EE412 Foundation of Big Data Analytics, Fall 2018
HW0*

Due date: 09/12/2018 (11:59pm)

The purpose of this tutorial is to get you started with Spark. Here you will learn how to write, compile, debug, and execute a simple Spark program. The first part of the homework assignment serves as a tutorial, and the second part asks you to write your own Spark program.

Section 1 explains how to download and install a stand-alone Spark instance. All operations done in this Spark instance will be performed against the files in your local file system. You may setup a full (single-node) spark cluster if you prefer; the results will be the same. You can find instructions online. If you would like to experiment with the full Spark environment where Spark workloads are executed by YARN against file stored in HDFS, we recommend either the Cloudera Quickstart Virtual Machine: https://www.cloudera.com/downloads/quickstart_vms/5-13.html for a pre-installed single-node cluster or Cloudera Manager:
http://archive.cloudera.com/cm5/installer/latest/cloudera-manager-installer.bin to install a multi-node cluster on machines you control. You can also use Google Cloud Dataproc: https://cloud.google.com/sdk/gcloud/reference/dataproc/jobs/submit/pyspark, which gives $300 credit when you sign up for the first time (instructions).

Section 2 explains how to launch the Spark shell for interactively building Spark applications.

Section 3 explains how to use Spark to launch Spark applications written in an IDE or editor.

Section 4 gives an example of writing a simple word count application for Spark.

Section 5 is the actual homework assignment. There are no deliverables for sections 2, 3, and 4. In section 5, you are asked to write and submit your own Spark application based on the word count example.

Use KAIST KLMS to submit your homeworks. Your submission should be one gzipped tar file whose name is `YourStudentID_hw0.tar.gz`. For example, if your student ID is 20161234, and it is for homework 0, please name the file as `20161234_hw0.tar.gz`.

*Material adapted from Stanford University CS246.

1 Setting up a stand-alone Spark instance

- Download and install Spark 2.3.1 on your machine:
  https://www.apache.org/dyn/closer.lua/spark/spark-2.3.1/spark-2.3.1-bin-hadoop2.7.tgz

- Unpack the compressed TAR ball:
  tar -zxvf spark-2.3.1-bin-hadoop2.7.tgz

Spark requires **JDK 8**. Do not install JDK 9; Spark is currently incompatible with JDK 9. If you need to download the JDK, please visit Oracle's download site: http://www.oracle.com/technetwork/java/ javase/downloads/index.html. To use Python, you will need Python 2.7 or higher or 3.4 or higher.

On Ubuntu Linux, you can run the following commands (http://tipsonubuntu.com/2016/07/31/install-oracle-java-8-9-ubuntu-16-04-linux-mint-18/):

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt install oracle-java8-installer
sudo apt install oracle-java8-set-default
```

In case you want to use Ubuntu Linux using VMware, here are some instructions.

The Haedong Lounge machines (eelab1.kaist.ac.kr ∼ eelab36.kaist.ac.kr) already have JDK 8 installed (you can check by typing `javac -version`).

These commands are just guidelines, and what you have to add to your file may vary depending on your specific computer setup.

2 Running the Spark shell

Spark gives you two different ways to run your applications. The easiest is using the Spark shell, a Read–Eval–Print Loop (REPL) that lets you interactively compose your application. Spark supports Python, which we will use as our language.

To start the Spark shell for Python, do the following:

- Open a terminal window on Mac or Linux or a command window on Windows.
- Change into the directory where you unpacked the Spark binary.
- Run: bin/pyspark on Mac or Linux or bin\pyspark on Windows.

As the Spark shell starts, you may see large amounts of logging information displayed on the screen, possibly including several warnings. You can ignore that output for now. Regardless, the startup is complete when you see something like:

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ ‘/ __/  ’_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Python version 2.7.12 (default, Dec  4 2017 14:50:18)
SparkSession available as ’spark’.
>>>
```

The Spark shell is a full python interpreter and can be used to write and execute regular python programs. For example:

```
>>> print "Hello!"
Hello!
```

The Spark shell can also be used to write Spark applications in python. To learn about writing Spark applications, please read through the Spark programming guide: https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html

3 Submitting Spark applications

The Spark shell is great for exploring a data set or experimenting with the API, but it's often best to write your Spark applications outside of the Spark interpreter using an IDE or other smart editor (e.g., emacs, vim). One of the advantages of this approach for this class is that you will have created a submittable file that contains your application, rather than having to piece it together from the Spark shell's command history.

