**HU Extension School E-185 Big Data Analytics Assignment 06**

**Handed out: 03/15/2013 Due by 5:30PM on Saturday, 03/23/2013**

**Problem 1) Attached file “Using String Tokenizer in Java.docx” contains a small tutorial article on properties of jav.util.StringTokenizer class. You should also consult Java docs for that class. You can find those at:** [**http://docs.oracle.com/javase/6/docs/api/java/util/StringTokenizer.html**](http://docs.oracle.com/javase/6/docs/api/java/util/StringTokenizer.html)

**Both from the article and from the class description you realize that the String Tokenizer used in our class example WordCount.java, file is attached, could do more. If not told otherwise, it breaks strings into tokens on empty spaces. That left a lot of punctuations, parenthesis and the like on the result of our wordcount.jar MapReduce program. Modify WordCount.java class so that when you run wordcount.jar on all-bible text, you get the list of words and without punctuations and other non-word characters.**

**As an advice, please extract the map() method into a standalone Java class which you can run on the command prompt. Instead of writing to the context object, use System.out.println to write to the console. When you are sure that you the map() method does what you expect it to do, only then place it back into the class WordCount.java.**

**Similarly, modify the reduce() method, so that it emits only those words which appear less than 7000 times. Let reduce() also get rid of verse counts in the form 01:004:010. Run your new version of wordcount.jar on your favorite VM. Use the attached all-bible.txt file for your test.**

Initially, we copied the WordCount.java file and the all-bible.txt file to the Hadoop VM, and made sure that we can compile the example as is:

**[cloudera@centos-e185 Assign06]$ javac -classpath `hadoop classpath` -Xlint:deprecation -d class WordCount.java**

WordCount.java:58: warning: [deprecation] Job(org.apache.hadoop.conf.Configuration,java.lang.String) in org.apache.hadoop.mapreduce.Job has been deprecated

Job job = new Job(conf, "word count");

^

1 warning

**[cloudera@centos-e185 Assign06]$ ls -l class/org/apache/hadoop/examples/**

total 12

-rw-rw-r--. 1 cloudera cloudera 1911 Mar 20 11:15 WordCount.class

-rw-rw-r--. 1 cloudera cloudera 1789 Mar 20 11:15 WordCount$IntSumReducer.class

-rw-rw-r--. 1 cloudera cloudera 1790 Mar 20 11:15 WordCount$TokenizerMapper.class

We create the jar file:

**[cloudera@centos-e185 class]$ jar -cvf MyWordCount.jar org/\***

added manifest

adding: org/apache/(in = 0) (out= 0)(stored 0%)

adding: org/apache/hadoop/(in = 0) (out= 0)(stored 0%)

adding: org/apache/hadoop/examples/(in = 0) (out= 0)(stored 0%)

adding: org/apache/hadoop/examples/WordCount$IntSumReducer.class(in = 1789) (out= 747)(deflated 58%)

adding: org/apache/hadoop/examples/WordCount$TokenizerMapper.class(in = 1790) (out= 765)(deflated 57%)

adding: org/apache/hadoop/examples/WordCount.class(in = 1911) (out= 996)(deflated 47%)

In order to test it against the all-bible.txt file, we first we have to clean up our HDFS home directory and load the file up to it.

**[cloudera@centos-e185 Assign06]$ hadoop fs -ls**

**[cloudera@centos-e185 Assign06]$ hadoop fs -mkdir input**

**[cloudera@centos-e185 Assign06]$ hadoop fs -ls**

Found 1 items

drwxr-xr-x - cloudera supergroup 0 2013-03-20 11:27 input

**[cloudera@centos-e185 Assign06]$ hadoop fs -copyFromLocal all-bible.txt input/**

**[cloudera@centos-e185 Assign06]$ hadoop fs -ls input/**

Found 1 items

-rw-r--r-- 1 cloudera supergroup 5258688 2013-03-20 11:30 input/all-bible.txt

**[cloudera@centos-e185 Assign06]$ hadoop fs -rm -R output**

rm: `output': No such file or directory

And we execute the jar against the our input files and check the response is in-line with what we expected:

**[cloudera@centos-e185 Assign06]$ hadoop jar class/MyWordCount.jar** org.apache.hadoop.examples.WordCount input output

13/03/20 11:36:41 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is inited.

