CODES FOR EACH STEP:

STEP1:  Run Word Count 1 example  on your local psudo-distributed system with supplied text files

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.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.output.FileOutputFormat;

public class wc1 {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

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

private Text word = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

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

while (itr.hasMoreTokens()) {

word.set(itr.nextToken());

context.write(word, one);

}

}

}

public static class IntSumReducer

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

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

Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(wc1.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}

}

Step2: Run Word Count 2 example on your local psudo-distributed system with supplied text files

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.net.URI;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

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.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.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.Counter;

import org.apache.hadoop.util.GenericOptionsParser;

import org.apache.hadoop.util.StringUtils;

public class wc2 {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

static enum CountersEnum { INPUT\_WORDS }

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

private Text word = new Text();

private boolean caseSensitive;

private Set<String> patternsToSkip = new HashSet<String>();

private Configuration conf;

private BufferedReader fis;

@Override

public void setup(Context context) throws IOException,

InterruptedException {

conf = context.getConfiguration();

caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);

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

URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();

for (URI patternsURI : patternsURIs) {

Path patternsPath = new Path(patternsURI.getPath());

String patternsFileName = patternsPath.getName().toString();

parseSkipFile(patternsFileName);

}

}

}

private void parseSkipFile(String fileName) {

try {

fis = new BufferedReader(new FileReader(fileName));

String pattern = null;

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

patternsToSkip.add(pattern);

}

} catch (IOException ioe) {

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

+ StringUtils.stringifyException(ioe));

}

}

@Override

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

String line = (caseSensitive) ?

value.toString() : value.toString().toLowerCase();

for (String pattern : patternsToSkip) {

line = line.replaceAll(pattern, "");

}

StringTokenizer itr = new StringTokenizer(line);

while (itr.hasMoreTokens()) {

word.set(itr.nextToken());

context.write(word, one);

Counter counter = context.getCounter(CountersEnum.class.getName(),

CountersEnum.INPUT\_WORDS.toString());

counter.increment(1);

}

}

}

public static class IntSumReducer

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

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

Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

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

Configuration conf = new Configuration();

GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);

String[] remainingArgs = optionParser.getRemainingArgs();

if ((remainingArgs.length != 2) && (remainingArgs.length != 4)) {

System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");

System.exit(2);

}

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(wc2.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

List<String> otherArgs = new ArrayList<String>();

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

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

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

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

} else {

otherArgs.add(remainingArgs[i]);

}

}

FileInputFormat.addInputPath(job, new Path(otherArgs.get(0)));

FileOutputFormat.setOutputPath(job, new Path(otherArgs.get(1)));

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

}

}

Step 3: Modify Wordcount 1 to look for only words that occur more than 4 times

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.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.output.FileOutputFormat;

public class wc1 {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

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

private Text word = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

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

while (itr.hasMoreTokens()) {

word.set(itr.nextToken());

context.write(word, one);

}

}

}

public static class IntSumReducer

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

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

Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

if(sum>4)

{

result.set(sum);

context.write(key, result);

}

else

{

System.out.println(“\n “ ); //I have kept it blank as I don’t want any output to be sent in this case.

}

}

}

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(wc1.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}

}

Step 4: Modify Wordcount 2 to  modify and use the **-skip** command line parameter from the example and add to the**pattern.txt** file to skip: all punctuation and English prepositions

It is the same code as step 2.

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.net.URI;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

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.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.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.Counter;

import org.apache.hadoop.util.GenericOptionsParser;

import org.apache.hadoop.util.StringUtils;

public class wc2 {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

static enum CountersEnum { INPUT\_WORDS }

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

private Text word = new Text();

private boolean caseSensitive;

private Set<String> patternsToSkip = new HashSet<String>();

private Configuration conf;

private BufferedReader fis;

@Override

public void setup(Context context) throws IOException,

InterruptedException {

conf = context.getConfiguration();

caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);

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

URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();

for (URI patternsURI : patternsURIs) {

Path patternsPath = new Path(patternsURI.getPath());

String patternsFileName = patternsPath.getName().toString();

parseSkipFile(patternsFileName);

}

}

}

private void parseSkipFile(String fileName) {

try {

fis = new BufferedReader(new FileReader(fileName));

String pattern = null;

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

patternsToSkip.add(pattern);

}

} catch (IOException ioe) {

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

+ StringUtils.stringifyException(ioe));

}

}

@Override

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

String line = (caseSensitive) ?

value.toString() : value.toString().toLowerCase();

for (String pattern : patternsToSkip) {

line = line.replaceAll(pattern, "");

}

StringTokenizer itr = new StringTokenizer(line);

while (itr.hasMoreTokens()) {

word.set(itr.nextToken());

context.write(word, one);

Counter counter = context.getCounter(CountersEnum.class.getName(),

CountersEnum.INPUT\_WORDS.toString());

counter.increment(1);

}

}

}

public static class IntSumReducer

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

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

Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

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

Configuration conf = new Configuration();

GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);

String[] remainingArgs = optionParser.getRemainingArgs();

if ((remainingArgs.length != 2) && (remainingArgs.length != 4)) {

System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");

System.exit(2);

}

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(wc2.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

List<String> otherArgs = new ArrayList<String>();

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

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

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

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

} else {

otherArgs.add(remainingArgs[i]);

}

}

FileInputFormat.addInputPath(job, new Path(otherArgs.get(0)));

FileOutputFormat.setOutputPath(job, new Path(otherArgs.get(1)));

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

}

}

Now what I did to get the output(top 10 words) was that I copied the output file of each output step( part-r-00000) into my local folder using the following command:



I then opened the file in notepad and copied the contents into an excel document. I then sorted the excel document based on the number of occurances of each word.