Spark Tutorial (Local Machine)

Spring 2022

Junghoon Kang, (Edited by Yuxi Liu)

1 Overview

In this tutorial, you will learn how to install Spark and run a simple Spark application on your local machine. As this tutorial simply concatenates parts of documents provided in Spark's main website (http://spark.apache.org/), please refer to it for more information. Also, note that I am writing this tutorial based on Ubuntu, a Linux distribution. Other Linux distros and Mac OS users can follow this tutorial as the procedures are very similar. However, if you are a Windows user, I highly recommend creating a Ubuntu virtual machine using VMWare. Or you can use the VCM provided by Duke OIT (refer to pages 15-18 in CS516-Spring2022-SQL-installation.pdf) and then avoid possible subsequent problems after modifying environment variables in your own machine, but you should get familiar with how to work without the GUI.

2 Installation

2.1 Ubuntu

1. Install Java (jdk-8u321).

   - If you already have Java installed on your machine, you can skip this. The reason I don't use newer Java is Java 8 is compatible with Spark 1.6.0, and newer versions should work, but with some warnings.
   - Download jdk-8u321 of Java (https://download.oracle.com/otn/java/jdk/8u321-b07/df5ad55fdd604472a86a45a217032c7d/jdk-8u321-linux-x64.tar.gz). You need an Oracle account for downloading. Sign up one for free.
   - Assuming that you download through a browser, and it will be downloaded into `~/Downloads/` (it should be a default path for both Ubuntu and Mac OS).
   - Follow the command lines below to untar the file.

     ```
     $ cd ~
     $ mkdir -p application/java
     $ mv /Downloads/jdk-8u321-linux-x64.tar.gz application/java
     $ cd ~/application/java
     $ tar -xvzf jdk-8u73-linux-x64.tar.gz
     ```

   - Add the following lines in `~/.bashrc`

     ```
     export APPLICATION_HOME=~/application
     export JAVA_HOME=$APPLICATION_HOME/java/jdk1.8.0_321
     export PATH=$JAVA_HOME/bin:$PATH
     ```

- Test whether the Java installation is successful.

```
$ source ~/.bashrc
$ java -version
java version "1.8.0_321"
Java(TM) SE Runtime Environment (build 1.8.0_321-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.321-b07, mixed mode)
```

2. Install SBT (v.1.0.2).

   - Download sbt-1.0.2.tgz of SBT (https://www.scala-sbt.org/download.html).
   - Follow the command lines below to untar the file.

```
$ cd ~
$ mkdir application/sbt
$ mv /Downloads/sbt-1.0.2.tgz application/sbt
$ cd ~/application/sbt
$ tar -xvzf sbt-1.0.2.tgz
$ mv ~/application/sbt/sbt ~/application/sbt/sbt-1.0.2
```

   - Add the following lines in `~/.bashrc`

```
export SBT_HOME=$APPLICATION_HOME/sbt/sbt-1.0.2
export PATH=$SBT_HOME/bin:$PATH
```

3. Install Spark (v.1.6.0).

   - Download spark-1.6.0-bin-hadoop2.6.tgz, which is a prebuilt Spark for Hadoop 2.6 or later (https://archive.apache.org/dist/spark/spark-1.6.0/).
   - The reason I don't use newer Spark is it doesn't include spark-ec2 required for Part-2 anymore.
   - Follow the command lines below to untar the file.

```
$ cd ~
$ mkdir application/spark
$ mv /Downloads/spark-1.6.0-bin-hadoop2.6.tgz application/spark
$ cd ~/application/spark
$ tar -xvzf spark-1.6.0-bin-hadoop2.6.tgz
```

   - Add the following lines in `~/.bashrc`

```
export SPARK_HOME=$APPLICATION_HOME/spark/spark-1.6.0-bin-hadoop2.6
export PATH=$SPARK_HOME/bin:$PATH
export PATH=$SPARK_HOME/ec2:$PATH
```

   - Try running Spark interactive shell, which is inside the `spark-1.6.0-bin-hadoop2.6/bin` directory, by typing:

```
$ source ~/.bashrc
$ spark-shell
```

   - Use :q to quit spark-shell. And you can also test whether spark-ec2 is installed successfully, although it is not required for Part-1.

```
$ spark-ec2
```

2.2 Mac OS

Mac OS users can follow the same steps from above.
If you do not have `~/.bash_profile` file, then you will need to create one. This is where you will be setting the environment variables (see `export` commands) instead of `~/.bashrc`.
Run the following command to identify the path to use to set the `JAVA_HOME` variable in your `~/.bash_profile` file.

```
$ /usr/libexec/java_home -V
...jdk/Contents/Home
```

2.3 VCM (Ubuntu 18)

VCM has 2GB base memory, which is enough for HW2. As above, a challenge of using VCM is to work with terminal. If you want a graphical desktop, try this link:
https://vcm.duke.edu/help/14. I will stick on using CLI (Command-Line Interface) only in the following. Since the procedure of using VCM is almost the same, I will only give additional tools you need.

1. scp : secure copy protocol (SCP) is a means of securely transferring computer files between a local host and a remote host or between two remote hosts.