Python is a convenient choice of language as your Python application does not need to be compiled or linked. Assume you have the following program in a text file called myapp.py:

```
import sys
from pyspark import SparkConf, SparkContext

conf = SparkConf()
sc = SparkContext(conf=conf)
print "%d lines" % sc.textFile(sys.argv[1]).count()
```

This short application opens the file path given as the first argument from the local working directory and prints the number of lines in it. To run this application, do the following:

- Open a terminal window on Mac or Linux or a command window on Windows.
- Change into the directory where you unpacked the Spark binary.
- Run: `bin/spark-submit path/to/myapp.py path/to/pg100.txt` on Mac or Linux or `bin\spark-submit path\to\myapp.py path\to\pg100.txt` on Windows.

(See section 4 for where to find the `pg100.txt` file.) As Spark starts, you may see large amounts of logging information displayed on the screen, possibly including several warnings. You can ignore that output for now. Regardless, near the bottom of the output you will see the output from the application:

```
2018-07-01 23:27:36 INFO  TaskSetManager:54 - Finished task 0.0 in stage 0.0
(TID 0) in 560 ms on localhost (executor driver) (1/2)
2018-07-01 23:27:36 INFO  TaskSetManager:54 - Finished task 1.0 in stage 0.0
(TID 1) in 552 ms on localhost (executor driver) (2/2)
2018-07-01 23:27:36 INFO  TaskSchedulerImpl:54 - Removed TaskSet 0.0, whose
tasks have all completed, from pool
2018-07-01 23:27:36 INFO  DAGScheduler:54 - ResultStage 0 (count at
/home/swhang/myapp.py:5) finished in 0.622 s
2018-07-01 23:27:36 INFO  DAGScheduler:54 - Job 0 finished: count at
/home/swhang/myapp.py:5, took 0.668426 s
124787 lines
2018-07-01 23:27:36 INFO  SparkContext:54 - Invoking stop() from shutdown
hook
2018-07-01 23:27:36 INFO  AbstractConnector:318 - Stopped
Spark@565381a1{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2018-07-01 23:27:36 INFO  SparkUI:54 - Stopped Spark web UI at
http://di.kaist.ac.kr:4040
2018-07-01 23:27:36 INFO  MapOutputTrackerMasterEndpoint:54 -
MapOutputTrackerMasterEndpoint stopped!
2018-07-01 23:27:36 INFO  MemoryStore:54 - MemoryStore cleared
2018-07-01 23:27:36 INFO  BlockManager:54 - BlockManager stopped
2018-07-01 23:27:36 INFO  BlockManagerMaster:54 - BlockManagerMaster stopped
2018-07-01 23:27:36 INFO
OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 -
OutputCommitCoordinator stopped!
2018-07-01 23:27:36 INFO  SparkContext:54 - Successfully stopped SparkContext
2018-07-01 23:27:36 INFO  ShutdownHookManager:54 - Shutdown hook called
2018-07-01 23:27:36 INFO  ShutdownHookManager:54 - Deleting directory
/tmp/spark-5131bd2f-0aff-43fb-88c0-2dd66d88c273/pyspark-680af869-a8fa-4ac6-
b164-664a1f4631ff
2018-07-01 23:27:36 INFO  ShutdownHookManager:54 - Deleting directory
/tmp/spark-5131bd2f-0aff-43fb-88c0-2dd66d88c273
2018-07-01 23:27:36 INFO  ShutdownHookManager:54 - Deleting directory
/tmp/spark-79f09ff7-5624-44c7-8988-f20f0e7ae1da
```

