Lab Distributed Big Data Analytics

Worksheet-1: **Setting up environment and getting started with Scala**

Dr. Hajira Jabeen, **Gezim Sejdiu**, **Prof. Dr. Jens Lehmann**

October 17, 2017

Over the lab during the semester, we are going to use a variety of tools, including Apache Hadoop (HDFS)[1], Apache Spark[2], Docker[3] and many more. Instaling this tools and getting started can often be a hassle, mainly because of the dependencies. You are allowed to choose which option : (i) on your own machine; (ii) using Virtualbox[4] and Docker; (iii) or via VMs, to use when you install these packages.

Tasks:

1. **Virtualbox**: - it is a virtualization software package similar to VMWare or other virtual tools. We will make use of it to setup and configure our working environment in order to complete assignments.
   Here are the steps to be followed:
   a) Download the latest Ubuntu ISO from http://www.ubuntu.com/download/desktop (use 64 bit).
   b) Create a new virtual machine with options: `Type = Linux`, `Version = Ubuntu (64 bit)`.
   c) Recommended memory size: 2GB
   d) Select: "Create a Virtual Hard Drive Now".
      i) Leave the setting for Hard Drive File Type unchanged (i.e., VDI).
      ii) Set the hard drive to be "*Dynamically Allocated*".
      iii) Size: ~10GB
   e) The virtual machine is now created.
   f) Press "**Start**"
      i) Navigate to the Ubuntu ISO that you have downloaded, and Press Start.
      ii) On the Boot Screen: "Install Ubuntu"
      iii) Deselect both of "Download Updates while Installing" and "Install Third-Party Software"
      iv) Press "Continue"

[1] http://hadoop.apache.org/
[2] http://spark.apache.org/
[3] https://www.docker.com/
[4] https://www.virtualbox.org/

v) Select "Erase disk and install Ubuntu"
vi) Add your account informations:
vii) Name = "yourname"; username = "username"; password = "****";
viii) Select "Log In Automatically"
ix) Press "Restart Now"
x)

g) Log in to the machine.
i) Open the terminal (Ctrl + Alt + T) and execute these commands:

1) Download and upgrade the packages lists

```
sudo apt-get update
sudo apt-get upgrade
```

2) Installing JAVA

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

- Setting the JAVA_HOME Environment Variable

```
sudo update-alternatives --config java
sudo nano /etc/environment
```

- At the end of this file, add the following line, making sure to replace the highlighted path with your own copied path.

```
JAVA_HOME="/usr/lib/jvm/java-8-oracle"
source /etc/environment
echo $JAVA_HOME
```

3) Install Maven

```
sudo apt-get update
sudo apt-get install maven
```

4) 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 642AC823
sudo apt-get update sudo apt-get install sbt
```

5) Install Hadoop (Single Node Setup)

- Creating a Hadoop user for accessing HDFS

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
```

- Installing SSH

```
sudo apt-get install openssh-server
```

Configuring SSH

#First login with hduser (and from now use only hduser account for further steps).

```
sudo su hduser
```

Generate ssh key for hduser account

```
ssh-keygen -t rsa -P ""
```

Copy id_rsa.pub to authorized keys from hduser

```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

- Installations Steps

Download latest Apache Hadoop source from Apache mirrors

```
wget
http://mirror.nohup.it/apache/hadoop/common/hadoop-2.8.1/hadoo
p-2.8.1.tar.gz
```

Extract Hadoop source

```
tar xzf hadoop-2.8.1.tar.gz
```

```
rm hadoop-2.8.1.tar.gz
```

Move hadoop-2.8.1 to hadoop folder

```
sudo mv hadoop-2.8.1 /usr/local
```

```
sudo ln -sf /usr/local/hadoop-2.8.1/ /usr/local/hadoop
```

Assign ownership of this folder to Hadoop user./

