Laboratory 2 - Spark RDD

The objective of this laboratory is to start playing around with Apache Spark, processing large text files.

Problem Specification: Analysis of the file

Run exercise 1. using a Jupyter Notebook (Local). On HDFS, at the path "/data/students/bigdata_internet /lab2/word_frequency.tsv" you have a large text file with the frequencies of words in food reviews in Amazon, in the format word\tfreq , where freq is an integer.

In [1]:
```
#Input file address and read data
input_address = "/data/students/bigdata_internet/lab2/word_frequency.tsv"
readfile = sc.textFile(input_address)
```

Questions

Question 1.0.1:

Can you draw 5 samples from the input RDD? Which command do you use?

Answer:

- **To obtain a sample from an RDD in PySpark, the takeSample() function is utilized. This method returns a local Python list comprising num random elements extracted from the RDD under consideration.**
- **To take sample of data take() function can also be utilized to show first 'n' element of data.**

In [2]:
```
#Take sample
five_samples = readfile.takeSample(False, 5)
print(five_samples)
```

```
['spits\t171', 'Juicyfruit\t3', 'homeland\t16', 'faithful\t97', "drinkers'\t2"]
```

Question 1.0.2:

Now pick the first 5 words in order of frequency.

Answer:

To obtain the result, the top(N, sort_function) function is utilized here; the takeOrdered method also could be used. I wrote it with a def() just as a practice!!!

In [3]:
```
#Function to get 5 most used words
def pairData(list):
    pairList = list.map(lambda line: line.split("\t")).map(lambda pair: (pair[0] , int
    top5words = pairList.top(5, key=lambda x: x[1])
    return top5words
```

In [4]:
```
#show the result
result = pairData(readfile)
print(result)
```

```
[('the', 1630750), ('I', 1448619), ('and', 1237250), ('a', 1164419), ('to', 997979)]
```

Question 1.0.3:

How many words does the file contain?

Answer:

To obtain the result, the number of words should be sum.

In [5]:
```
numberOfWords_without_repeat = readfile.count()
print("Number of words without repetition:", numberOfWords_without_repeat)
```

```
Number of words without repetition: 339819
```

In [6]:
```
numberOfWords = readfile.map(lambda x: int(x.split('\t')[1])).sum()
print("Number of words with repetition:", numberOfWords)
```

```
Number of words with repetition: 45444841
```

Question 1.0.4:

Is word_frequency.tsv a folder or a file?

Answer:

This is a .tsv (Tab-Separated Values) file, which is a type of text file where data is separated by tabs.

Filter words starting with a specified prefix

Keep only the lines containing words that start with the prefix 'ho' .

Questions

Question 1.1.1:

How many lines are left?

Answer:

Using a def() to define the prefix, the number of lines left can be calculated, which is equal to 2327. In this case both 'ho' & 'Ho' are considered as prefix.

In [7]:
```
#Function for filter the words - considering 'ho' & 'Ho' as prefix
def filterWord(list):
    prefix = 'ho'
    wordsWithPrefix = list.filter(lambda line: line.lower().startswith(prefix))
    return wordsWithPrefix
```

In [8]:
```
#Show the result
wordsList = filterWord(readfile)
lines = wordsList.count()
print("Number of words with prefix 'ho' & 'Ho' is : ",lines)
```

```
Number of words with prefix 'ho' & 'Ho' is :  2327
```

Question 1.1.2:

How frequent is the most frequent word of the selected sample (i.e., the maximum value of freq in the lines obtained by applying the filter)?

Answer:

the most frequent word starting with "ho" is "how" with the total number of 36264.

In [9]:
```
words_highest_freq = wordsList.map(lambda x: x.split('\t')).map(lambda pair: (pair[0]
top_word = words_highest_freq.top(1, lambda x: x[1])
print(top_word)
```

```
[('how', 36264)]
```

Question 1.1.3:

Report the code of 3 different ways to solve the task number 1.1.2 (we only want the frequency, i.e., a number and not a tuple/list)

Answer:

In [10]:
```
#Sulution 1
words_New_Solution1 = wordsList.map(lambda x: int(x.split('\t')[1])).sortBy(lambda x:
greatestNumOfFreq = words_New_Solution1.takeOrdered(1, lambda x: -x)[0]
print("Number of the most frequent word:", greatestNumOfFreq)
```

```
Number of the most frequent word: 36264
```

