Hanjiao Qiu
CSCIE63 Big Data Analytics
Assignment 04

Assignment 04 Solution

Problem 1. Write a working `WordCount` program using Spark Java API that reads a file, e.g. Ulysis/4300.txt from an HDFS directory and writes the results of your calculations to an HDFS file. To improve your word count, remove any punctuation that might have attached itself to your words. Also transform all words into lower case so that the capitalization does not affect the word count. The original code used in lecture notes is provided in the attached `mini-example-java.tar` file. That archive also contains Maven's `pom.xml` file. Run your program and demonstrate that it works. Submit working code inside the customary MS Word Document. Describe steps in your program.

Solution:

1. Source code for this problem is WordCountSpark.java. Before splitting the line into words, I first use a function "replaceAll()" to replace all punctuations with space. The regular expression "[^a-zA-Z\\d]" means any number of characters other than alphabetic or numbers. Then I transform them into lower cases by using "toLowerCase()".

2. The words might be separated by multiple spaces since I replace punctuations with space. So when splitting, I first use "trim()" to remove leading or trailing spaces attached to the line, then split the line using regex "\\s+". It will match multiple consecutive spaces. Then all the null words will be excluded when calculate.

3. Another way is to list all the punctuations and replace them with "". Since we haven't insert more spaces, we can split the line based on one space. This will cause the results a little bit different. For example, the word "www.gutenberg.org". In the first method, this string will be split into three words since dots will be replaced by space. In the second method, it will become one word after removing the dots directly.

4. We can also remove the digits as they are not quite meaningful. We just need to change the regex to "[^a-zA-Z]".

5. We can also extract this step out to a separate map() function and put it before flatMap(). I document this method into the comments below.

Snippet of the code:

```
// Split up into words.
JavaRDD<String> words = input.flatMap(new FlatMapFunction<String, String>() {
    public Iterable<String> call(String x) {

        // First remove punctuation and transform all words into lower case before split.

        return Arrays.asList(x.replaceAll("[^a-zA-Z\\d]", "
").toLowerCase().trim().split("\\s+"));
```

```
        // Another way to remove punctuations:
        // return Arrays.asList(x.replaceAll("[.,;:$%&'+/\\*\\#\\-\\?!\\\"\\[\\]\\(\\)\\{\\}_]",
"").toLowerCase().trim().split(" "));

        // Since the count of digits isn't quite meaningful, we can exclude the digits too.
        // We can only remove punctuation by using:  x.replaceAll("[^a-zA-Z]", " ")

        // We can also extract this step out and put it before flatMap() like this:
        // input = input.map(s -> s.replaceAll("[^a-zA-Z\\d]", " ").toLowerCase());
    }
});
```

6. To make the results more clear and easy to compare, I sort the results by using a simple fuction “sortByKey()”. The output will sort the key / value pairs based on the key and list them alphabetically. We can also sort them based on the value by swapping the key value pairs, sorting by keys and swapping back. I’ve implemented this in Problem 4 in Python.

```
// Sort by key and save to file.
counts.sortByKey().saveAsTextFile(outputFile);
```

7. Compile the code (first method) and list the results.

```
[cloudera@localhost mini-examples-java]$ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building example 0.0.1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ spark-cscie63 ---
[INFO] Deleting /home/cloudera/Documents/hw04/mini-examples-java/target

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.189 s
[INFO] Finished at: 2016-02-25T18:45:47-08:00
[INFO] Final Memory: 25M/63M
[INFO] ------------------------------------------------------------------------


[cloudera@localhost mini-examples-java]$ hadoop fs -ls ulysis
Found 1 items
-rw-r--r--   1 cloudera supergroup    1573079 2016-02-23 09:44 ulysis/4300.txt

[cloudera@localhost mini-examples-java]$ spark-submit --class edu.hu.examples.WordCountSpark
./target/spark-cscie63-0.0.1.jar ulysis/4300.txt wordcounts_java
```