```
sudo chown -R hduser:hadoop /usr/local/hadoop-2.8.1/
```

Create Hadoop temp directories for Namenode and Datanode

```
sudo mkdir -p /usr/local/hadoop/hadoop_store/hdfs/namenode
sudo mkdir -p /usr/local/hadoop/hadoop_store/hdfs/datanode
```

Again assign ownership of this Hadoop temp folder to Hadoop user

```
sudo chown hduser:hadoop -R /usr/local/hadoop/hadoop_store/
```

- Update Hadoop configuration files

User profile : Update $HOME/.bashrc

```
nano ~/.bashrc
```

```
# Set Hadoop-related environment variables
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
# Native path
export
HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
export
HADOOP_OPTS="-Djava.library.path=$HADOOP_PREFIX/lib/native"
# Java path
export JAVA_HOME="/usr/lib/jvm/java-8-oracle"
# Add Hadoop bin/ directory to PATH
export
PATH=$PATH:$HADOOP_HOME/bin:$JAVA_PATH/bin:$HADOOP_HOME/sbin
```

In order to have the new environment variables in place, reload .bashrc

```
source ~/.bashrc
```

- Configure Hadoop

```
cd /usr/local/hadoop/etc/hadoop
nano yarn-site.xml
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>

nano core-site.xml:
<configuration>
  <property>
       <name>fs.defaultFS</name>
       <value>hdfs://localhost:54310</value>
  </property>
</configuration>

cp mapred-site.xml.template mapred-site.xml
nano  mapred-site.xml
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
    <description>The  host  and  port  that  the  MapReduce  job
tracker runs
   at.  If "local", then jobs are run in-process as a single
map
  and reduce task.
  </description>
</property>
</configuration>

nano hdfs-site.xml:
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
```

```
<value>file:/usr/local/hadoop/hadoop_store/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>

<value>file:/usr/local/hadoop/hadoop_store/hdfs/datanode</value>
  </property>
</configuration>
```

- Finally, set to "`/usr/lib/jvm/java-8-oracle`" the JAVA_HOME variable in `/usr/local/hadoop/etc/hadoop/hadoop-env.sh`.

- Starting Hadoop

```
sudo su hduser
hdfs namenode -format
start-dfs.sh
start-yarn.sh
```

- Create a directory on HDFS.

```
hdfs dfs -mkdir /user
hdfs dfs -mkdir /user/hduser
```

- Run a MapReduce job.

```
hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.1.jar pi 10 50
```

- Track/Monitor/Verify

```
jps
```

For ResourceManager – http://localhost:8088

For NameNode – http://localhost:50070

Finally, to stop the hadoop daemons, simply invoke `stop-dfs.sh` and `stop-yarn.sh`.

6) Install Spark

```
mkdir $HOME/spark
cd $HOME/spark
wget http://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-hadoop2.7.tgz
tar xvf spark-2.2.0-bin-hadoop2.7.tgz
nano ~/.bashrc
```

```
export SPARK_HOME=$HOME/spark/spark-2.2.0-bin-hadoop2.7/
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
source ~/.bashrc
start-master.sh
start-slave.sh <master-spark-URL>
spark-shell --master <master-spark-URL>
```

SparkMaster – http://localhost:8080/

Installing Scala

```
wget https://downloads.lightbend.com/scala/2.11.8/scala-2.11.8.tgz
sudo tar xvf scala-2.11.8.tgz
nano ~/.bashrc
export SCALA_HOME=$HOME/scala-2.11.8/
export PATH=$SCALA_HOME/bin:$PATH
source ~/.bashrc
scala -version
```

2. Configure IDE with Scala and Spark

Here are steps how to configure scala-ide for eclipse.

1. Go to http://scala-ide.org/download/sdk.html and download the version needed for your Linux 32 or 64-bit to your linux server/workstation. The latest version is 4.7.0.
2. Meet JDK requirements: JDK 8
3. Copy the archive to your preferred folder and decompress.
4. Find eclipse executable and execute it
    a. Please configure it to work with Scala 2.11.x

After the scala-ide have been configured properly we could set up a spark-template project to get started with Scala and Spark.

