.toString();

if (!caseSensitive) {

line = line.toLowerCase();

}

Text currentWord = new Text();

for (String word : WORD\_BOUNDARY.split(line)) {

if (word.isEmpty() || patternsToSkip.contains(word)) {

continue;

}

currentWord = new Text(word);

context.write(currentWord,one);

}

}

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

@Override

public void reduce(Text word, Iterable<IntWritable> counts, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable count : counts) {

sum += count.get();

}

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

}

}

}

COMMANDS

hadoop version

2 javac -version

3 export HADOOP\_CLASSPATH=$(hadoop classpath)

4 echo $HADOOP\_CLASSPATH

5 hadoop fs -mkdir /stopword

6 start-all.sh

7

8 hadoop fs -mkdir hdfs://localhost:9000/stopword

9 hadoop fs -mkdir hdfs://localhost:9000/stopword/input

10 hadoop fs -put '/home/hadoop/input\_data/input.txt' /stopword/input

11 ls

12 javac -classpath ${HADOOP\_CLASSPATH} -d '/home/hadoop/tutorial\_classes' '/home/hadoop/WordCount.java'

13 jar -cvf firstTutorial.jar -C tutorial\_classes/ .

14 hadoop jar '/home/hadoop/Desktop/wscw/firstTutorial.jar' org.myorg.WordCount /stopword/input /stopword/output -skip /stopword/input.txt

15 hadoop dfs -cat /stopword/output/\*

16 hadoop fs -cat hdfs://localhost:9000/stopword/output/\*

17 hadoop fs -cat hdfs://localhost:9000/stopword/output

18 hadoop dfs -cat /stopword/output/\*

19 hadoop hdfs -cat /stopword/output/\*

20 hadoop dfs -ls /stopword/output/\*

21 hadoop dfs -cat /stopword/output/\_SUCCESS

22 hadoop dfs -cat /stopword/output/part-r-00000