File will be read from and write to HDFS.

```
[cloudera@localhost mini-examples-java]$ hadoop fs -ls wordcounts_java
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2016-02-25 18:51 wordcounts_java/_SUCCESS
-rw-r--r--   1 cloudera supergroup     369829 2016-02-25 18:51 wordcounts_java/part-00000

[cloudera@localhost mini-examples-java]$ hadoop fs -cat wordcounts_java/part-00000 | tail -20
```

```
(zigzagging,1)
(zigzags,1)
(zinfandel,9)
(zingari,1)
(zion,5)
(zip,1)
(zivio,1)
(zmellz,1)
(zodiac,2)
(zodiacal,2)
(zoe,106)
(zones,1)
(zoo,2)
(zoological,1)
(zouave,2)
(zrads,4)
(zulu,1)
(zulus,1)
(zurich,1)
(zut,1)
```

8. This is the results using the second method (list the punctuations in regex).

```
[cloudera@localhost mini-examples-java]$ hadoop fs -ls wordcounts_java_2
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2016-02-25 18:58 wordcounts_java_2/_SUCCESS
-rw-r--r--   1 cloudera supergroup     377615 2016-02-25 18:58 wordcounts_java_2/part-00000

[cloudera@localhost mini-examples-java]$ hadoop fs -cat wordcounts_java_2/part-00000 | tail -20
(zinfandel,8)
(zinfandels,1)
(zingari,1)
(zion,5)
(zivio,1)
(zmellz,1)
(zodiac,2)
(zodiacal,2)
(zoe,103)
(zoefanny,1)
(zoes,2)
(zones,1)
(zoo,2)
(zoological,1)
(zouave,1)
(zouaves,1)
(zrads,4)
(zulu,1)
(zulus,1)
(zut,1)
```

Problem 2. Write a working `WordCount` program using Spark Scala API that reads a file, e.g. Ulysis/4300.txt from a local file system directory and writes the results of your calculations to a local file. To improve your word count, remove any punctuation that might have attached itself to your words. Also transform all words into lower case so that the capitalization does not affect the word count. The original code is provided in the attached `mini-example-scala.tar` file. That archive also contains Scala Build Tool `build.sbt` file. Run your program and demonstrate that it works. Submit working code inside the customary MS Word Document. Describe steps in your program.

Solution:

1. Source code for this problem is WordCountSpark.scala. I use the same idea as Problem 1. Scala uses the same API for this problem as Java. I try both methods here too. The answer should be the same as Problem 1.

```
 // Split up into words.
 val words = input.flatMap(line => line.replaceAll("[^a-zA-Z\\d]", " ").toLowerCase().trim().split("\\s+"))

// val words = input.flatMap(line => line.replaceAll("[.,;:$%&'+/\\*\\#\\-\\?!\\\"\\[\\]\\(\\)\\{\\}_]", "").toLowerCase().trim().split(" "))
```

At last, sort key / value pairs by key.

```
// Save the word count back out to a text file, causing evaluation.
counts.sortByKey().saveAsTextFile(outputFile)
```

2. Compile the code (first method) and list the results.

```
[cloudera@localhost mini-examples-scala]$ sbt clean package
[info] Set current project to mini-example (in build file:/home/cloudera/Documents/hw04/mini-examples-scala/)
[success] Total time: 1 s, completed Feb 25, 2016 7:11:24 PM
[info] Updating {file:/home/cloudera/Documents/hw04/mini-examples-scala/}mini-examples-scala...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 1 Scala source to /home/cloudera/Documents/hw04/mini-examples-scala/target/scala-2.10/classes...
[info] Packaging /home/cloudera/Documents/hw04/mini-examples-scala/target/scala-2.10/mini-example_2.10-0.0.1.jar ...
[info] Done packaging.
[success] Total time: 12 s, completed Feb 25, 2016 7:11:35 PM

[cloudera@localhost mini-examples-scala]$ spark-submit --class edu.hu.examples.WordCountSpark ./target/scala-2.10/mini-example_2.10-0.0.1.jar file:///home/cloudera/Downloads/4300.txt file:///home/cloudera/Documents/hw04/wordcounts_scala
```

