Aims

This exercise aims to get you to:

- Write self-contained Spark applications using Scala in Eclipse
- Package self-contained Spark applications using sbt

Background

The detailed Spark programming guide is available at:

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

The 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

Write Self-Contained Spark Applications in Eclipse

1. Download Scala-IDE plugins

Open Eclipse, Help-> Install new software…->Add, and enter the following URL:

http://download.scala-ide.org/sdk/lithium/e44/scala211/stable/site

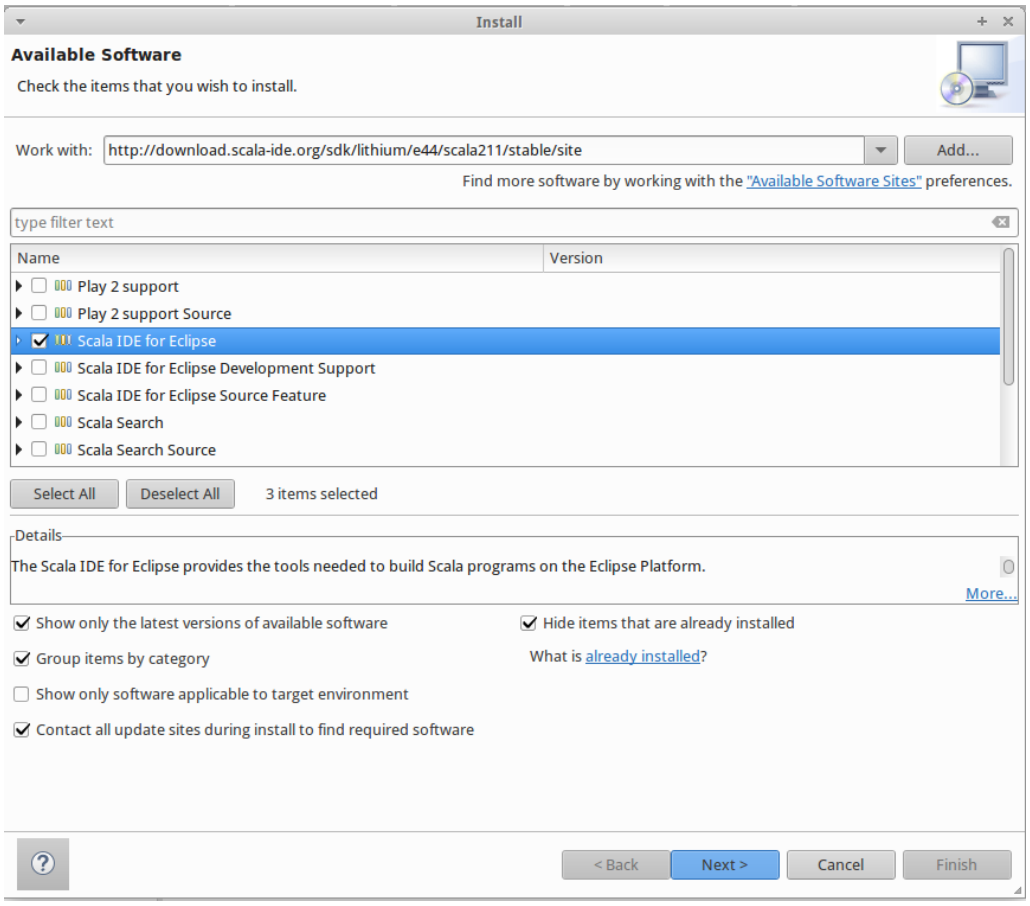
Give a name to this repository, such as “scala-ide”.

Then, select “Scala IDE for Eclipse” to install. It includes:

- Scala 2.11.8 jars
- Sbt 0.13.8 jars
- Scala IDE for Eclipse

Click Next, and accept the terms of the license agreements. The installation may take a few seconds.

You can also download the pre-configured Scala IDE for Eclipse package directly at http://scala-ide.org/download/sdk.html.



2. Select File->New->Project to create a Scala project. Name the project as "WordCountSpark".

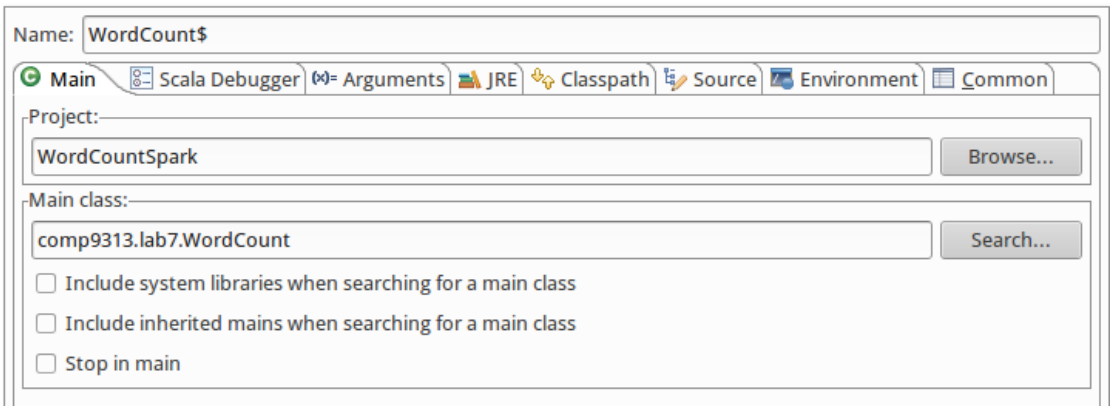
Right click the project, Properties->Java Build Path->Libraries->Add External JARs, go to the directory "/home/comp9313/spark/jars", and add all jars into the project.

