DS/CMPSC 410

Lab 2 Solution: A MapReduce Implementation of Basic WordCount

Spring 2021

Instructor: Professor John Yen

TA: Rupesh Prajapati and Dongkuan Xu

Student name: Kangdong Yuan

Learning Objectives:

- Be able to install pyspark
- Be able to use map and reduce in Spark to implement word count.
- Be able to understand the difference between map/reduce in Spark and their similar counterparts in Python: Lazy Evaluation.

This lab include 3 exercises:

- Exercise 1: 5 points
- Exercise 2: 5 points
- Exercise 3: 10 points
- Exercise 4: 7 points
- Exercise 5: 8 points
- Exercise 6: 10 points

Total: 45 points

Due: midnight of Jan 31st (Sunday).

Problem: Given a Large Document, calculate the term frequency (TF) of all its words.

Real World Examples: Google's processing of new/updated webpages to calculate and index their TF for Search Engine.

Part A: A Python Implementation of Word Count

Before we introduce a PySpark implementation, let's first see a Python implementation that uses Python map function. This can help us later to compare the Python implementation with a PySpark implementation.

In [1]:
```
import numpy as np
```

In [2]:
```
from urllib.request import urlopen
import re
def read_url(url):
    return re.sub('\\s+', ' ', urlopen(url).read().decode())
```

In [3]:
```
# Read The Adventures of Huckleberry Finn
huck_finn_url = 'http://introcs.cs.princeton.edu/python/33design/huckfinn.txt'
huck_finn_text = read_url(huck_finn_url)
```

Split the text into words (also referred to as terms or tokens), separated by space.

In [4]:
```
huck_finn_words = huck_finn_text.split(sep = ' ')
print(huck_finn_words[0:50])
```

```
['The', 'Project', 'Gutenberg', 'EBook', 'of', 'The', 'Adventures', 'of', 'Huckleberry', 'Finn,', 'Complete', 'by',
'Mark', 'Twain', '(Samuel', 'Clemens)', 'This', 'eBook', 'is', 'for', 'the', 'use', 'of', 'anyone', 'anywhere', 'at',
'no', 'cost', 'and', 'with', 'almost', 'no', 'restrictions', 'whatsoever.', 'You', 'may', 'copy', 'it,', 'give', 'i
t', 'away', 'or', 're-use', 'it', 'under', 'the', 'terms', 'of', 'the', 'Project']
```

In [5]:
```
word_1_pairs = list( map(lambda x: [x, 1], huck_finn_words) )
print(word_1_pairs[0:50])
```

```
[['The', 1], ['Project', 1], ['Gutenberg', 1], ['EBook', 1], ['of', 1], ['The', 1], ['Adventures', 1], ['of', 1], ['H
uckleberry', 1], ['Finn,', 1], ['Complete', 1], ['by', 1], ['Mark', 1], ['Twain', 1], ['(Samuel', 1], ['Clemens)',
1], ['This', 1], ['eBook', 1], ['is', 1], ['for', 1], ['the', 1], ['use', 1], ['of', 1], ['anyone', 1], ['anywhere',
1], ['at', 1], ['no', 1], ['cost', 1], ['and', 1], ['with', 1], ['almost', 1], ['no', 1], ['restrictions', 1], ['what
soever.', 1], ['You', 1], ['may', 1], ['copy', 1], ['it,', 1], ['give', 1], ['it', 1], ['away', 1], ['or', 1], ['re-u
se', 1], ['it', 1], ['under', 1], ['the', 1], ['terms', 1], ['of', 1], ['the', 1], ['Project', 1]]
```

We want to group all of these key value paris for the same word together so that we can calculate the total time each word occurs in the input text.

One way to achieve this is to transform the list of key value pairs into a Panda Dataframe, then use groupby function of Dataframe.

In [6]:
```
import pandas as pd
df = pd.DataFrame( word_1_pairs, columns=['Word', 'Count'])
print(df)
```

```
             Word  Count
0             The      1
1         Project      1
2       Gutenberg      1
3           EBook      1
4              of      1
...           ...    ...
113340       hear      1
113341      about      1
113342        new      1
113343    eBooks.      1
113344                 1

[113345 rows x 2 columns]
```

In [7]:
```
word_total = df.groupby(['Word']).sum()
print(word_total)
```

```
                Count
Word
                    1
!                  14
!"                  1
!--I                1
"$200               1
...               ...
yourself--just      1
yourself."          2
yourselves          2
yourselves?         1
yuther              7

[13833 rows x 1 columns]
```

Sort the word counts in descending order so that we can easily so what words have highest frequency.