In [11]:
```
#Sulution 2
words_New_Solution2 = wordsList.filter(lambda line: line.split('\t')[0].startswith('ho
words_New_Solution2_tuple = words_New_Solution2.map(lambda line: tuple(line.split('\t'
greatestNumOfFreq = words_New_Solution2_tuple.map(lambda x: x[1]).reduce(lambda x, y:
print("Number of the most frequent word:", greatestNumOfFreq)
```

```
Number of the most frequent word: 36264
```

In [12]:
```
#Sulution 3
words_New_Solution3 = wordsList.filter(lambda line: line.split('\t')[0].startswith('ho
words_New_Solution2_tuple = words_New_Solution3.map(lambda line: tuple(line.split('\t'
greatestNumOfFreq = words_New_Solution2_tuple.sortBy(lambda x: x[1], ascending=False).
print("Number of the most frequent word:", greatestNumOfFreq)
```

```
Number of the most frequent word: 36264
```

Filter most frequent words

In this part of application, among the lines selected by the first filter, you have to apply another filter to select only the most frequent words. The application must select those lines that contain words with a frequency freq greater than 70% of the maximum frequency maxfreq computed before.

In [13]:
```
#Filter the data with threshold
threshold = (70/100)*greatestNumOfFreq
most_frequent_words = words_highest_freq.filter(lambda x: x[1] > threshold)
```

Count the remaining words and save the output

perform the following operations on the selected lines (the ones selected by applying both filters)

Questions

Question 1.3.1:

Count the number of selected lines and print this number on the standard output.

Answer:

The following lines of code demonstrate how to count the number of lines in a filtered list after subtracting the most used words. This process results in a final count of 2325 lines:

In [14]:
```
#The number of selected data, after filtering by threshold
most_frequent_words.count()
```

Out[14]:
```
2
```

In [15]:
```
#Show the result (the two words with most frequency)
most_frequent_words.collect()
```

Out[15]:
```
[('hot', 32944), ('how', 36264)]
```

Question 1.3.2:

Save the selected words (without frequency) in an hdfs output folder. Every line should contain a single word and ends with a semicolumn (;).

Answer:

The code lines, are provided below:

```
In [16]: #Extract the word from the tuple
maximum_frequent_words = most_frequent_words.map(lambda item: item[0])
```

```
In [17]: #Function to add ";" for end of each line
def append_semicolon(word):
    return word + ';'
modified_rdd = maximum_frequent_words.map(append_semicolon)
```

```
In [18]: #save the file
#modified_rdd.saveAsTextFile("Lab2_BigData_PliTo/")
```

Run the application in different ways

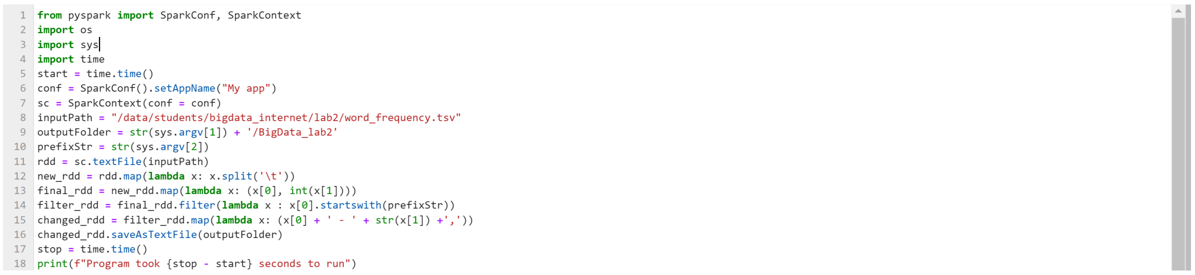
Run exercise 2. using a Python Script.

Question 2.1:

Run your script locally and in the cluster (--master option). How much time do the two modes require to run? Is there a difference? Can you give a plausible explanation?