```
git clone https://github.com/SANSA-Stack/SANSA-Template-Maven-Spark.git
```

5. Open eclipse.
6. Click File > Import.
7. Type Maven in the search box under Select an import source:
8. Select Existing Maven Projects.
9. Click Next.
10. Click Browse and select the folder that is the root of the Maven project (probably contains the pom.xml file)
11. Click OK.

Assignment:

IN CLASS

---

1. Data Store & Processing using HDFS
   a. Start HDFS and verify its status.
      ```
      hadoop-daemon.sh start namenode
      hadoop-daemon.sh start datanode
      hdfs dfsadmin -report
      ```
   b. Create a new directory /youname on HDFS.
      ```
      hdfs dfs -mkdir /gezim
      ```
   c. Download page_links_simple.nt.bz2, unzip it, on your local filesystem and upload it to HDFS under /yourname folder.
      ```
      hdfs dfs -put page_links_simple.nt /gezim
      ```
   d. View the content and the size of /yourname directory.
      ```
      hdfs dfs -ls -h /gezim/page_links_simple.nt
      ```
   e. Copy the file just created on /yourname into page_links_simple_hdfscopy.nt
      ```
      hdfs dfs -cp /gezim/page_links_simple.nt /gezim/page_links_simple_hdfscopy.nt
      ```
   f. Copy your file back to local filesystem and name it page_links_simple_hdfscopy.nt
      ```
      hdfs dfs -get /gezim/page_links_simple_hdfscopy.nt
      ```
   g. Remove your file from HDFS.
      ```
      hdfs dfs -rm /gezim/page_links_simple.nt
      ```
   h. Remove /yourname directory from HDFS.
      ```
      hdfs dfs -rm -r /gezim
      ```
2. Basics of Scala
   a. Define a class `Point` which describes an (x, y) coordinate.
      ```
      class Point(val x: Int, val y: Int) extends App {
      }
      ```
   b. Create a companion object `Point` such will allow you to instantiate `Point` without using `new`.
      ```
      object Point {
      def apply(x: Int, y: Int) = new Point(x, y)}
      ```
   c. Create a singleton object `Origin` that represents the (0, 0) coordinates of `Point`.
      ```
      object Origin extends Point(0, 0)
      ```
   d. Instantiate the two object of `Origin` and check if both refer to the same object in memory.
      ```
      val p1 = Point
      val p2 = Point
      println(p1.eq(p2))
      ```
   e. Implement a function `distanceTo` which calculates the distance between two `Point` instances.
      ```
      def distanceTo(other: Point): Double = {
      ```

```
        val dx = math.abs(x - other.x)
        val dy = math.abs(y - other.y)
        math.sqrt(math.pow(dx, 2) + math.pow(dy, 2))
    }
```

AT HOME

---

1. Read and explore
    a. Functional Programming
    b. Recursion, and Tail recursion
    c. Anonymous functions
    d. High Order Functions
    e. Currying
2. Read a textfile and do a word count on that file. Hint: create and populate a Map with words as keys and counts of the number of occurrences of the word as values[5].
    a. Hello Hello World
    b. (Hello, 2)
    c. (World, 1)
3. Apply what you have read above by creating a function that returns sum of a range of integers by applying a user defined function to it.

$$\sum_{i=a}^{b} f(i)$$

   e.g. f=cube, and sum(2,4) would result in $2^3+3^3+4^3$.
4. Further readings
    a. http://www.artima.com/scalazine/articles/steps.html
    b. http://www.scala-lang.org/
    c. http://twitter.github.io/scala_school/

[5] http://ampcamp.berkeley.edu/big-data-mini-course/introduction-to-the-scala-shell.html