Aims

This exercise aims to get you to:

- Package self-contained Spark applications using sbt
- practice more on Spark Programming using RDD

Background

The detailed Spark programming guide is available at:

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

The RDD transformation and action functions examples are available at:

http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

The documentation of sbt is at:

http://www.scala-sbt.org/1.x/docs/index.html

A tutorial of Scala is available at:

http://docs.scala-lang.org/tutorials/?_ga=1.99469143.850382266.1473265612

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

Install sbt

sbt is a build tool for Scala, Java, etc.

Type the following commands to install sbt:

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https curl gnupg -yqq
$ echo "deb https://repo.scala-sbt.org/scalasbt/debian all main" | sudo tee
/etc/apt/sources.list.d/sbt.list
$ echo "deb https://repo.scala-sbt.org/scalasbt/debian /" | sudo tee
/etc/apt/sources.list.d/sbt_old.list
$ curl -sL
"https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x2EE0EA64E40A89B84B2DF7
3499E82A75642AC823" | sudo -H gpg --no-default-keyring --keyring gnupg-
ring:/etc/apt/trusted.gpg.d/scalasbt-release.gpg --import
$ sudo chmod 644 /etc/apt/trusted.gpg.d/scalasbt-release.gpg
$ sudo apt-get update
$ sudo apt-get install sbt
```

Use the following command to check the version of the sbt. This may take several minutes for the first time of run.

```
$ sbt sbtVersion
```

You will see the output ending with the sbt version number like below:

```
[info] Fetched artifacts of
[info] set current project to comp9313 (in build file:/home/comp9313/)
[info] 1.5.5
```

Package self-contained Spark applications using sbt:

1. Write a self-contained application

Create a working folder for this application, e.g., ~/sparkapp, and create a file SimpleApp.scala in folder ~/sparkapp/src/main/scala

```
$ mkdir -p ~/sparkapp/src/main/scala
$ cd ~/sparkapp/src/main/scala
$ touch SimpleApp.scala
```

Type the following code in SimpleApp.scala:

```
package comp9313.lab6
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "file:///home/comp9313/spark/README.md" //testing
file
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println(s"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.

2. Package your application using sbt

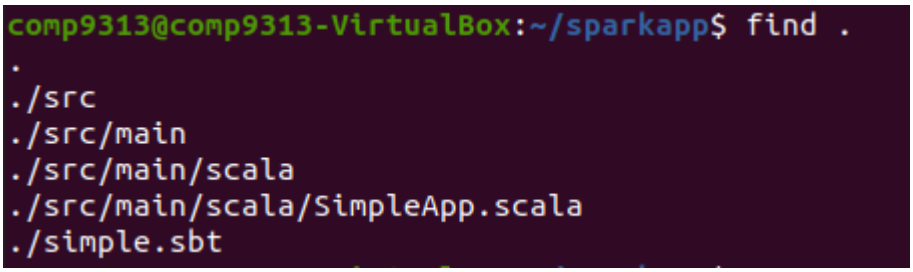
In the folder ~/sparkapp, create a file simple.sbt, and add the following contents:

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.12.10"
libraryDependencies += "org.apache.spark" % "spark-core_2.12" %
"3.1.2"
```

The application depends on the Spark API, and this configuration file explains that Spark is a dependency. This file also adds a repository that Spark depends on.

The scala version and the Spark version can be observed when the Spark shell is started.

For sbt to work correctly, you need to layout SimpleApp.scala and simple.sbt according to the typical directory structure. Your directory layout should look like below (remember that you are working in the folder ~/sparkapp).



Finally, use the following command to package the application:

```
$ sbt package
```

This may take several minutes for the first time of run. sbt will download necessary files from internet.

You can see the following message if this step is successful (the last few lines).

If you run the command again, you will observe that it is much faster.

The generated jar file is located at: ~/sparkapp/target/scala-2.12/simple-project_2.12-1.0.jar

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 --class "comp9313.lab6.SimpleApp" ~/sparkapp/target/scala-2.12/simple-project_2.12-1.0.jar
```

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 --class "comp9313.lab6.SimpleApp" ~/sparkapp/target/scala-2.12/simple-project_2.12-1.0.jar > temp
```

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

Lines with a: 64, Lines with b: 32

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

```
$ spark-submit --class "comp9313.lab6.SimpleApp" --master local[3] ~/sparkapp/target/scala-2.12/simple-project_2.12-1.0.jar > 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:

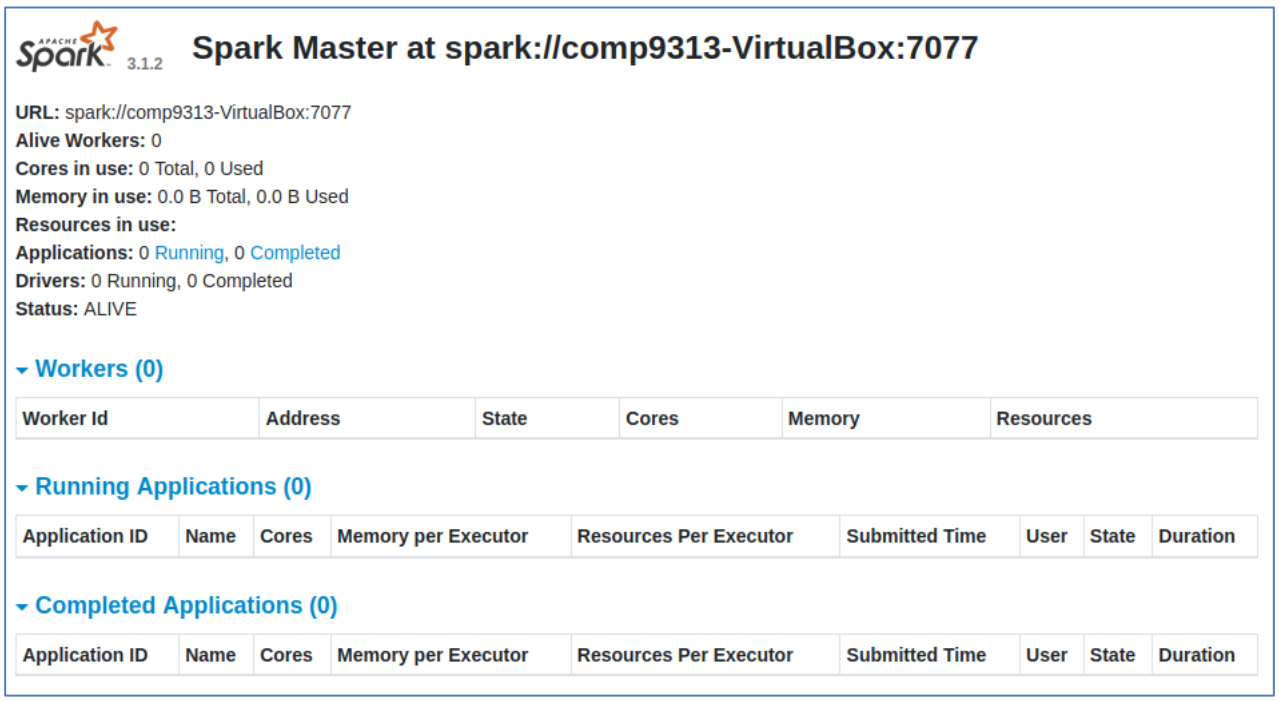
a). Start the Spark standalone Master:

```
$ $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
```

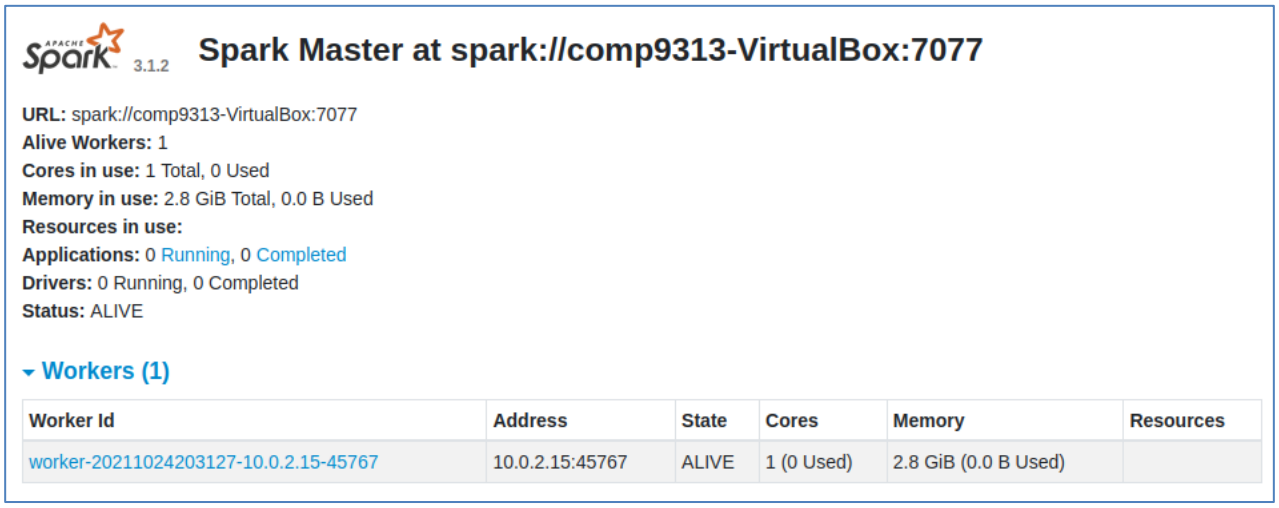
Now you can access http://localhost:8080 to check the cluster information.



b). Start a Worker

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

You will see the work in 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 --class "comp9313.lab6.SimpleApp" --master spark://comp9313-VirtualBox:7077 ~/sparkapp/target/scala-2.12/simple-project_2.12-1.0.jar > temp
```

You can see the application listed in "Completed Applications" at 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 lower case.
- 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 scala file as “Problem1.scala”, the package as “comp9313.lab6”, and the object (or class) as “LetterCount”. Put the input file in HDFS folder “/user/comp9313/input”, and store your output in HDFS folder “user/comp9313/output”. The input and output paths are obtained from the arguments. The answer can be found at (the same as that in Lab 3): https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68733.

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

```
$ spark-submit --class "comp9313.lab6.LetterCount" ~/sparkapp/target/scala-2.12/simple-project_2.12-1.0.jar 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 X.length.
- Use the following split function to split the documents into terms:

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

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/COMP9313/21T3/resources/68729.

Name your scala file as “Problem2.scala”, the object as “WordAverageLen”, and put it in a package “comp9313.lab6”. Put the input file in HDFS folder “/user/comp9313/input”, and store your output in HDFS folder “user/comp9313/output”. The input and output paths are obtained from the arguments.

Question 3. Download the sample input file “Votes.csv” from: https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68728, and put it in 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 format of: VoteTypeId\tNumber of posts.

(ii). Find all posts that are favoured 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.

(Hint: the mkString function is useful to format your output)

You can download the code template at:
https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68731.

You can see the result at:
https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68732.