In [8]:
```
sorted_WC = word_total.sort_values(by=['Count'], ascending = False)
```

In [9]:
```
print(sorted_WC.head(50))
```

```
        Count
Word
and      6035
the      4645
I        3041
a        2916
to       2899
was      1941
of       1718
it       1430
he       1372
in       1357
you      1023
that      892
for       819
on        768
all       743
but       738
up        697
we        674
out       645
so        626
got       602
they      577
with      547
his       533
as        531
me        496
him       473
no        461
had       445
she       413
see       409
down      409
at        403
said      399
about     393
my        383
or        377
would     368
them      367
be        363
there     359
then      337
when      336
if        336
by        335
get       331
didn't    318
says:     311
what      310
come      308
```

Part B: A PySpark Implementaion

The first thing we need to do in each Jupyter Notebook running PySpark is to import PySpark first.

In [10]:
```
import pyspark
```

Once we import pyspark, we need to import an important object called "SparkContext". Every spark program needs a SparkContext object.

In [11]:
```
from pyspark import SparkContext
```

We then create a Spark Context variable. Once we have a spark context variable, we can execute spark codes.

Note: We can not create another Spark Context after creating one in an application, unless you first terminate the Spark context using the command

```
sc.stop
```

In [36]:
```
sc=SparkContext("local", "Lab2")
sc
```

Out[36]: **SparkContext**

Spark UI

Version

`v3.0.1`

Master

`local`

AppName

`Lab2`

Exercise 1 (5 points) (a) Add your name below AND (b) replace the path below with the path of your home directory.

Answer for Exercise 1

- a: Your Name: Kangdong Yuan

In [13]:
```
text_RDD = sc.textFile("/storage/home/kky5082/ds410/lab2/words.txt")
text_RDD
```

Out[13]: /storage/home/kky5082/ds410/lab2/words.txt MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0

In [14]:
```
word_RDD = text_RDD.flatMap(lambda line: line.strip().split(" "))
word_RDD
```

Out[14]: PythonRDD[2] at RDD at PythonRDD.scala:53

In [15]:
```
word_pair_RDD = word_RDD.map(lambda word: (word, 1))
word_pair_RDD
```

Out[15]: PythonRDD[3] at RDD at PythonRDD.scala:53

In [16]:
```
word_count_RDD = word_pair_RDD.reduceByKey(lambda a, b: a + b, 1)
```

In [17]:
```
count_word_RDD = word_count_RDD.map(lambda x : [x[1], x[0]] )
```

In [18]:
```
sorted_count_word_RDD = count_word_RDD.sortByKey(ascending=False)
```

In [19]:
```
ssorted_count_word_RDDorted_CW_list = sorted_count_word_RDD.collect()
print(ssorted_count_word_RDDorted_CW_list[0:50])
```

```
[[8, 'State'], [7, 'and'], [7, 'the'], [7, 'of'], [4, 'Penn'], [3, 'University'], [3, 'a'], [3, 'campuses'], [3, 'i
n'], [3, 'campus,'], [3, 'College'], [3, 'has'], [3, 'located'], [2, 'The'], [2, 'is'], [2, 'university'], [2, 'as'],
[2, 'Its'], [2, 'mission'], [2, 'education'], [2, 'Park'], [2, 'Law,'], [1, 'Pennsylvania'], [1, '(Penn'], [1, 'or'],
[1, 'PSU)'], [1, 'state-related,'], [1, 'land-grant,'], [1, 'doctoral'], [1, 'with'], [1, 'facilities'], [1, 'through
out'], [1, 'Pennsylvania.'], [1, 'Founded'], [1, '1855'], [1, "Farmers'"], [1, 'High'], [1, 'School'], [1, 'Pennsylva
nia,'], [1, 'conducts'], [1, 'teaching,'], [1, 'research,'], [1, 'public'], [1, 'service.'], [1, 'instructional'],
[1, 'includes'], [1, 'undergraduate,'], [1, 'graduate,'], [1, 'professional'], [1, 'continuing']]
```