The file will be read from and write to local file system (Path start with "file:///").

```
[cloudera@localhost mini-examples-scala]$ ls -l ~/Documents/hw04/wordcounts_scala
total 364
-rw-r--r--. 1 cloudera cloudera 369829 Feb 25 19:15 part-00000
-rw-r--r--. 1 cloudera cloudera      0 Feb 25 19:15 _SUCCESS

[cloudera@localhost mini-examples-scala]$ cat ~/Documents/hw04/wordcounts_scala/part-00000 | tail -20
(zigzagging,1)
(zigzags,1)
(zinfandel,9)
(zingari,1)
(zion,5)
(zip,1)
```

```
(zivio,1)
(zmellz,1)
(zodiac,2)
(zodiacal,2)
(zoe,106)
(zones,1)
(zoo,2)
(zoological,1)
(zouave,2)
(zrads,4)
(zulu,1)
(zulus,1)
(zurich,1)
(zut,1)
```

The file size and content is identical to Problem 1. Below is the results for the second method, still identical to Problem 1.

```
[cloudera@localhost mini-examples-scala]$ ls -l ~/Documents/hw04/wordcounts_scala_2
total 372
-rw-r--r--. 1 cloudera cloudera 377615 Feb 25 19:24 part-00000
-rw-r--r--. 1 cloudera cloudera      0 Feb 25 19:24 _SUCCESS

[cloudera@localhost mini-examples-scala]$ cat ~/Documents/hw04/wordcounts_scala_2/part-00000 |
tail -20
(zinfandel,8)
(zinfandels,1)
(zingari,1)
(zion,5)
(zivio,1)
(zmellz,1)
(zodiac,2)
(zodiacal,2)
(zoe,103)
(zoefanny,1)
(zoes,2)
(zones,1)
(zoo,2)
(zoological,1)
(zouave,1)
(zouaves,1)
(zrads,4)
(zulu,1)
(zulus,1)
(zut,1)
```

Compare two results using the Linux command:

```
[cloudera@localhost hw04]$ hadoop fs -get wordcounts_java .
[cloudera@localhost hw04]$ hadoop fs -get wordcounts_java_2 .
[cloudera@localhost hw04]$ diff wordcounts_java/part-00000 wordcounts_scala/part-00000
[cloudera@localhost hw04]$ diff wordcounts_java_2/part-00000 wordcounts_scala_2/part-00000
```

The same! No difference!

Problem 3. Write a working `WordCount` script using Spark Python API. Read `Ulysis (4300.txt)` file from an HDFS directory and write the results of your calculations to an HDFS file. To improve your word count, remove any punctuation that might have attached itself to your words. Also transform all words into lower case so that the capitalization does not affect the word count. Run your script using `submit-spark` tool and demonstrate that it works. Submit working code. Describe steps in your program in the MS Word document.

Solution:

1. Source code for this problem is WordCountSpark.py. The idea is basically the same as the previous two. However the python API is a little bit different. The python API "replaceAll()" doesn't recognize regex, so I use "re.sub()", which uses the regex package and do the substitution. The regex keeps the same as previous.

2. The python API "split()" will split on multiple consecutive spaces automatically.

3. The python output is unicode, I use "str()" to transform them into string format.

```
from pyspark import SparkContext, SparkConf

import re


conf = SparkConf().setMaster("local").setAppName("wordCount")

sc = SparkContext(conf = conf)

lines = sc.textFile("/user/cloudera/ulysis/4300.txt")

counts = lines.flatMap(lambda line: re.sub(r"[^a-zA-Z\d]", " ", str(line)).lower().split()) \
         .map(lambda word: (word, 1)) \
         .reduceByKey(lambda a, b: a + b)
# counts = lines.flatMap(lambda line: re.replace("[.,;:$%&'+/\\*\\#\\-
\\?!\\\"\\[\\]\\(\\)\\{\\}_]", "", str(line)).lower().split(" ")) \
#           .map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

counts.sortByKey().saveAsTextFile("wordcounts_python")
```

File will be read from / write to HDFS.