Executing the application this way causes it to be run single-threaded. To run the application with 4 threads, launch it as
`bin/spark-submit --master ’local[4]’ path/to/myapp.py path/to/pg100.txt`. You can replace the “4” with any number. To use as many threads as are available on your system, launch the application as
`bin/spark-submit --master ’local[*]’ path/to/myapp.py path/to/pg100.txt`. To learn about writing Spark applications, please read through the Spark programming guide: https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html

4 Word Count

The typical "Hello, world!" app for Spark applications is known as word count. The map/reduce model is particularly well suited to applications like counting words in a document. In this section, you will see how to develop a word count application in Python. Prior to reading this section, you should read through the Spark programming guide if you haven't already.

All operations in Spark operate on data structures called RDDs, Resilient Distributed Datasets. An RDD is nothing more than a collection of objects. If you read a file into an RDD, each line will become an object (a string, actually) in the collection that is the RDD. If you ask Spark to count the number of elements in the RDD, it will tell you how many lines are in the file. If an RDD contains only two-element tuples, the RDD is known as a "pair RDD" and offers some additional functionality. The first element of each tuple is treated as a key, and the second element as a value. Note that all RDDs are immutable, and any operations that would mutate an RDD will instead create a new RDD.

For this example, you will create your application in an editor instead of using the Spark shell. The first step of every such Spark application is to create a Spark context:

```
import re
import sys
from pyspark import SparkConf, SparkContext

conf = SparkConf()
sc = SparkContext(conf=conf)
```

Next, you'll need to read the target file into an RDD:

```
lines = sc.textFile(sys.argv[1])
```

You now have an RDD filled with strings, one per line of the file.

Next you'll want to split the lines into individual words:

```
words = lines.flatMap(lambda l: re.split(r'[^\w]+', l))
```

The flatMap() operation first converts each line into an array of words, and then makes each of the words an element in the new RDD. If you asked Spark to count the number of elements in the words RDD, it would tell you the number of words in the file.

Next, you'll want to replace each word with a tuple of that word and the number 1. The reason will become clear shortly.

```
pairs = words.map(lambda w: (w, 1))
```

The map() operation replaces each word with a tuple of that word and the number 1. The pairs RDD is a pair RDD where the word is the key, and all of the values are the number 1.

Now, to get a count of the number of instances of each word, you need only group the elements of the RDD by key (word) and add up their values:

```
counts = pairs.reduceByKey(lambda n1, n2: n1 + n2)
```

The reduceByKey() operation keeps adding elements' values together until there are no more to add for each key (word).

Finally, you can store the results in a file and stop the context:

```
counts.saveAsTextFile(sys.argv[2])
sc.stop()
```

The complete file should look like:

```
import re
import sys
from pyspark import SparkConf, SparkContext

conf = SparkConf()
sc = SparkContext(conf=conf)
lines = sc.textFile(sys.argv[1])
words = lines.flatMap(lambda l: re.split(r'[^\w]+', l))
pairs = words.map(lambda w: (w, 1))
counts = pairs.reduceByKey(lambda n1, n2: n1 + n2)
counts.saveAsTextFile(sys.argv[2])
sc.stop()
```

Save it in a file called wc.py. To run this application, do the following:

1. Download a copy of the complete works of Shakespeare from the following link: http://www.di.kaist.ac.kr/~swhang/ee412/pg100.txt

2. Open a terminal window on Mac or Linux or a command window on Windows.

3. Change into the directory where you unpacked the Spark binary.

4. Run:

   ```
   bin/spark-submit path/to/wc.py path/to/pg100.txt path/to/output
   ```

   on Mac or Linux or

   ```
   bin\spark-submit path\to\wc.py path\to\pg100.txt path\to\output
   ```

   on Windows.

After the application completes, you will find the results in the output directory you specified as the second argument to the application.

5 Write your own Spark Job

Now you will write your first Spark job to accomplish the following task:

- Write a Spark application 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 *unique* 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.
- Run your program over the same input data as above.

Please submit the following results:

- Printout of the output file.
- Your source code in one file.