3. Create a new package "comp9313.lab7" in this project, and create a file "WordCount.scala" containing the following code:

```
package comp9313.lab7
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
object WordCount {
    def main(args: Array[String]) {
      val inputFile = args(0)
      val outputFolder = args(1)
      val conf = new
SparkConf().setAppName("WordCount").setMaster("local")
      val sc = new SparkContext(conf)
      val input =  sc.textFile(inputFile)
      val words = input.flatMap(line => line.split(" "))
```

```
      val counts = words.map(word => (word, 1)).reduceByKey(_+_)
      counts.saveAsTextFile(outputFolder)
    }
}
```

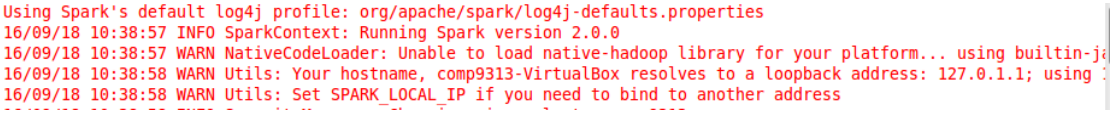
4. Right click the new created file WordCount.scala, and select Run as->Run Configurations->Scala Application. In the dialog, click the tab “Main”, and make input “comp9313.lab7.WordCount” as the “Main class”.



Then configure the arguments for this project: make the arguments as “hdfs://localhost:9000/user/comp9313/pg100.txt hdfs://localhost:9000/user/comp9313/output”. Finally, click “Run”.

Start HDFS and YARN first, and upload pg100.txt to HDFS first!

5. If everything works normally, you will see the Spark running message in Eclipse console:



Wait until the program finishes, and go to HDFS to check the results.

6. You can try more examples given at: http://spark.apache.org/examples.html.

Package self-contained Spark applications using sbt:

1. Install openjdk 8

The installation of sbt requires openjdk 8, which is not installed in the virtual machine. You can install it by:

```
$ sudo apt-get install openjdk-8-jdk
```

If the command cannot be successfully completed, you need to add the openjdk 8 source, and then finish the installation:

```
$ sudo add-apt-repository ppa:openjdk-r/ppa
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk
```

This may take several minutes depending on the network.

You also need to update JAVA_HOME. In ~/.bashrc, edit:

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

Next, check the defaut java version by:

```
$ javac -version
```

If it shows that the version is 1.8.0_141, you can proceed to the next step.

2. Install sbt

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

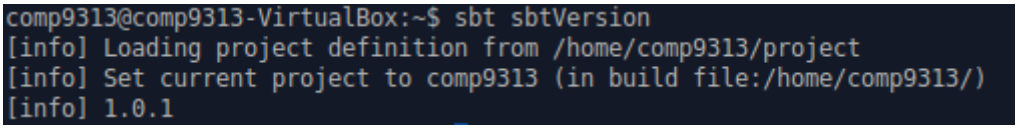
Type the following commands to install sbt:

```
$ echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a
/etc/apt/sources.list.d/sbt.list
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
2EE0EA64E40A89B84B2DF73499E82A75642AC823
$ 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:



3. 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

```
$ makdir -p ~/sparkapp/src/main/scala
$ cd ~/sparkapp
```

Type the following code in this scala file:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "/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.

4. 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 := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0"
```

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 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).

```
comp9313@comp9313-VirtualBox:~/sparkap$ find .
.
./src
./src/main
./src/main/scala
./src/main/scala/SimpleApp.scala
./simple.sbt
```

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).

```
[info]  [SUCCESSFUL ] jline#jline;2.12.1!jline.jar (1316ms)
[info] Done updating.
[info] Compiling 1 Scala source to /home/comp9313/sparkap/target/scala-2.11/clas
ses...
[info] 'compiler-interface' not yet compiled for Scala 2.11.8. Compiling...
[info]   Compilation completed in 29.517 s
[info] Packaging /home/comp9313/sparkap/target/scala-2.11/simple-project_2.11-1.
0.jar ...
[info] Done packaging.
[success] Total time: 589 s, completed 24/04/2017 2:58:39 AM
```

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

The generated jar file is located at: ~/sparkapp/target/scala-2.11/simple-project_2.11-1.0.jar

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

Type the following command to submit your application to Spark:

```
$ spark-submit --class "SimpleApp" ~/sparkapp/target/scala-
2.11/simple-project_2.11-1.0.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 "SimpleApp" ~/sparkapp/target/scala-
2.11/simple-project_2.11-1.0.jar > temp
```