13/03/20 11:36:41 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is started.

13/03/20 11:36:41 INFO input.FileInputFormat: Total input paths to process : 1

13/03/20 11:36:41 INFO mapreduce.JobSubmitter: number of splits:1

(...)

File Input Format Counters

Bytes Read=5258688

File Output Format Counters

Bytes Written=717079

**[cloudera@centos-e185 Assign06]$ hadoop fs -ls output**

Found 2 items

-rw-r--r-- 1 cloudera supergroup 0 2013-03-20 11:37 output/\_SUCCESS

-rw-r--r-- 1 cloudera supergroup 717079 2013-03-20 11:37 output/part-r-00000

**[cloudera@centos-e185 Assign06]$ hadoop fs -cat output/part-r-00000 | head -20**

"Defects," 2

"Information 2

"Plain 4

"Project 10

"Right 2

#7999] 2

$5,000) 2

'AS-IS' 2

("the 2

($1 2

(801) 2

(According 1

(After 1

(Also 1

(And 6

(As 4

(Beforetime 1

(But 2

(For 34

(Howbeit 1

cat: Unable to write to output stream.

**[cloudera@centos-e185 Assign06]$ hadoop fs -cat output/part-r-00000 | tail -20**

youth. 15

youth: 7

youth; 8

youth? 2

youthful 1

youths 1

youths, 1

zeal 13

zeal, 3

zealous 8

zealously 2

{ 12

{background:#faebd7; 68

{border: 2

{color:#A82C28} 66

{float: 4

{font-size: 2

{font-size:14pt} 66

{font-size:16pt} 66

} 8

Given the output, we start making the changes to the Mapper and Reducer functions. In order to make things separate and repeatable, we create a new directory and a new copy of the .java file to change and edit.

[cloudera@centos-e185 Assign06]$ mkdir P1

[cloudera@centos-e185 Assign06]$ mv WordCount.java P1/WordCount.java

[cloudera@centos-e185 Assign06]$ cd P1

[cloudera@centos-e185 P1]$ vi WordCountP1.java

We made the following changes to the WordCountP1.java file:

* The package name was changed to "e185.assign06" and the class name was changed to WordCountP1:

1 package e185.assign06;

2

3 import java.io.IOException;

4 import java.util.StringTokenizer;

5

(...)

17 public class WordCountP1 {

* We added filtering to the Mapper tokenizer function to filter all of non-word and non-digit characters, with exception to the ":" character, which is necessary to a later expression that will be filtered on the Reducer:

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

26 ) throws IOException, InterruptedException {

27 StringTokenizer itr = new StringTokenizer(value.toString(), " \t\n\r\f-.,;?'@#$!%^&\*~`<>|/\"()[]{}\\");

28 while (itr.hasMoreTokens()) {

29 word.set(itr.nextToken());

30 context.write(word, one);

31 }

* Finally, in the mapper function we filtered out words with verse counts such as 01:004:010, and also only printed words that appear less than 7000 times.

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

40 Context context

41 ) throws IOException, InterruptedException {

42 int sum = 0;

43 String strKey = key.toString();

44

45 // P1: Skip words with verse counts such as 01:004:010

46 if (strKey.matches("^\\d{2}:\\d{3}:\\d{3}$")) {

47 return;

48 }

49

50 for (IntWritable val : values) {

51 sum += val.get();

52 }

53

54 // P1: Only print words that appear less then 7000 times

55 if (sum < 7000) {

56 result.set(sum);

57 context.write(key, result);

58 }

We finally recompile the new and modified Java class and jar file to run it again the input and have the final output.