4. No need to compile, run directly!

```
[cloudera@localhost mini-examples-python]$ spark-submit WordCountSpark.py

[cloudera@localhost mini-examples-python]$ hadoop fs -cat wordcounts_python/part-00000 | tail -20
('zigzagging', 1)
('zigzags', 1)
('zinfandel', 9)
('zingari', 1)
('zion', 5)
('zip', 1)
('zivio', 1)
('zmellz', 1)
('zodiac', 2)
('zodiacal', 2)
('zoe', 106)
('zones', 1)
('zoo', 2)
('zoological', 1)
('zouave', 2)
```

```
('zrads', 4)
('zulu', 1)
('zulus', 1)
('zurich', 1)
('zut', 1)
```

The answer for the second method.

```
[cloudera@localhost mini-examples-python]$ hadoop fs -cat wordcounts_python_2/part-00000 | tail -
20
('zinfandel', 8)
('zinfandels', 1)
('zingari', 1)
('zion', 5)
('zivio', 1)
('zmellz', 1)
('zodiac', 2)
('zodiacal', 2)
('zoe', 103)
('zoefanny', 1)
('zoes', 2)
('zones', 1)
('zoo', 2)
('zoological', 1)
('zouave', 1)
('zouaves', 1)
('zrads', 4)
('zulu', 1)
('zulus', 1)
('zut', 1)
```

Both methods have identical results to previous problems.

Randomly “grep” some word and compare the counts:

```
[cloudera@localhost hw04]$ hadoop fs -cat wordcounts_python/part-00000 | grep your
('whorusalaminyourhighhohhhh', 1)
('your', 496)
('youre', 8)
('yourn', 1)
('yours', 19)
('yourself', 38)
('yourselves', 2)

[cloudera@localhost hw04]$ hadoop fs -cat wordcounts_java/part-00000 | grep your
(whorusalaminyourhighhohhhh,1)
(your,496)
(youre,8)
(yourn,1)
(yours,19)
(yourself,38)
(yourselves,2)

[cloudera@localhost hw04]$ cat wordcounts_scala/part-00000 | grep your
(whorusalaminyourhighhohhhh,1)
(your,496)
(youre,8)
(yourn,1)
(yours,19)
```

```
(yourself,38)
(yourselves,2)
```

All identical!!

Problem 4. In a Spark API of your choice, write a working `BigramCount` program which would count occurrences of every pair of consecutive words. You should clean your words just as you did in the previous problem by removing punctuations and cases. However, do not count two words separated by a point at the end of a sentence as a bigram. If you are an experienced programmer add to the bigram count word pairs in which the first word is the last word on the line and the second word is the first word on the subsequent line. If you are not an experienced programmer, than do not do it. Test your program on a small text file, where for comparison, you could identify bigrams manually. Run your program on `Ulysis(4300.txt)` file and demonstrate that it works. Provide us with the total count of your bigrams, first 20 bigrams and all bigrams containing word "heaven". Read your file from the local operating system and write results to the local operating system. Include working code in the MS Word Document. Submit the file with the complete working code separately. Describe steps in your program in the MS Word document.

Solution:

1. Source code for this problem is BigramCountSpark.py, BigramCountSparkSimple.py. BigramCountSpark.py counts in word pairs separated by a point at the end of a sentences and sort the bigrams based on their occurrences. The sort is achieved by swapping the key / value pairs, sorting on the key and then swapping them back. BigramCountSparkSimple.py doesn't count in pairs separated by the end of a line, and do the sort using the existing Spark library API.

2. As explained in the comments. I first use "glom()" method to create a single entry for the file, which contains a list of all lines. Then I join these lines up to compose a whole passage. After that, we can split the passage by period, question mark and exclamation mark. The variable "sentences" will get a list of sentences after flatMap() function. In this way, the sentence will not be split over multiple lines.