In [20]:
```
text_RDD
```

Out[20]: /storage/home/kky5082/ds410/lab2/words.txt MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0

In [21]:
```
word_RDD
```

Out[21]: PythonRDD[2] at RDD at PythonRDD.scala:53

In [22]:
```
word_pair_RDD
```

Out[22]: PythonRDD[3] at RDD at PythonRDD.scala:53

In [23]:
```
word_count_RDD
```

Out[23]: PythonRDD[9] at RDD at PythonRDD.scala:53

In [24]:
```
word_pair_RDD.collect()
```

Out[24]:
```
[('The', 1),
 ('Pennsylvania', 1),
 ('State', 1),
 ('University', 1),
 ('(Penn', 1),
 ('State', 1),
 ('or', 1),
 ('PSU)', 1),
 ('is', 1),
 ('a', 1),
 ('state-related,', 1),
 ('land-grant,', 1),
 ('doctoral', 1),
 ('university', 1),
 ('with', 1),
 ('campuses', 1),
 ('and', 1),
 ('facilities', 1),
 ('throughout', 1),
 ('Pennsylvania.', 1),
 ('Founded', 1),
 ('in', 1),
 ('1855', 1),
 ('as', 1),
 ('the', 1),
 ("Farmers'", 1),
 ('High', 1),
 ('School', 1),
 ('of', 1),
 ('Pennsylvania,', 1),
 ('Penn', 1),
 ('State', 1),
 ('conducts', 1),
 ('teaching,', 1),
 ('research,', 1),
 ('and', 1),
 ('public', 1),
 ('service.', 1),
 ('Its', 1),
 ('instructional', 1),
 ('mission', 1),
 ('includes', 1),
 ('undergraduate,', 1),
 ('graduate,', 1),
 ('professional', 1),
 ('and', 1),
 ('continuing', 1),
 ('education', 1),
 ('offered', 1),
 ('through', 1),
 ('resident', 1),
 ('instruction', 1),
 ('and', 1),
 ('online', 1),
 ('delivery.', 1),
 ('Its', 1),
 ('University', 1),
 ('Park', 1),
 ('campus,', 1),
 ('the', 1),
 ('flagship', 1),
 ('campus,', 1),
 ('lies', 1),
 ('within', 1),
 ('the', 1),
 ('Borough', 1),
 ('of', 1),
 ('State', 1),
 ('College', 1),
 ('and', 1),
 ('College', 1),
 ('Township.', 1),
 ('It', 1),
 ('has', 1),
 ('two', 1),
 ('law', 1),
 ('schools:', 1),
 ('Penn', 1),
 ('State', 1),
 ('Law,', 1),
 ('on', 1),
 ('the', 1),
 ("school's", 1),
 ('University', 1),
 ('Park', 1),
 ('campus,', 1),
 ('and', 1),
 ('Dickinson', 1),
 ('Law,', 1),
 ('located', 1),
 ('in', 1),
 ('Carlisle,', 1),
 ('90', 1),
 ('miles', 1),
 ('south', 1),
 ('of', 1),
 ('State', 1),
 ('College.', 1),
 ('The', 1),
 ('College', 1),
 ('of', 1),
 ('Medicine', 1),
 ('is', 1),
 ('located', 1),
 ('in', 1),
 ('Hershey.', 1),
 ('Penn', 1),
 ('State', 1),
 ('has', 1),
 ('another', 1),
 ('19', 1),
 ('commonwealth', 1),
 ('campuses', 1),
 ('and', 1),
 ('5', 1),
 ('special', 1),
 ('mission', 1),
 ('campuses', 1),
 ('located', 1),
 ('across', 1),
 ('the', 1),
 ('state.', 1),
 ('Penn', 1),
 ('State', 1),
 ('has', 1),
 ('been', 1),
 ('labeled', 1),
 ('one', 1),
 ('of', 1),
 ('the', 1),
 ('"Public', 1),
 ('Ivies,"', 1),
 ('a', 1),
 ('publicly', 1),
 ('funded', 1),
 ('university', 1),
 ('considered', 1),
 ('as', 1),
 ('providing', 1),
 ('a', 1),
 ('quality', 1),
 ('of', 1),
 ('education', 1),
 ('comparable', 1),
 ('to', 1),
 ('those', 1),
 ('of', 1),
 ('the', 1),
 ('Ivy', 1),
 ('League.', 1)]
```

Exercise 2 (5 points) Modify the path so that you can save the output into your directory.