**[cloudera@centos-e185 P1]$ javac -classpath `hadoop classpath` -Xlint:deprecation -d . WordCountP1.java**

WordCountP1.java:69: warning: [deprecation] Job(org.apache.hadoop.conf.Configuration,java.lang.String) in org.apache.hadoop.mapreduce.Job has been deprecated

Job job = new Job(conf, "word count");

^

1 warning

**[cloudera@centos-e185 P1]$ jar -cvf WordCountP1.jar e185/\***

added manifest

adding: e185/assign06/(in = 0) (out= 0)(stored 0%)

adding: e185/assign06/WordCountP1.class(in = 1880) (out= 1002)(deflated 46%)

adding: e185/assign06/WordCountP1$IntSumReducer.class(in = 1955) (out= 859)(deflated 56%)

adding: e185/assign06/WordCountP1$TokenizerMapper.class(in = 1853) (out= 817)(deflated 55%)

**[cloudera@centos-e185 P1]$ hadoop fs -rm -r output**

Deleted output

**[cloudera@centos-e185 P1]$ hadoop jar WordCountP1.jar e185.assign06.WordCountP1 input output**

13/03/20 15:33:39 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is inited.

13/03/20 15:33:40 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is started.

13/03/20 15:33:40 INFO input.FileInputFormat: Total input paths to process : 1

13/03/20 15:33:40 INFO mapreduce.JobSubmitter: number of splits:1

(...)

File Input Format Counters

Bytes Read=5258688

File Output Format Counters

Bytes Written=173985

**[cloudera@centos-e185 Assign06]$ hadoop fs -cat output/part-r-00000 | wc -l**

16807

**[cloudera@centos-e185 Assign06]$ hadoop fs -cat output/part-r-00000 | head -20**

0 2

000 2

01 4

02 4

03 4

04 4

05 4

06 4

07 4

08 4

09 4

0em 2

1 120

10 4

10px 2

11 4

12 6

13 4

14 4

15 6

cat: Unable to write to output stream.

**[cloudera@centos-e185 Assign06]$ hadoop fs -cat output/part-r-00000 | tail -20**

you: 154

young 296

young: 2

younger 30

younger: 1

youngest 16

youngest: 2

your 1780

yours 4

yours: 1

yourselves 182

yourselves: 9

youth 63

youth: 7

youthful 1

youths 2

zeal 16

zealous 8

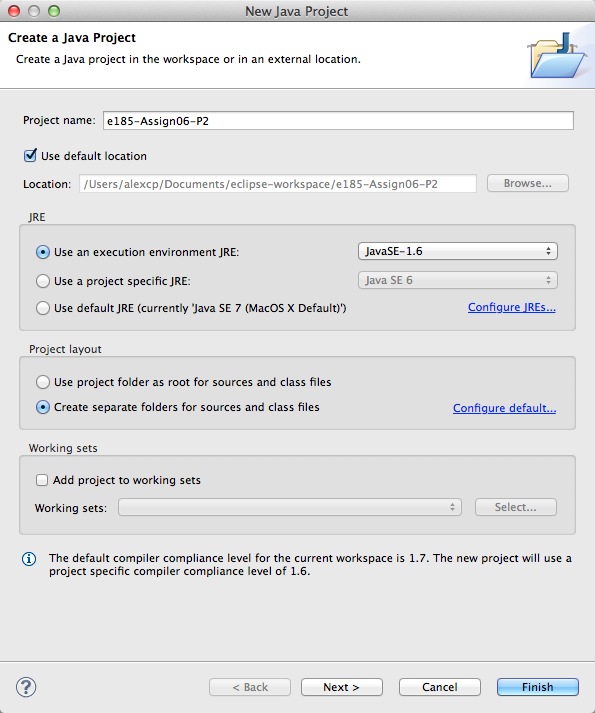
zealously 2

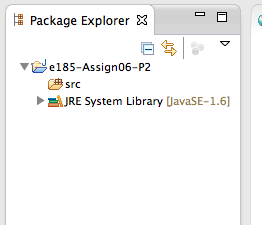
zip 2

As you can see, there are words still with the ":" character as it was explicitly not filtered out on the Mapper function. This is an intended behaviour as we wanted to filter out the "verse count" words on the Reducer function. The Java file and the output generated are available in the P1 directory included with this document.