3. Then we can remove the punctuations and change the sentence to lower cases. We should use map() to split each sentence to a list of words and compose the tuples within current sentence. Thus, we will not compose a tuple using two words separated by ".?!" (across two sentences). For each sentence, we compose a list of tuples (a pair of consecutive words and number 1). And we use flatMap() to put these tuples into one list.

```python
from pyspark import SparkContext, SparkConf

import re


conf = SparkConf().setMaster("local").setAppName("wordCount")

sc = SparkContext(conf = conf)

# lines = sc.textFile("file:///home/cloudera/Downloads/4300.txt")
lines = sc.textFile("/user/cloudera/ulysis/4300.txt")
```

```
# The read in lines are split by lines rather than sentences. So the sentences may be split
# over multiple lines. The glom() method is used to create a single entry for each document
# containing the list of all lines, we can then join the lines up, and then resplit them into
# sentences using "." or "!" or "?" as the separator. Using flatMap() so every object in our
# RDD is now a sentence.
sentences = lines.glom() \
            .map(lambda x: " ".join(x)) \
            .flatMap(lambda x: re.split("[.?!]", x))

bigrams = sentences.map(lambda x: re.sub(r"[^a-zA-Z\d]", " ", str(x)).lower().split()) \
          .flatMap(lambda x: [ ((x[i], x[i + 1]), 1) for i in range(0, len(x) - 1) ])

# If we do not add the pairs in which the first word is the last word on the line and the second
# word is the first word on the subsequent line.
# bigrams = lines.map(lambda x: re.split("[.?!]", str(x))) \
#            .map(lambda x: re.sub(r"[^a-zA-Z\d]", " ", str(x)).lower().split()) \
#            .flatMap(lambda x: [ ((x[i], x[i + 1]), 1) for i in range(0, len(x) - 1) ])

bigramsFreq = bigrams.reduceByKey(lambda a, b: a + b)
```

4. To sort them by value "occurrences / count", I first swap the key, value pairs. The tuple changes from (bigram, count) to (count, bigram). Then I sort them based on the key, which is "count" now. After sort, swap them back.

5. Use "RDD.count()" method will get the total number of bigrams.

6. After sorting them based on the "occurrences", the RDD API "take(20)" will get the first 20 bigrams, which already are the 20 bigrams with highest occurrences (top 20). Since "take()" is an action, we don't get an RDD here. I use local file system "open(), write(), close()" API to write the list into file. "str()" will convert the unicode to string.

7. The "filter()" function will examine each key of the tuples, which is the bigrams and check if there's an exact match for "heaven". If so, the tuple will be selected out.

```
bigramsFreqSort = bigramsFreq.map(lambda x: (x[1], x[0])) \
                  .sortByKey(False) \
                  .map(lambda x: (x[1], x[0]))

print "Total bigrams: %d " % bigramsFreqSort.count()

# Take the first 20 pairs from the sorted list.
topTwentyBigrams = bigramsFreqSort.take(20)

# Another way for sort:
# print bigramsFreq.takeOrdered(20, lambda x : -x[1])

# Fliter out the bigrams contains "heaven"
bigramsWithHeaven = bigramsFreqSort.filter(lambda x: "heaven" in x[0])

bigramsFreqSort.saveAsTextFile("all_bigrams_sorted")

file = open('top_twenty_bigrams', 'w')
```

```
output = ""
for x in topTwentyBigrams:
        output += str(x[0]) + ", " + str(x[1]) + "\n"

file.write(output)
file.close()

bigramsWithHeaven.saveAsTextFile("bigrams_with_heaven")
```

8. The method below is to skip the step to count in bigrams separated by a point at the end of a sentence. The "lines" we get here is a list of lines. For each line, we split them again into a list of sentences separated by ".?!". Then we do the same thing as before, for each list, compose a tuple with bigrams (consecutive words) and count 1.

```
# If we do not add the pairs in which the first word is the last word on the line and the second
# word is the first word on the subsequent line.
bigrams = lines.map(lambda x: re.split("[.?!]", str(x))) \
          .map(lambda x: re.sub(r"[^a-zA-Z\d]", " ", str(x)).lower().split()) \
          .flatMap(lambda x: [ ((x[i], x[i + 1]), 1) for i in range(0, len(x) - 1) ])

bigramsFreq = bigrams.reduceByKey(lambda a, b: a + b)

bigramsFreqSort = bigramsFreq.map(lambda x: (x[1], x[0])) \
                  .sortByKey(False) \
                  .map(lambda x: (x[1], x[0]))
```