Answer:

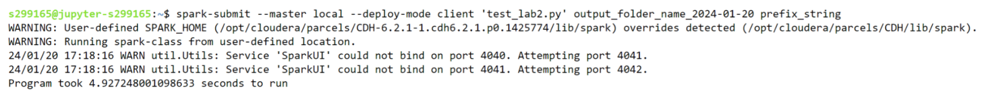
The script below used as the python script,



```python
from pyspark import SparkConf, SparkContext
import os
import sys
import time
start = time.time()
conf = SparkConf().setAppName("My app")
sc = SparkContext(conf = conf)
inputPath = "/data/students/bigdata_internet/lab2/word_frequency.tsv"
outputFolder = str(sys.argv[1]) + '/BigData_lab2'
prefixStr = str(sys.argv[2])
rdd = sc.textFile(inputPath)
new_rdd = rdd.map(lambda x: x.split('\t'))
final_rdd = new_rdd.map(lambda x: (x[0], int(x[1])))
filter_rdd = final_rdd.filter(lambda x : x[0].startswith(prefixStr))
changed_rdd = filter_rdd.map(lambda x: (x[0] + ' - ' + str(x[1]) +','))
changed_rdd.saveAsTextFile(outputFolder)
stop = time.time()
print(f"Program took {stop - start} seconds to run")
```

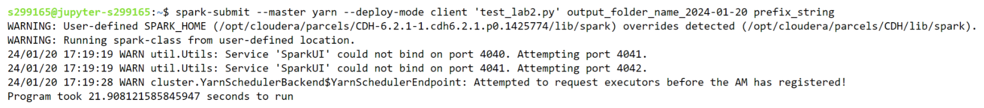
To address the questions:

- **In local mode : Program took 4.927248001098633 seconds to run**



```
s299165@jupyter-s299165:~$ spark-submit --master local --deploy-mode client 'test_lab2.py' output_folder_name_2024-01-20 prefix_string
WARNING: User-defined SPARK_HOME (/opt/cloudera/parcels/CDH-6.2.1-1.cdh6.2.1.p0.1425774/lib/spark) overrides detected (/opt/cloudera/parcels/CDH/lib/spark).
WARNING: Running spark-class from user-defined location.
24/01/20 17:18:16 WARN util.Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
24/01/20 17:18:16 WARN util.Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
Program took 4.927248001098633 seconds to run
```

- In cluster mode: Program took 21.908121585845947 seconds to run



```
s299165@jupyter-s299165:~$ spark-submit --master yarn --deploy-mode client 'test_lab2.py' output_folder_name_2024-01-20 prefix_string
WARNING: User-defined SPARK_HOME (/opt/cloudera/parcels/CDH-6.2.1-1.cdh6.2.1.p0.1425774/lib/spark) overrides detected (/opt/cloudera/parcels/CDH/lib/spark).
WARNING: Running spark-class from user-defined location.
24/01/20 17:19:19 WARN util.Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
24/01/20 17:19:19 WARN util.Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
24/01/20 17:19:28 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
Program took 21.908121585845947 seconds to run
```

Question 2.2:

In this application, would caching an RDD increase the performance? If yes, which RDD would you cache?

In this case, as it is obvious the program took less time to run in local mode compared to cluster mode. As caching might be beneficial when there are stages in the Spark application that operate on the same RDD multiple times, "filter_rdd" might be a good choice.

Bonus Task

On HDFS, at the path /data/students/bigdata_internet /lab2/finefoods_text.txt you have the original huge file that was used to produce the word frequencies that you used in step 1 and 2 of previous tasks (/data/students/bigdata_internet/lab2/word_frequency.tsv)

Question 3.1:

How many words (with repetitions) does it contain? Consider a word all the characters between spaces (elements found with `split()` method)

Answer:

The code lines, with their goal as a comment are provided below:

In [19]:
```python
# Load the RDD from HDFS
input_path = "/data/students/bigdata_internet/lab2/finefoods_text.txt"
readFile = sc.textFile(input_path)

# Split lines into words and flatten the result
words_rdd = readFile.flatMap(lambda line: line.split())
#words_rdd.take(10)
```

In [20]:
```python
word_counting = words_rdd.countByValue()
# Count the total num. of words (with repetitions)
total_words = sum(word_counting.values())

# Show the result
print("Total number of words (with repetitions):", total_words)
```

```
Total number of words (with repetitions): 45444841
```

Question 3.2:

Report the code to obtain the word frequency file starting from the original file.

Answer:

The code lines, with their goal as a comment are provided below; the output of these lines is the same as the original file that we started to work with in this Lab:

In [21]:
```
#Count the frequency
word_freq_rdd = words_rdd.map(lambda word: (word.lower(), 1)).reduceByKey(lambda a, b:
```

In [22]:
```
# Convert to original RDD format
transformed_to_original_rdd = word_freq_rdd.map(lambda x: "{}\t{}".format(x[0], x[1]))
```

In [23]:
```
#save the file
#modified_rdd.saveAsTextFile("Lab2_BigData_PliTo_Bonus/")
```