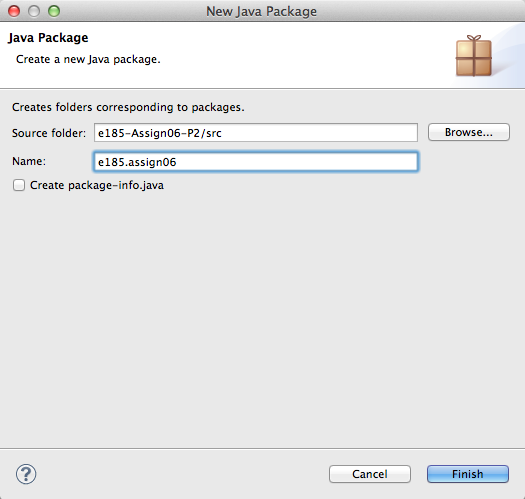
**Problem 2) Demonstrate that your modified WordCount.java will produce the same result whether you compile it and jar it on your PC (MAC) or on your Linux VM.**

Give we did the process for Problem 1 on the Linux VM, we start by starting up Eclipse and creating a project for the Java application, taking care to select the Java 1.6 version as the execution environment.

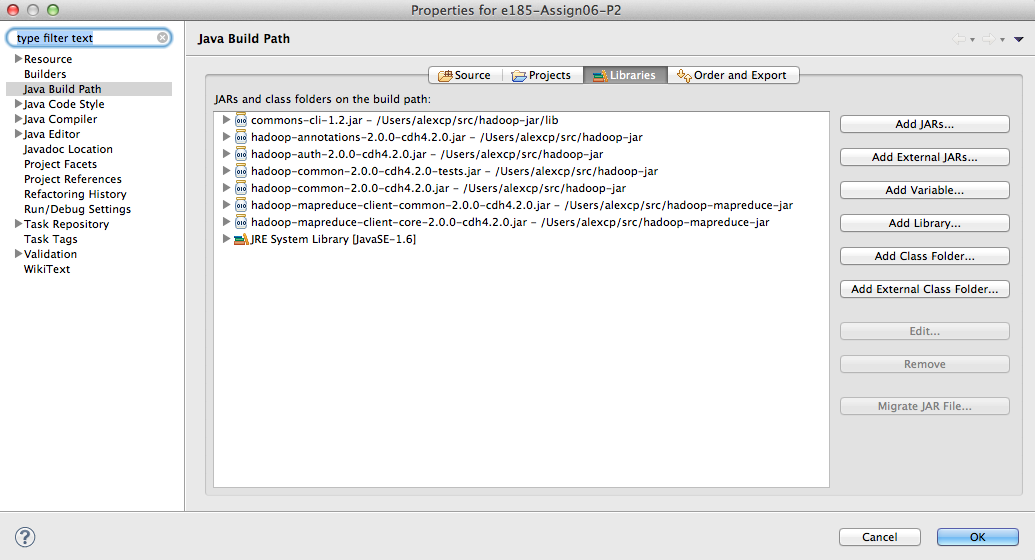




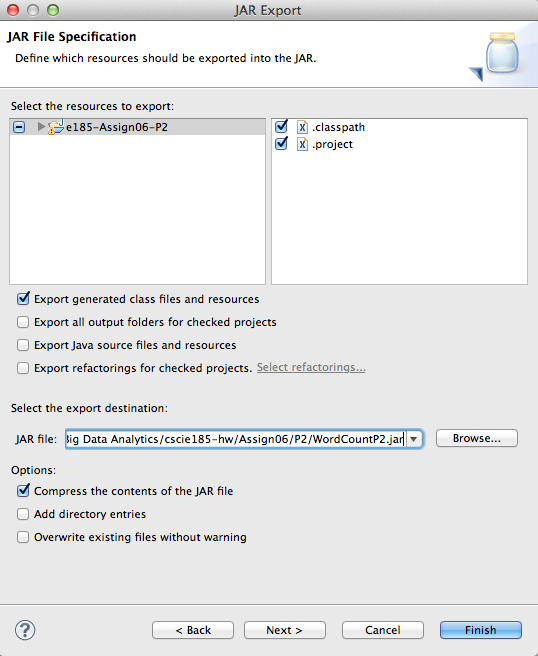
And we continue to create the package with the same name as the one we created before on the Problem 1.