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

Lines with a: 61, Lines with b: 30

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

```
$ spark-submit --class "SimpleApp" --master local[3]
~/sparkapp/target/scala-2.11/simple-project_2.11-1.0.jar > temp
```

6. 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:

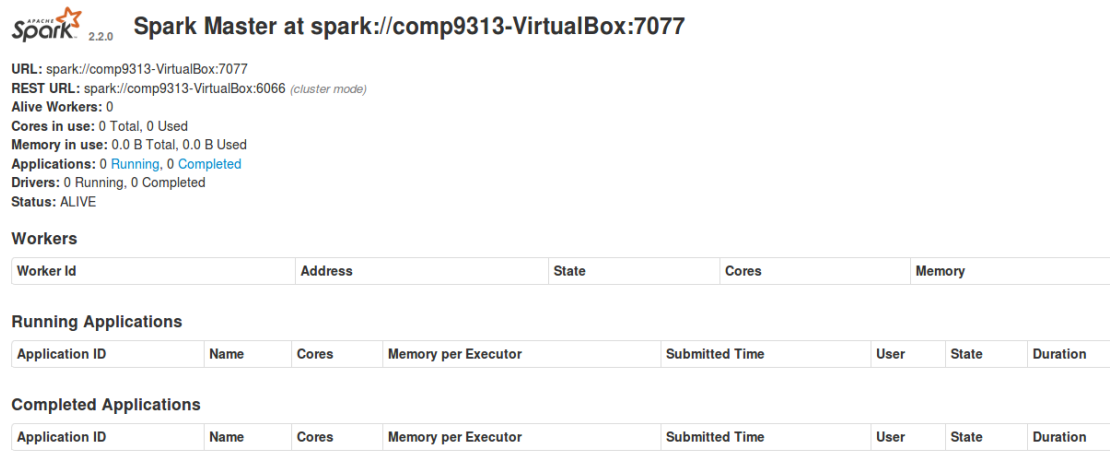
a). Start the Spark standalone Master:

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

You should see something like the following:

```
comp9313@comp9313-VirtualBox:~/sparkap$ ~/spark/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
```

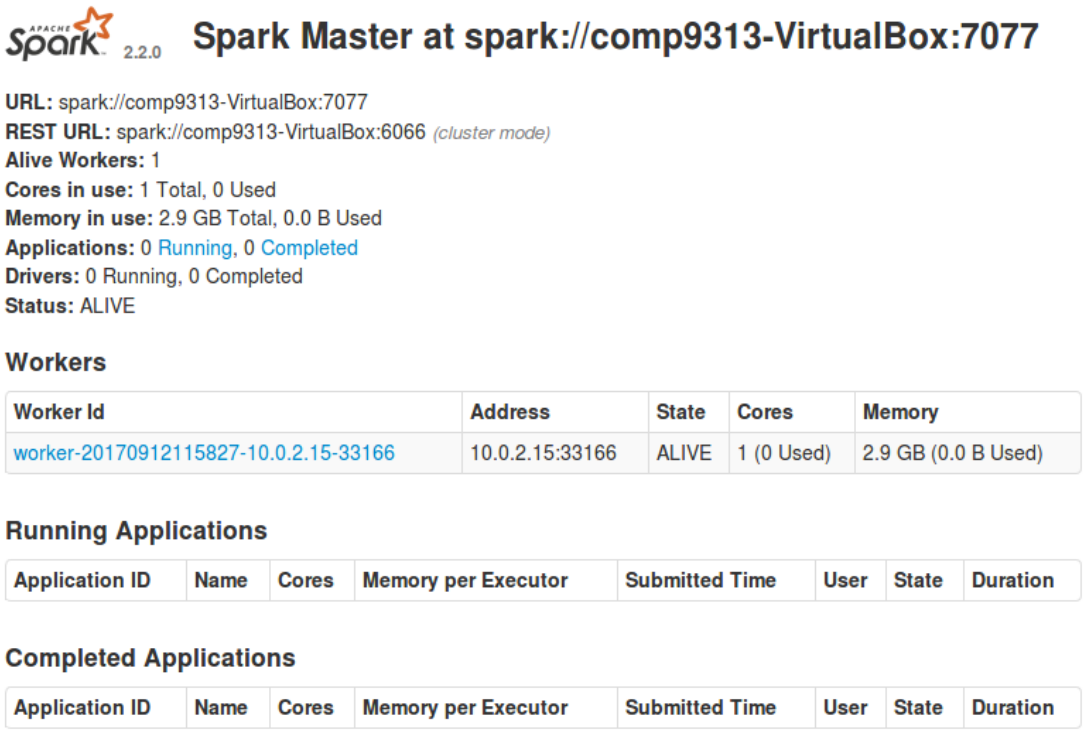
Now you can access http://localhost:8080 to check the cluster information.



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 "SimpleApp" --master spark://comp9313-
VirtualBox:7077 ~/sparkapp/target/scala-2.11/simple-project_2.11-
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.

Question:

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. In your implementation ignore the letter case, i.e., consider all words as lower case. You can ignore all non-alphabetic characters.

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)