9. The API "TakeOrdered" will automatically select out the first 20 bigrams ordered by the value (x[1]). The "-x[1]" means sort by value "occurrence" decreasingly.

```
# Another way for sort:
topTwentyBigrams = bigramsFreq.takeOrdered(20, lambda x : -x[1])
```

10. Directly run the code and check the results.

```
[cloudera@localhost mini-examples-python]$ spark-submit BigramCountSpark.py

Total bigrams: 138268
```

Total number of bigrams is 138268. The file "all_bigrams_sorted" also has 138268 lines.

```
[cloudera@localhost mini-examples-python]$ hadoop fs -cat all_bigrams_sorted/part-00000 | head -
30
(('of', 'the'), 1655)
(('in', 'the'), 1455)
(('on', 'the'), 701)
(('to', 'the'), 630)
(('and', 'the'), 517)
(('of', 'a'), 405)
(('at', 'the'), 379)
(('for', 'the'), 359)
(('from', 'the'), 356)
(('with', 'a'), 351)
(('in', 'a'), 344)
(('to', 'be'), 333)
(('with', 'the'), 329)
```

```
(('he', 'was'), 328)
(('of', 'his'), 328)
(('out', 'of'), 323)
(('by', 'the'), 313)
(('it', 'was'), 292)
(('in', 'his'), 292)
(('he', 'said'), 264)
(('mr', 'bloom'), 234)
(('all', 'the'), 227)
(('he', 'had'), 218)
(('for', 'a'), 187)
(('i', 'was'), 186)
(('don', 't'), 184)
(('and', 'a'), 181)
(('was', 'a'), 173)
(('with', 'his'), 169)
(('and', 'he'), 166)
```

```
[cloudera@localhost mini-examples-python]$ cat top_twenty_bigrams
('of', 'the'), 1655
('in', 'the'), 1455
('on', 'the'), 701
('to', 'the'), 630
('and', 'the'), 517
('of', 'a'), 405
('at', 'the'), 379
('for', 'the'), 359
('from', 'the'), 356
('with', 'a'), 351
('in', 'a'), 344
('to', 'be'), 333
('with', 'the'), 329
('he', 'was'), 328
('of', 'his'), 328
('out', 'of'), 323
('by', 'the'), 313
('it', 'was'), 292
('in', 'his'), 292
('he', 'said'), 264
```

The list above is identical with the first twenty bigrams from the whole list "all_bigrams_sorted".

```
[cloudera@localhost mini-examples-python]$ hadoop fs -cat all_bigrams_sorted/part-00000 | tail -
20
(('a', 'perhaps'), 1)
(('said', 'superpolitely'), 1)
(('express', 'herself'), 1)
(('table', 'or'), 1)
(('ordinary', 'at'), 1)
(('of', 'antrim'), 1)
(('bottle', 'in'), 1)
(('recognised', 'his'), 1)
(('school', 'treat'), 1)
(('mocassins', 'with'), 1)
(('yellow', 'stockings'), 1)
(('fee', 'mendancers'), 1)
(('before', 'rarely'), 1)
(('reflections', 'affected'), 1)
(('writing', 'or'), 1)
(('mermaid', 'coolest'), 1)
```

```
(('boy', 'horhot'), 1)
(('some', 'measure'), 1)
(('still', 'remaining'), 1)
(('a', 'congestion'), 1)
```