And we add the WordCountP1.java file to the package, and suddenly we have the 48 errors as expected, for not having the necessary Hadoop jars included to the project. We take care of that next by downloading the contents of /usr/lib/hadoop and /usr/lib/hadoop-mapreduce to our local machine and adding the packages to the Eclipse project.



Finally we export the .jar file from the project as WordCountP2.jar.



We upload our file to the Linux VM to run it.

**aperture-2:P2 alexcp$ scp WordCountP2.jar cloudera@192.168.2.138:**

cloudera@192.168.2.138's password:

WordCountP2.jar 100% 4551 4.4KB/s 00:00

And we execute it from the Linux VM, getting the exact same result as we had before.

**[cloudera@centos-e185 P2]$ hadoop fs -rm -r output**

Deleted output

**[cloudera@centos-e185 P2]$ hadoop jar WordCountP2.jar e185.assign06.WordCountP1 input output**

13/03/20 18:24:49 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is inited.

13/03/20 18:24:49 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is started.

13/03/20 18:24:49 INFO input.FileInputFormat: Total input paths to process : 1

13/03/20 18:24:50 INFO mapreduce.JobSubmitter: number of splits:1

(...)

File Input Format Counters

Bytes Read=5258688

File Output Format Counters

Bytes Written=173985

**[cloudera@centos-e185 P2]$ hadoop fs -cat output/part-r-00000 | wc -l**

16807

**[cloudera@centos-e185 P2]$ hadoop fs -cat output/part-r-00000 | head -20**

0 2

000 2

01 4

02 4

03 4

04 4

05 4

06 4

07 4

08 4

09 4

0em 2

1 120

10 4

10px 2

11 4

12 6

13 4

14 4

15 6

cat: Unable to write to output stream.

**[cloudera@centos-e185 P2]$ hadoop fs -cat output/part-r-00000 | tail -20**

you: 154

young 296

young: 2

younger 30

younger: 1

youngest 16

youngest: 2

your 1780

yours 4

yours: 1

yourselves 182

yourselves: 9

youth 63

youth: 7

youthful 1

youths 2

zeal 16

zealous 8

zealously 2

zip 2

The output generated and the Jar file are located on the P2 directory.

**Problem 3) The result of wordcount.jar MapReduce job is an unordered list of words and their frequencies, i.e. zeal 13, youths 1, etc. Write a simple MapReduce program that would flip the words and frequencies and output frequencies vs. words, like: 13 zeal, 1 youths, etc,**

We create a new Java file called FlipWordFreq.java, and we start it out as a copy of the original WordCountP1.Java. Our intention is a little bit different, because we will be reading from an already formatted output from the WordCountP1 class.

So, we make the following changes to the FlipWordFreq class in order to make it work as intended:

* We alter the Mapper class so that it expects the input as the original output file, consisting of a String and Integer pair separated by whitespace. So we modify our function to read the tokens in pairs and only separate them by whitespace on the StringTokenizer class:

19 public static class TokenizerMapper

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

21

22 private IntWritable freq = new IntWritable();

23 private Text word = new Text();

24

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

26 ) throws IOException, InterruptedException {

27 // P3: Expecting to receive a line with the word and the frequency separated by whitespace

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

29 int intFreq = 0;

30 while (itr.hasMoreTokens()) {

31 word.set(itr.nextToken());

32 intFreq = Integer.parseInt(itr.nextToken());

