Spark Tutorial

Estimated time: 4 hours

This tutorial is intended as an introduction on how to use Spark, a cluster computing framework for data analytics. We will not be examining the internals of the framework, but rather how to interface with this framework using the python API.

Submission Instructions:
Submit a single log file from running the commands in Section 3-5. If you end up doing the optional MLLIB section, submit a zip that also includes a .PDF of your solution.

Section 0: Setup and Installation

For the purpose of this tutorial, as well as all of the future projects that use Spark, we will assume that you are using the Ubuntu 14.04 64-bit operating system. If you have OS-X, you should be fine with the same instructions. However, if you are using windows, the best solution would be to run a virtual machine instance with the Ubuntu OS. Of course, you are welcome to use whatever environment you would like (and figure out how the setup works on your own) as long as the code runs as we expect it to in our environment.

Once you have the OS, you need the following packages:

- **(Java) JDK and JRE** - Information for installing Java can be found here (default should be fine). To make sure Java installed, run `java -version`.
- **Python 2.7.x** - It should be pre-installed in Ubuntu. Try running `python` from terminal to make sure it is (ctrl-d will exit you out of the python interactive shell). Alternatively, you can use Anaconda which is a python distribution that already contains many popular python packages (e.g., IPython).
- **Python developer dependencies** - `sudo apt-get install python-dev`.
- **Pip** - This is a package management tool for python software. To install, open terminal, and simply run `sudo apt-get install python-pip`.
- **IPython** - An interactive shell with a lot of functionality. To install, simply run `sudo pip install ipython` and `sudo pip install jupyter`.

Finally, you will need to download Spark, which can be obtained from http://spark.apache.org/downloads.html (or use the commands shown below). We will use the latest version of Spark, and for package type, select *Pre-built for Hadoop 2.x and later*. The following commands below are a shortcut for this process:

```
$ wget http://d3kbcqa49mib13.cloudfront.net/spark-1.6.2-bin-hadoop2.6.tgz
$ tar -xzf spark-1.6.2-bin-hadoop2.6.tgz
$ cd spark-1.6.2-bin-hadoop2.6

# Run Hello world example to calculate value of Pi.
$ ./bin/run-example SparkPi 10
```

If everything worked correctly, your spark engine should start and shut down, and you should see the message *Pi is roughly 3.144176*. For convenience, you should also set your `SPARK_HOME` environment variable to the location of the Spark directory. To do this, add `export SPARK_HOME=$HOME/spark-1.6.2-bin-hadoop2.6` to the `$HOME/.bashrc` file. Don't forget that you need to either run `source $HOME/.bashrc` or restart the terminal to make this environment variable to come into effect.

Section 1: Introduction to Spark

First, before we begin coding with PySpark, you should read the overview of Spark given in the first chapter in the book *Learning Spark: Lightning-fast Data Analytics*. The book in its entirety is available online at the NCSU library: http://catalog.lib.ncsu.edu/record/NCSU3500562. In addition, the first chapter is available in a free sampler, which can be found here.

Section 2: Getting Started with PySpark

Reference Site: https://spark.apache.org/docs/0.9.0/python-programming-guide.html

PySpark is the Python API that exposes the Spark programming model and, just like with regular python, you can program PySpark interactively via a shell or you write code in a file which will then get interpreted in its entirety. For now, we will use a basic interactive shell. To initiate the shell, you can do:

```
$ $SPARK_HOME/bin/pyspark
```

Once the shell is up you can start running Python/PySpark commands.

```
print("Hello world")
a = sc.parallelize([1, 2, 3, 4])
a.collect()
```

To exit the shell press `ctrl-d`. For this tutorial, you will be required to turn in the history of commands. To keep track of the history, you will have to launch PySpark in IPython. This can be accomplished in by setting the `IPYTHON` variable to 1 when running the PySpark command.

```
$ IPYTHON=1 $SPARK_HOME/bin/pyspark
```

Once the shell has started, run `%logstart <history_file> append` where `<history_file>` is the name of the file that you want to store history to. Once you are finish with the `%logstop` to stop logging. Note, your history file does not need to match exactly the

commands that we are going over and you may try exploring other commands during the process as well.

Section 3: Resilient Distributed Dataset (RDD)

Reference Site: http://spark.apache.org/docs/1.6.2/api/python/pyspark.html

An RDD stands for Resilient Distributed Dataset. It is simply a distributed collection of elements. In Spark, most work is expressed as either creating new RDDs, transforming existing RDDs, or calling actions on RDDs. Under the hood, Spark automatically distributes the data contained in RDDs across your cluster and parallelizes the operations you perform on them.

Section 3a: Creating RDDs

RDDs can be created by passing simple collections found in the language (e.g., lists in Python or arraylists in Java). RDDs also can be created directly from text files using a built-in function.

```
rdd_1 = sc.parallelize([1, 2, 3, 4, 5, 6, 7])
rdd_2 = sc.textFile('data/mllib/pic_data.txt')
```

The `sc` stands for "Spark context." It represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster. For now, we can just treat it as a requirement that is already satisfied within the interactive shell. If you were to create a standalone program, you would need to define a Spark context.

