Recitation 3

Setting up the environment-

- Download Spark
- Add Spark binaries to path.

```
local$ vim ~/.bashrc
# hadoop
export HADOOP_HOME="/path/to/hadoop-2.8.1"
export HADOOP_CONF_DIR="$HADOOP_HOME/etc/hadoop"
# spark
export SPARK_HOME="/path/to/spark/dir"
export PATH="$SPARK_HOME/bin:$PATH"
export PATH="$SPARK_HOME/sbin:$PATH"
```

- Install sbt.

```
$ sudo apt-get update
$ sudo apt-get install sbt
```

- You will require a working Python2 version.

```
# Check installed python version
$ python --version
```

- Clone the Recitation3 repository.

```
$ git clone https://github.com/ece579/ece579_f17.git
$ cd recitation3
```

- Start Hadoop daemons.

Part 1: Spark Installation

- Run the SparkPi example from Spark using the run-example and spark-submit.
- What is the other way you can submit a spark job?

- Go to the `$SPARK_HOME/bin` directory and read the code in the run-example file. What is the code doing? How does run-example launch a spark submit job
- What does the spark submit file do?
- How are Spark environmental variables set?
- Edit Spark environment variables in `conf/spark-env.sh`

  ```
  # add this to your Spark environment
  export HADOOP_HOME="/path/to/hadoop-2.8.1"
  export HADOOP_CONF_DIR="$HADOOP_HOME/etc/hadoop"
  ```

Part 2: Sbt and Scala Wordcount

sbt (Simple Build Tool) is an open-source interactive build tool for Scala and Java projects, similar to Java's Maven and Ant. SBT manages library dependencies internally with Apache Ivy. "You are most likely to benefit from adopting SBT if you're writing a pure Scala Spark application or you have a mixed codebase of both Java and Scala code." Since Hadoop natively runs Java applications, we can also package our Scala spark job into a `*.jar` to and then submit it to Hadoop.

- Go to `$ cd part2/`
- Follow sbt guidelines: http://spark.apache.org/docs/latest/quick-start.html
- Edit build.sbt as per your sbt Scala libraries.
- Execute-

  ```
  $ cd part2/
  # edit directory such that directory structure looks like-
  $ find .
  .
  ./input
  ./input/input.txt
  ./run_wordcount.sh
  ./build.sbt
  ./src
  ./src/main
  ./src/main/scala
  ./src/main/scala/wordcount.scala

  # compile wordcount example
  $ sbt package
  # check target/scala* dir for wordcount jar
  # run compiled program
  $ sbt run
  ```

- Find your sbt executable. What new folders are created?

Part 3: Spark with Ipython Notebook

A Jupyter/Ipython notebook lets you write and execute Python code in your web browser. Jupyter notebooks make it very easy to tinker with code and execute it in bits and pieces; for this reason Jupyter notebooks are widely used in scientific computing.

By convention, Jupyter notebooks are expected to be run from top to bottom. Failing to execute some cells or executing cells out of order can result in errors. Within a Notebook we can easily develop and debug our spark job. Moreover, we can preserve our notebook state for future use.

A brief tutorial on Jupyter Notebook can be found here. We will use Python2 for our recitation.

```
# Python2
# You can install Anaconda distribution or work with system's Python2
# Conda installation- https://www.anaconda.com/download/#download

# Python2 pip
$ pip install --upgrade pip
$ pip install jupyter
$ echo /
"export PYTHONPATH=\"$SPARK_HOME/python:$SPARK_HOME/python/lib/py4j-0.10.4-src.zip\"" /
>> ~/.bashrc

$ cd part3/
$ jupyter notebook spark-notebook.ipynb # run jupyter notebook
```

Part 4: YARN with Python and Spark

Yet Another Resource Negotiator (YARN) is the framework responsible for providing the computational resources (e.g., CPUs, memory, etc.) needed for application executions. A global ResourceManager runs as a master daemon, usually on a dedicated machine, that arbitrates the available cluster resources among various competing applications. The ResourceManager tracks how many live nodes and resources are available on the cluster and coordinates what applications submitted by users should get these resources and when. The ResourceManager is the single process that has this information so it can make its allocation (or rather, scheduling) decisions in a shared, secure, and multi-tenant manner (for instance, according to an application priority, a queue capacity, ACLs, data locality, etc.). For more info regarding YARN operations refer this.

- Another way to execute our Scala Spark application is to use spark-submit

```
# Another way to execute our Scala Spark application is to use spark-submit
# edit paths in run_wordcount.sh as per yours
# execute spark job for the built package
# run Spark job
$ cd part2/
$ bash run_wordcount.sh
```

- Check whether YARN has been used in the Spark job (`run_wordcount.sh`)
- How can you submit a job on YARN?
- Configure yarn by editing `conf/yarn-site.xml` in Hadoop conf directory.
  Add the file `yarn-site.xml` to your `conf/yarn-site.xml`
- Configure yarn-

  ```
  # add this to your .bashrc and yarn-env.sh/yarn-env.cmd
  export HADOOP_HOME="/path/to/hadoop-2.8.1" # path to hadoop dir
  export JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64" # path to java
  ```

- Run the wordcount example using the Makefile.
  `$ cd part4/`
- How do we allocate MapReduce task?
- How much physical memory, is made available to running yarn containers?
- How do you run a spark job on YARN in a cluster?

Part 5: Spark Monte Carlo Sampling

- Read the pi calculation example from:
  https://computing.llnl.gov/tutorials/parallel_comp/#ExamplesPI
  http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopimod.html
- Write the code to approximate pi using monte carlo sampling in parallel with Spark (use python).
  Create a Makefile to execute the code.
- Go to `$ cd part5/`

Part 6: Spark Code

- Go to `$SPARK_HOME/core` folder and navigate the core code of Spark (schedulers, etc)

Part 7: Spark data types

In this part you will learn how to work with spark datatypes. Open the ipython notebook and follow along: `./part3/spark-notebook.ipynb`
Now we will use the sparse matrix in HW1- part-00000

1- Read the matrix in coordinate format. Handin the output that shows you correctly created the RDD in coordinate format.

2- Convert the matrix from coordinate format to row format and show the output that demonstrates that you have correctly done the conversion.

Hints:

- Guidelines to working with datatypes in Spark can be found at: https://spark.apache.org/docs/2.1.0/mllib-data-types.html
- If you need to read the source code visit here: https://github.com/apache/spark/tree/v2.2.0/mllib/src/main/scala/org/apache/spark/mllib