33 freq.set(intFreq);

34 context.write(word, freq);

35 }

36 }

37 }

* We keep the original Reducer class as the Combiner class, removing the filterings for the verse counts and the words that appear more than 7000 times, as those have already been dealt with on Problem 1:

39 public static class IntSumReducer

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

41 private IntWritable result = new IntWritable();

42

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

44 Context context

45 ) throws IOException, InterruptedException {

46 int sum = 0;

47

48 for (IntWritable val : values) {

49 sum += val.get();

50 }

51 result.set(sum);

52

53 context.write(key, result);

54 }

55 }

* For the Reducer, we implement a new class called WordFreqReducer, that is similar to the previous one but inverts the way the output is written (word frequency first, word later).

57 public static class WordFreqReducer

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

59 private IntWritable result = new IntWritable();

60

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

62 Context context

63 ) throws IOException, InterruptedException {

64 int sum = 0;

65

66 // P3: Do not expect to have more than one int, but I will iterate on the collection anyway

67 for (IntWritable val : values) {

68 sum += val.get();

69 }

70 result.set(sum);

71

72 // P3: Inverting the output

73 context.write(result, key);

74 }

75 }

* Finally, we must make changes to the Job object so that we can have different outputs on the Mapper and Reducer functions, and assign the proper classes to their Mapper, Combiner and Reducer roles.

84 Job job = new Job(conf, "flip word freq");

85 job.setJarByClass(FlipWordFreq.class);

86 job.setMapperClass(TokenizerMapper.class);

87 job.setCombinerClass(IntSumReducer.class);

88 job.setReducerClass(WordFreqReducer.class);

89 job.setOutputKeyClass(IntWritable.class);

90 job.setOutputValueClass(Text.class);

91 job.setMapOutputKeyClass(Text.class);

92 job.setMapOutputValueClass(IntWritable.class);

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

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

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

After the changes are made into the Java file, we compile the classes, create the Jar file and run it against the output of the Problem 1 MapReduce job to create our flipped output.

**[cloudera@centos-e185 P3]$ javac -classpath `hadoop classpath` -Xlint:deprecation -d . FlipWordFreq.java**

FlipWordFreq.java:84: warning: [deprecation] Job(org.apache.hadoop.conf.Configuration,java.lang.String) in org.apache.hadoop.mapreduce.Job has been deprecated

Job job = new Job(conf, "flip word freq");

^

1 warning

**[cloudera@centos-e185 P3]$ jar -cvf FlipWordFreq.jar e185/\***

added manifest

adding: e185/assign06/(in = 0) (out= 0)(stored 0%)

adding: e185/assign06/FlipWordFreq$WordFreqReducer.class(in = 1776) (out= 753)(deflated 57%)

adding: e185/assign06/FlipWordFreq.class(in = 2055) (out= 1054)(deflated 48%)

adding: e185/assign06/FlipWordFreq$TokenizerMapper.class(in = 1837) (out= 805)(deflated 56%)

adding: e185/assign06/FlipWordFreq$IntSumReducer.class(in = 1772) (out= 752)(deflated 57%)

**[cloudera@centos-e185 P3]$ hadoop fs -rm -r outputP3**

Deleted outputP3

**[cloudera@centos-e185 P3]$ hadoop jar FlipWordFreq.jar e185.assign06.FlipWordFreq output outputP3**

13/03/20 23:57:30 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is inited.

13/03/20 23:57:30 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is started.

13/03/20 23:57:31 INFO input.FileInputFormat: Total input paths to process : 1

13/03/20 23:57:31 INFO mapreduce.JobSubmitter: number of splits:1

(...)