The results for the bigrams containing word “heaven”: (This is an exact match).

```
[cloudera@localhost mini-examples-python]$ hadoop fs -cat bigrams_with_heaven/part-00000 | head -
30
(('to', 'heaven'), 9)
(('in', 'heaven'), 9)
(('of', 'heaven'), 7)
(('heaven', 'and'), 4)
(('by', 'heaven'), 2)
(('heaven', 'were'), 2)
(('heaven', 'by'), 2)
(('heaven', 'was'), 2)
(('into', 'heaven'), 1)
(('like', 'heaven'), 1)
(('heaven', 'spilt'), 1)
(('heaven', 'hight'), 1)
(('heaven', 'foretold'), 1)
(('a', 'heaven'), 1)
(('heaven', '4'), 1)
(('and', 'heaven'), 1)
(('heaven', 'ever'), 1)
(('heaven', 'calling'), 1)
(('thank', 'heaven'), 1)
(('heaven', 'murmuring'), 1)
(('seventh', 'heaven'), 1)
(('nearer', 'heaven'), 1)
(('visit', 'heaven'), 1)
(('heaven', 'i'), 1)
(('heaven', 'becomes'), 1)
(('heaven', 'theres'), 1)
(('gracious', 'heaven'), 1)
(('heaven', 'a'), 1)
(('hearts', 'heaven'), 1)
(('heaven', 'theodore'), 1)

[cloudera@localhost mini-examples-python]$ hadoop fs -cat all_bigrams_sorted/part-00000 | grep
heaven
(('to', 'heaven'), 9)
(('in', 'heaven'), 9)
(('of', 'heaven'), 7)
(('the', 'heavens'), 5)
(('heaven', 'and'), 4)
(('the', 'heavenly'), 2)
(('by', 'heaven'), 2)
(('heaven', 'were'), 2)
(('heaven', 'by'), 2)
(('heaven', 'was'), 2)
(('heavens', 'so'), 1)
(('into', 'heaven'), 1)
(('like', 'heaven'), 1)
(('the', 'heavenborn'), 1)
(('a', 'heaventree'), 1)
(('heavenly', 'host'), 1)
(('heavens', 'to'), 1)
(('heavenly', 'weather'), 1)
```

```
(('heaven', 'spilt'), 1)
```

I use "grep" to check my answers for filtering out the bigrams with word "heaven". This is NOT an exact match. The word "heavens", "heavenborn" will be grep out. We don't count this in our solution.

Results for BigramCountSparkSimple.py:

```
[cloudera@localhost mini-examples-python]$ spark-submit BigramCountSparkSimple.py

[cloudera@localhost mini-examples-python]$ hadoop fs -cat all_bigrams_sorted_simple/part-00000 |
head -20
(('of', 'the'), 1564)
(('in', 'the'), 1399)
(('on', 'the'), 668)
(('to', 'the'), 585)
(('and', 'the'), 486)
(('of', 'a'), 396)
(('at', 'the'), 361)
(('with', 'a'), 335)
(('for', 'the'), 335)
(('in', 'a'), 333)
(('from', 'the'), 331)
(('out', 'of'), 318)
(('to', 'be'), 315)
(('of', 'his'), 313)
(('with', 'the'), 313)
(('he', 'was'), 307)
(('by', 'the'), 297)
(('it', 'was'), 282)
(('in', 'his'), 278)
(('he', 'said'), 255)

[cloudera@localhost mini-examples-python]$ hadoop fs -cat bigrams_with_heaven_simple/part-00000 |
head -20
(('to', 'heaven'), 9)
(('in', 'heaven'), 8)
(('of', 'heaven'), 7)
(('heaven', 'and'), 4)
(('by', 'heaven'), 2)
(('heaven', 'by'), 2)
(('heaven', 'was'), 2)
(('into', 'heaven'), 1)
(('like', 'heaven'), 1)
(('heaven', 'spilt'), 1)
(('heaven', 'hight'), 1)
(('a', 'heaven'), 1)
(('heaven', '4'), 1)
(('heaven', 'had'), 1)
(('heaven', 'for'), 1)
(('and', 'heaven'), 1)
(('heaven', 'ever'), 1)
(('heaven', 'calling'), 1)
(('thank', 'heaven'), 1)
(('heaven', 'murmuring'), 1)
```

The counts from the second method is lower than the first one. Since the bigrams separated by the newline haven't been calculated.