   - In the CLI of your local machine, transfer a file from the local host (in current directory) to the remote host and vice versa.

     ```
     $ scp [file] ab123@vcm-12345.vm.duke.edu:~/
     $ scp ab123@vcm-12345.vm.duke.edu:~/[file] .
     ```

   - Replace `[file]` with the filename. Replace `ab123` with your own netid. Replace `vcm-12345.vm.duke.edu` with your own hostname of the VCM.
   - It requires your password for your netid/passphrase for you SSH key.

2. wget : wget is a computer program that retrieves content from web servers

   - In the CLI of the vcm, download sbt and scala. JDK's url is also available after loggin in Oracle.

     ```
     $ cd ~/application/sbt/
     $ wget https://github.com/sbt/sbt/releases/download/v1.0.2/sbt-1.0.2.tgz
     $ cd ~/application/spark/
     $ wget https://archive.apache.org/dist/spark/spark-1.6.0/ \
     spark-1.6.0-bin-hadoop2.6.tgz
     $ cd ~/application/java/
     $ wget [jdk's url]
     ```

3. how to implement a code/modify a file?

   - vim : implement code in CLI
   - Visual Studio Code : implement code remotely from your local machine with a graphical window by setting up SSH in vscode.

3 Spark Application

3.1 Write

Open up a text editor and copy-paste the following code into the `WordCount.scala` file. This application simply counts the number of words in an input textfile.

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

object WordCount {
    def main(args: Array[String]) {
        val conf = new SparkConf()
            .setMaster("local[*]")
            .setAppName("WordCount")

        val sc = new SparkContext(conf)

        // Read in the input text file.
        // Then, for each line in the text file, apply remove_punctuation() function.
        val lines_rdd = sc
            .textFile("YOUR_INPUT_FILE.txt")  // {pass in your own input file}
            .map(remove_punctuation)

        // For each data (line string) in lines_rdd, split it into words.
        // Then, filter out empty strings.
        val words_rdd = lines_rdd
            .flatMap( line => line.split("\\s+") )
            .filter( word => word != "" )

        // For each data (word string) in words_rdd, create a (word,1) tuple.
        // Then, count the number of occurrences for each word.
        val wordcounts_rdd = words_rdd
            .map( word => (word, 1) )
            .reduceByKey( (a, b) => (a + b) )

        // Print the top 15 words which occurs the most.
        wordcounts_rdd
            .takeOrdered(10)(Ordering[Int].reverse.on(x => x._2))
            .foreach(println)
    }

    def remove_punctuation(line: String): String = {
        line.toLowerCase
            .replaceAll("""[\p{Punct}]""", " ")
            .replaceAll("""[^a-zA-Z]""", " ")
    }
}
```

You can find the details of each function under API Docs tab in the following website:
http://spark.apache.org/docs/latest/

3.2 Compile

As our application depends on the Spark API, we will include an sbt configuration file, `build.sbt`, which describes the dependencies of the application. Open up a text editor and copy-paste the following lines into the `build.sbt` file.

```
/* build.sbt */
name := "SparkApp"
version := "1.0"
scalaVersion := "2.10.5"
libraryDependencies += "org.apache.spark" %% "spark-core" % "1.6.0"
```

For `sbt` to work correctly, we will need to layout `WordCount.scala`, `YOUR_INPUT_FILE.txt` and `build.sbt` files according to the typical directory structure. Your directory layout should look something like below when you type `find` command inside your application directory. Suppose the absolute path of your application directory is `/home/ab123/sparkapp` (you can work in any folder, as long as it follows the directory structure as the following). Then,

```
$ cd /home/ab123/sparkapp
$ find .
.
./build.sbt
./src
./src/main
./src/main/scala
./src/main/scala/WordCount.scala
./YOUR_INPUT_FILE.txt
```

Once that is in place, we can create a JAR package containing the application code.

```
$ cd /home/ab123/sparkapp
$ sbt package
...
...
[success] Total time: ...
```

If you are a Mac OS user and having a compilation error, please read:
http://stackoverflow.com/questions/5748451/why-do-i-need-semicolons-after-these-imports
In short, do not use the native Mac text editor but use third-party text editor, such as eclipse, vim or emacs, in order to create/edit scala programs.

3.3 Run

You can modify the content in `YOUR_INPUT_FILE.txt`. Then, you can run the application using `spark-submit` script inside `spark-1.6.0-bin-hadoop2.6/bin` directory.

```
$ cd home/ab123/sparkapp
$ spark-submit \
--class WordCount \
target/scala-2.10/sparkapp_2.10-1.0.jar
```

Press enter after the backslash in the terminal (I used backslashes for the readability purpose).

3.4 Result

You might find that it is difficult to extract your result from a bunch of log info. My recommendation is that you can redirect your output to a file with `>`.

```
$ spark-submit \
--class WordCount \
target/scala-2.10/sparkapp_2.10-1.0.jar > YOUR_OUTPUT_FILE.txt
```

Then you can check your result in `YOUR_OUTPUT_FILE.txt`. Notice that `>` will overwrite the file, while `>>` will append the content to the end of the file.