File Input Format Counters

Bytes Read=173985

File Output Format Counters

Bytes Written=173985

**[cloudera@centos-e185 P3]$ hadoop fs -ls outputP3/**

Found 2 items

-rw-r--r-- 1 cloudera supergroup 0 2013-03-20 23:57 outputP3/\_SUCCESS

-rw-r--r-- 1 cloudera supergroup 173985 2013-03-20 23:57 outputP3/part-r-00000

**[cloudera@centos-e185 P3]$ hadoop fs -cat outputP3/part-r-00000 | wc -l**

16807

**[cloudera@centos-e185 P3]$ hadoop fs -cat outputP3/part-r-00000 | head -20**

2 0

2 000

4 01

4 02

4 03

4 04

4 05

4 06

4 07

4 08

4 09

2 0em

120 1

4 10

2 10px

4 11

6 12

4 13

4 14

6 15

cat: Unable to write to output stream.

**[cloudera@centos-e185 P3]$ hadoop fs -cat outputP3/part-r-00000 | tail -20**

154 you:

296 young

2 young:

30 younger

1 younger:

16 youngest

2 youngest:

1780 your

4 yours

1 yours:

182 yourselves

9 yourselves:

63 youth

7 youth:

1 youthful

2 youths

16 zeal

8 zealous

2 zealously

2 zip

As we can see, the output is the same as before, but with the position of the words and numbers flipped, as requested. The Java file and the output generated are available in the P3 directory included with this document.

**Problem 4) On slide 45 we made a claim that the original WordCount.java was not really needed and that we could have achieved the same result using built in Mappers and reducers in the manner contained in class WordCount2.java.**

**package org.apache.hadoop.examples;**

**public class WordCount2 {**

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

**JobClient client = new JobClient();**

**JobConf conf = new JobConf(WordCount2.class);**

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

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

**conf.setOutputKeyClass(Text.class);**

**conf.setOutputValueClass(LongWritable.class);**

**conf.setMapperClass(TokenCountMapper.class);**

**conf.setCombinerClass(LongSumReducer.class);**

**conf.setReducerClass(LongSumReducer.class); client.setConf(conf);**

**try {**

**JobClient.runJob(conf);**

**} catch (Exception e) { e.printStackTrace(); }**

**}**

**Please try to make the above class work and verify the claim.**

We copy the above class to a Java file we name SimpleWordCount.java, with a few changes to import the necessary classes we need. The code presented uses some MapReduce classes from a different MapReduce package, so we have to change most of our imports. The final code looks like this:

1 package e185.assign06;

2

3 import java.io.IOException;

4

5 import org.apache.hadoop.fs.Path;

6 import org.apache.hadoop.io.LongWritable;

7 import org.apache.hadoop.io.Text;

8 import org.apache.hadoop.mapred.JobConf;

9 import org.apache.hadoop.mapred.JobClient;

10 import org.apache.hadoop.mapred.FileInputFormat;

11 import org.apache.hadoop.mapred.FileOutputFormat;

12 import org.apache.hadoop.mapred.lib.TokenCountMapper;

13 import org.apache.hadoop.mapred.lib.LongSumReducer;

14

15 public class SimpleWordCount {

16

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

18

19 JobClient client = new JobClient();

20 JobConf conf = new JobConf(SimpleWordCount.class);

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

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

23 conf.setOutputKeyClass(Text.class);

24 conf.setOutputValueClass(LongWritable.class);

25 conf.setMapperClass(TokenCountMapper.class);

26 conf.setCombinerClass(LongSumReducer.class);

27 conf.setReducerClass(LongSumReducer.class);

28 client.setConf(conf);

29

30 try {

31 JobClient.runJob(conf);

32 } catch (Exception e) { e.printStackTrace(); }

33

34 }

35 }

After the file is completed, we compile and run the class file against our original input directory.

**[cloudera@centos-e185 P4]$ javac -classpath `hadoop classpath` -Xlint:deprecation -d . SimpleWordCount.java**

**[cloudera@centos-e185 P4]$ jar -cvf SimpleWordCount.jar e185/\***

added manifest

adding: e185/assign06/(in = 0) (out= 0)(stored 0%)

adding: e185/assign06/SimpleWordCount.class(in = 1466) (out= 769)(deflated 47%)

**[cloudera@centos-e185 P4]$ hadoop jar SimpleWordCount.jar e185.assign06.SimpleWordCount input outputP4**

13/03/21 01:19:21 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is inited.

