Aims

This exercise aims to get you to:

- Submitting python script to Spark
- practice more on Spark Programming using RDD with Python

Background

The Spark programming guide is available at:

https://spark.apache.org/docs/latest/rdd-programming-guide.html

The RDD APIs in pyspark can be found here:

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.html

More information about pyspark is available at:

https://spark.apache.org/docs/latest/api/python/getting_started/index.html

The following website is very useful and helpful for learning Spark:

https://sparkbyexamples.com/pyspark-tutorial

To debug your codes, you can first test in the pyspark shell.

Writing your first pySpark Script:

1. Create a python file by any text editor or IDE

Type the following code in SimpleApp.py:

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("SimpleApplication")
sc = SparkContext(conf=conf)

logFile = "README.md"
logData = sc.textFile(logFile, 2).cache()
numAs = logData.filter(lambda line: "a" in line).count()
numBs = logData.filter(lambda line: "b" in line).count()
print(f"Lines with a: {numAs}, Lines with b: {numBs}")
sc.stop()
```

This application computes the number of lines containing “a” and “b” respectively from the testing file. Spark will read the file from Hadoop by default.

Submit your job to Spark

1. Run your application in Spark using a single local thread

Type the following command to submit your application to Spark:

```
$ spark-submit SimpleApp.py
```

You can see the application running and lots of messages are output to the screen.

In order to view the result more clearly, you can do as follows:

```
$ spark-submit SimpleApp.py > temp
```

In the file “temp”, you can see the result:

Lines with a: 71, Lines with b: 38

If you want to make the application run in parallel, you can use more than 1 local thread:

```
$ spark-submit --master local[3] SimpleApp.py > temp
```

2. Connect to Spark standalone cluster URL to run your application

Spark provides a simple standalone deploy mode to deploy Spark on a private cluster. If you are running the application in a cluster, you can do as follows:

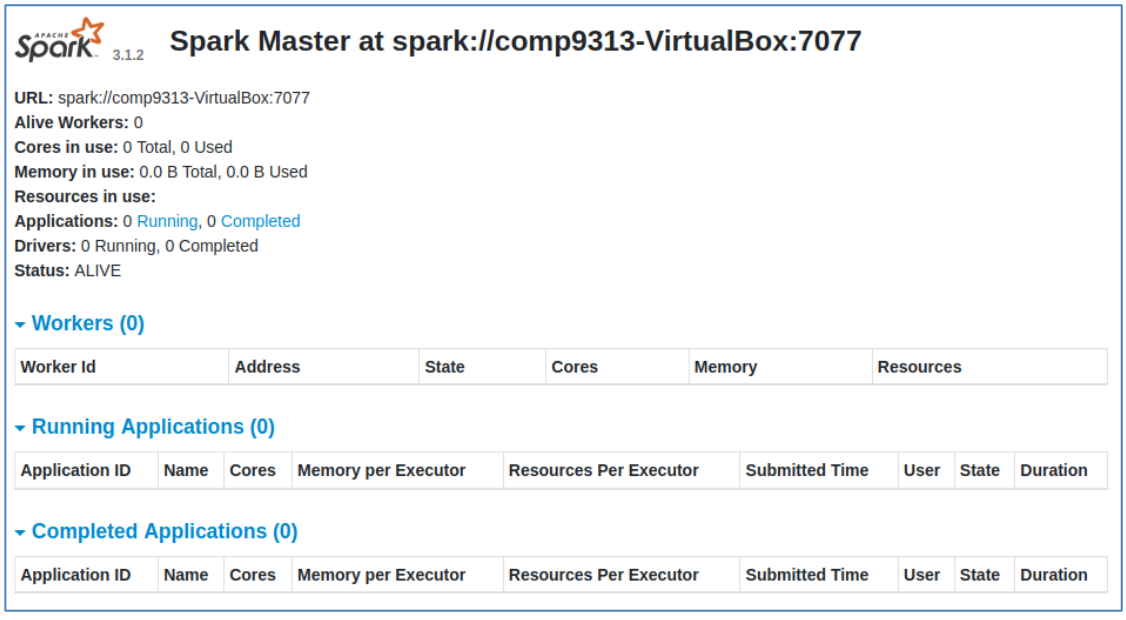
a). Start the Spark standalone Master:

```
$ $SPARK_HOME/sbin/start-master.sh
```

You should see something like the following:

```
comp9313@comp9313-VirtualBox:~/sparkapp$ $SPARK_HOME/sbin/start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /home/comp9313/spark/logs
/spark-comp9313-org.apache.spark.deploy.master.Master-1-comp9313-VirtualBox.out
```

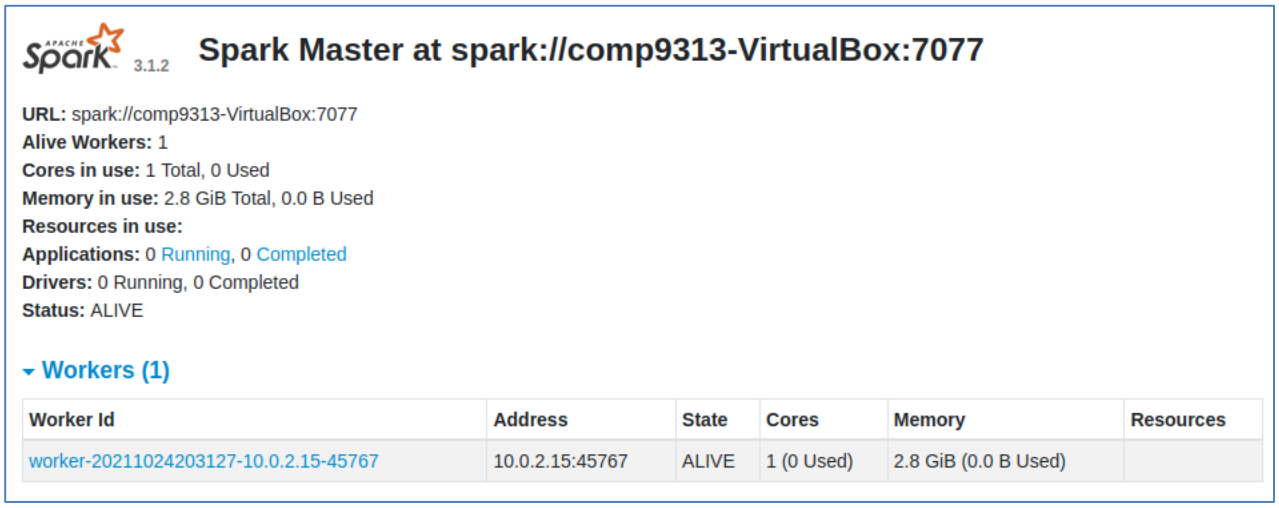
Now you can access http://localhost:8080 to check the cluster information.



b). Start a Worker

```
$ spark-class org.apache.spark.deploy.worker.Worker spark://comp9313-VirtualBox:7077
```

You will see the work on the page http://localhost:8080:



Note: you MUST keep the worker alive in order to run your application!! It means that you cannot stop the spark-class command until your application is finished. You can start more workers to run the application.

c). Run your application in the cluster by specifying the master URL

Type the following command to see the results:

```
$ spark-submit --master spark://comp9313-VirtualBox:7077 SimpleApp.py > temp
```

You can see the application listed in "Completed Applications" on page http://localhost:8080. If you run your application in local threads, it will not be listed there.

Spark RDD Programming

Question 1. Download the input text file pg100.txt from WebCMS3. Write a Spark program which outputs the number of words that start with each letter. This means that for every letter we want to count the total number of words that start with that letter.

- Ignore the letter case, i.e., consider all words as lowercase.
- Ignore terms starting with non-alphabetical characters, i.e., only consider terms starting with "a" to "z".
- Use the space character to split the documents into words:

Hint: File -> an array of words starting with 'a-z' (flatMap and filter) -> an array of pairs (first letter, 1) (map) -> an array of pairs (first letter, total count) (reduceByKey) -> store results to HDFS (saveAsTextFile)

Name your Python file as “problem1.py”. Put the input file in the HDFS folder “/user/comp9313/input”, and store your output in the HDFS folder “user/comp9313/output”. The input and output paths are obtained from the arguments. The result can be found at (the same as that in Lab 3):
https://webcms3.cse.unsw.edu.au/COMP9313/23T3/resources/92011

You can run your job by the following command (HDFS paths can also be used):

```
$ spark-submit problem1.py file:///home/comp9313/pg100.txt
file:///home/comp9313/output
```

Question 2. Download the input text file pg100.txt from WebCMS3. Compute the average length of words starting with each letter. This means that for every letter, you need to compute: the total length of all words that start with that letter divided by the total number of words that start with that letter.

- Ignore the letter case, i.e., consider all words as lower case.
- Ignore terms starting with non-alphabetical characters, i.e., only consider terms starting with “a” to “z”.
- The length of a term X can be obtained by len(X).
- Use the following split function to split the documents into terms:

```
import re
re.split("[\\s*$&#/\"\\,.:;?!\\[\\](){}<>~\\-_]+", line)
```

Your Spark program should generate a list of key-value pairs. Keys and values are separated by “,”, and the values are of double precision, ranked in alphabetical order. You can see the result at:
https://webcms3.cse.unsw.edu.au/COMP9313/23T3/resources/92012

Name your Python file as “Problem2.py”. Put the input file “pg100.txt” in the HDFS folder “/user/comp9313/input”, and store your output in the HDFS folder “user/comp9313/output”. The input and output paths are obtained from the arguments.

You can run your job by the following command (HDFS paths can also be used):

```
$ spark-submit problem2.py file:///home/comp9313/pg100.txt
file:///home/comp9313/output
```

Question 3. Download the sample input file “Votes.csv” from: https://webcms3.cse.unsw.edu.au/COMP9313/23T3/resources/92008, and put it in the HDFS folder “/user/comp9313/input”. In this file, the fields are separated by ‘,’ and the lines are separated by ‘\n’. The data format of “Votes.csv” is as below:

```
- Id
- PostId
- VoteTypeId
   - ` 1`: AcceptedByOriginator
   - ` 2`: UpMod
   - ` 3`: DownMod
   - ` 4`: Offensive
   - ` 5`: Favorite - if VoteTypeId = 5 UserId will be populated
   - ` 6`: Close
   - ` 7`: Reopen
   - ` 8`: BountyStart
```

```
        - ` 9`: BountyClose
        - `10`: Deletion
        - `11`: Undeletion
        - `12`: Spam
        - `13`: InformModerator
        - `14`:
        - `15`:
        - `16`:
    - UserId (only for VoteTypeId 5)
    - CreationDate
```

(i). Find the top-5 VoteTypeIds that have the most distinct posts. You need to output the VoteTypeId and the number of posts. The results are ranked in descending order according to the number of posts, and each line is in the format of:

VoteTypeId\tNumber of posts.

(ii). Find all posts that are favored by more than 10 users. You need to output both PostId and the list of UserIds, and each line is in format of:

PostId#UserId1,UserId2,UserId3,…,UserIdn

The lines are sorted according to the NUMERIC values of the PostIds in ascending order. Within each line, the UserIds are sorted according to their NUMERIC values in ascending order.

You can download the code template at:
https://webcms3.cse.unsw.edu.au/COMP9313/23T3/resources/92014

You can run your job by the following command (HDFS paths can also be used):

```
$ spark-submit problem3.py file:///home/comp9313/Votes.csv
```

You can see the result at:
https://webcms3.cse.unsw.edu.au/COMP9313/23T3/resources/92017