In [25]:
```
output_file = "/storage/home/kky5082/ds410/lab2/Lab2_WC_sorted_out.txt"

sorted_CW_list = sorted_count_word_RDD.saveAsTextFile(output_file)
```

Exercise 3 (10 points) Use PySpark Map and reduceByKey to implement a "Word/Term Frequency" calculation for the text from "The Adventures of Huckleberry Finn".

In [26]:
```
# Read The Adventures of Huckleberry Finn
```

In [37]:
```
text2_RDD = sc.textFile("/storage/home/kky5082/ds410/lab2/huckfinn.txt")
text2_RDD
```

Out[37]: /storage/home/kky5082/ds410/lab2/huckfinn.txt MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0

In [38]:
```
word2_RDD = text2_RDD.flatMap(lambda line: line.strip().split(" "))
```

In [39]:
```
word_pair2_RDD = word2_RDD.map(lambda word: (word, 1))
```

In [40]:
```
word_count2_RDD = word_pair2_RDD.reduceByKey(lambda a, b: a + b, 1)
```

In [41]:
```
count_word2_RDD = word_count2_RDD.map(lambda x : [x[1], x[0]] )
```

In [42]:
```
sorted_count_word2_RDD = count_word2_RDD.sortByKey(ascending=False)
```

In [43]:
```
sorted_CW_list2 = sorted_count_word2_RDD.collect()
```

In [44]:
```
output_file2 = "/storage/home/kky5082/ds410/lab2/Lab2_HuckFinn_WC_sorted_out.txt"

sorted_CW_list2 = sorted_count_word2_RDD.saveAsTextFile(output_file2)
```

In [47]:
```
sc.stop()
```

Exercise 4 (7 points) Describe the difference between the output of a pySpark command for reading a textfile with the output of a python command for reading a textfile.

Answer for Exercise 4: When we call the reading textfile in python it return a string object, which is the string of textfile we extracted. If we print the slice of this object, it will print the readable text.

In [45]:
```
huck_finn_url = 'http://introcs.cs.princeton.edu/python/33design/huckfinn.txt'
huck_finn_text = read_url(huck_finn_url)
print(type(huck_finn_text))
print(huck_finn_text[0:100])
```

```
<class 'str'>
The Project Gutenberg EBook of The Adventures of Huckleberry Finn, Complete by Mark Twain (Samuel Cl
```

Answer for Exercise 4: When we reading the text by pyspark command, it will return a RDD object, which is a lazy object that need to be computed. If we print this RDD object, it will print the basic information of the RDD object, but it will not print the content of the textfile.

In [46]:
```
text2_RDD = sc.textFile("/storage/home/kky5082/ds410/lab2/huckfinn.txt")
print(type(text2_RDD))
print(text2_RDD)
```

```
<class 'pyspark.rdd.RDD'>
/storage/home/kky5082/ds410/lab2/huckfinn.txt MapPartitionsRDD[11] at textFile at NativeMethodAccessorImpl.java:0
```

Exercise 5 (8 points) Describe how map and reduce in pyspark differs from their counter parts in Python.

Answer for Exercise 5:

1. The map and reduce in pyspark will return a lazy object, which will not be compute untill we use collect function at end. But in python, the result will be computed right after the excution of the function and return the result.

2. In pyspark, the map and reduce partition the data, and these data could be computed on cluster which include many machine. But the map and reduce in python only can run on one machine.

Exercise 6 (10 points) Can map/reduce in Pyspark be used to process a massive dataset (that does not fit in a computer)? If so, why? If not, why?

Answer to Exercise 6:Yes, the pyspark can be used to process massive datase, because pyspark is distributed data processing tool. Pyspark use the lazy object as it's return. The pyspark will compute untill program ask it to compute, so this feature allows program run on cluster which include many machine. The data will be partitioned to suitable size and send to each machine to compute, after the computation in each machine, the data will be reduce and collect to the root machine. So pyspark can handle lagre scale data.