13/03/21 01:19:21 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is started.

13/03/21 01:19:21 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is inited.

13/03/21 01:19:21 INFO service.AbstractService: Service:org.apache.hadoop.yarn.client.YarnClientImpl is started.

(...)

File Input Format Counters

Bytes Read=5258976

File Output Format Counters

Bytes Written=717079

**[cloudera@centos-e185 P4]$ hadoop fs -ls outputP4/**

Found 2 items

-rw-r--r-- 1 cloudera supergroup 0 2013-03-21 01:19 outputP4/\_SUCCESS

-rw-r--r-- 1 cloudera supergroup 717079 2013-03-21 01:19 outputP4/part-00000

**[cloudera@centos-e185 P4]$ hadoop fs -cat outputP4/part-00000 | wc -l**

60756

**[cloudera@centos-e185 P4]$ hadoop fs -cat outputP4/part-00000 | head -20**

"Defects," 2

"Information 2

"Plain 4

"Project 10

"Right 2

#7999] 2

$5,000) 2

'AS-IS' 2

("the 2

($1 2

(801) 2

(According 1

(After 1

(Also 1

(And 6

(As 4

(Beforetime 1

(But 2

(For 34

(Howbeit 1

cat: Unable to write to output stream.

**[cloudera@centos-e185 P4]$ hadoop fs -cat outputP4/part-00000 | tail -20**

youth. 15

youth: 7

youth; 8

youth? 2

youthful 1

youths 1

youths, 1

zeal 13

zeal, 3

zealous 8

zealously 2

{ 12

{background:#faebd7; 68

{border: 2

{color:#A82C28} 66

{float: 4

{font-size: 2

{font-size:14pt} 66

{font-size:16pt} 66

} 8

The output is consistent with what we observed on the original WordCount application presented in class. The Java file and the output generated are available in the P4 directory included with this document.

**Problem 5) Hadoop’s HDFS API allows you to manipulate files and data programmatically. When running your MapReduce jobs Hadoop prefers to work with one file rather that many. For whatever reasons, there appears to be no utility that merges files. The attached class PutMerger.java attempts to make up for that deficiency. Please try to fix the class if anything is wrong with it and then examine whether it could truly merge two files into one.**

We copied the file to a specific directory on the VM and tried to compile it after a quick inspection of the source code.

**[cloudera@centos-e185 Assign06]$ cd P5/**

**[cloudera@centos-e185 P5]$ ls**

PutMerge.java

**[cloudera@centos-e185 P5]$ vi PutMerge.java**

**[cloudera@centos-e185 P5]$ javac -classpath `hadoop classpath` -Xlint:deprecation -d . PutMerge.java**

**[cloudera@centos-e185 P5]$ jar -cvf PutMerge.jar PutMerge.class**

added manifest

adding: PutMerge.class(in = 1822) (out= 932)(deflated 48%)

According to the source code, we can see that the PutMerge class expects a local directory as the first parameter and an output file name in HDFS as the second one. So, we created an input directory with some files, as below.

**[cloudera@centos-e185 P5]$ cat input/file1.txt**

This is the first file

**[cloudera@centos-e185 P5]$ cat input/file2.txt**

This is the second file

**[cloudera@centos-e185 P5]$ cat input/file3.txt**

This is the third file

**[cloudera@centos-e185 P5]$ cat input/file4.txt**

This is the fourth file

Finally, we execute the PutMerge class on out input directory and check the output.

**[cloudera@centos-e185 P5]$ hadoop jar PutMerge.jar PutMerge input/ outputP5**

file3.txt

file1.txt

file2.txt

file4.txt

**[cloudera@centos-e185 P5]$ hadoop fs -cat outputP5**

This is the third file

This is the first file

This is the second file

This is the fourth file

As we can see, the combined file in HDFS is a combination of the files in the input directory. The Java file and the output generated are available in the P5 directory included with this document.