Section 3b: Actions and Transformations

Reference Site: http://spark.apache.org/docs/1.6.2/programming-guide.html

There are two types of operations that can be performed on RDDs: actions and transformations. Actions return a value to the drive program based on some computation performed on the RDD, while transformations create a new RDD from the existing one. A good list and description of these can be found at the above reference sites and we will practice with a few below. It is recommended that you read through the material at the reference site.

collect() - returns a list of all elements in the RDD

```
rdd = sc.parallelize([1, 2, 3])
rdd.collect()
```

count(), first(), and take() - returns the number of elements, the first element, and the size, respectively.

```
rdd.count()             # Returns the number of entries in RDD
rdd.first()             # Returns the first element of RDD
rdd.take(5)             # Returns the first 5 elements and return them as python list
```

reduce() - Aggregates the elements of this RDD using the specified commutative and associative binary operator.

```
def add(x, y):
    return x + y

def multiply(x, y):
    return x * y

rdd.reduce(add)
rdd.reduce(multiply)
```

As previously stated, **transformations** create a new RDD from some existing RDD based on some desired goal. Examples of some transformations include:

- Squaring all the elements
- Removing any negative or null values
- Splitting each line in the RDD into component words or values
- Filtering the error lines from log files

map() - Probably the most frequent transformation you will be using.

```
def square(x):
    return x**2

rdd_squared = rdd.map(square)
rdd_squared.collect()

# For such small functions, we can use lambda expressions.
# Just for a refresher, lambda is a shorthand notation for creating small functions on fly.
# The syntax is::  lambda <parameter>: <expression>

rdd_squared = rdd.map(lambda x: x**2)
```

filter() - Create a new rdd by filtering out the entries based on certain constraints.

```
def positive(x):
    if x> 0:
        return True
    return False

rdd = sc.parallelize([1, 2, -3, 4, -5, 6, 7])
rdd_pos = rdd.filter(positive)
rdd_pos.collect()

# Let us try to use lambda again for shortening the expression:
rdd.filter(lambda x: x > 0).collect()
```

groupByKey() - Return an RDD of grouped items.

```
rdd = sc.parallelize([("a", 1), ("a", 2), ("b", 1) ])
rdd_grouped = rdd.groupByKey()
rdd_grouped.map(lambda x:(x[0], list(x[1]))).collect() # Returns [('a', [1, 2]), ('b', [1])]
```

flatMap() - Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

```
rdd = sc.parallelize([2, 3, 4])

rdd.map(lambda x: range(1,x)).collect()         # Returns [[1], [1, 2], [1, 2, 3]]
rdd.flatMap(lambda x: range(1,x)).collect()     # Returns [1, 1, 2, 1, 2, 3]
```

Section 3c: Example Word Count

```
text = sc.parallelize(["I stepped on a Corn Flake, now I am a Cereal Killer", \
              "Hello world inside hello world", "Banana error", \
              "Not Lorem Ipsum", "what a wonderful day", "what up?"])
counts = text.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a,b: a+b)
counts.collect()
```

This word count solution is a good example of combining some of the aspects that you have been covered in the tutorial thus far. Make sure that you understand what is happening in this example. If you have trouble comprehending the pipe as a whole, try running just one function at a time and printing the results after each step.

Section 4: DataFrames

Reference Site: http://spark.apache.org/docs/1.6.2/sql-programming-guide.html

DataFrames are a new distributed data collection introduced in Spark 1.3 that are much more efficient than RDDs because they let Spark control the schema. As a result, the inefficient java serialization to disk can be avoided because Spark understands the schema in the DataFrame. You can read more about the differences between RDDs and DataFrames (and even a newer "Dataset") here.

Go through the following exercises:

- Starting Point: SQLContext
- Creating DataFrames
- DataFrame Operations
- Running SQL Queries Programmatically

After you have read these sections and run the commands in your shell, run the command `%logstop` to stop logging future commands and exit out of the shell

Section 5: GraphFrames

Reference Site: http://graphframes.github.io/index.html

GraphFrames is a graph processing library built on top of Spark. GraphFrames, which uses the DataFrames API, is essentially the replacement for GraphX, which uses RDDs. The two libraries are pretty similar from a user perspective, although GraphFrames has more features, such as being able to handle queries as well as motif finding.

Installation and running instructions can be found on the referenced site, but we will step through those here as well. To install, simply download the jar for your Spark version here.

```
$ wget
http://dl.bintray.com/spark-packages/maven/graphframes/graphframes/0.1.0-spark1.6/graphframes-0.1.0-spark1.6.jar
```

It is probably best if you just keep the jar in your $SPARK_HOME directory since you will need to include it when you boot up Spark:

```
$ IPYTHON=1 $SPARK_HOME/bin/pyspark --py-files graphframes-0.1.0-spark1.6.jar --jars graphframes-0.1.0-spark1.6.jar
```

Once you have installed it, go through this tutorial up to, but not including, the GraphX conversions. Make sure you start logging and append to your already created log file. Note: I had to use `from graphframes import *` before I could call the GraphFrame constructor.

Section 6: Machine Learning Library (OPTIONAL)

Reference Site: http://spark.apache.org/docs/1.6.2/mllib-guide.html

Spark has a few built-in libraries for different utilities; these are listed under `Programming Guides` at the top of the reference site web page. One of these libraries, is the Machine Learning Library, which can be explored more in-depth through the provided (as well as your projects). The other libraries may be of interest to you as well, so feel free to explore further.

In this part of the tutorial, we will also be introducing IPython notebooks. IPython notebooks can be thought of a more advanced interactive shell where you can intermix regular text, code snippets, and the results from that code. This makes it a really powerful tool for interactively exploring the data, trying different data mining techniques, and taking notes.

To run PySpark in an IPython notebook, run the following command:

```
$ IPYTHON_OPTS="notebook" $SPARK_HOME/bin/pyspark
```

A new window should be opened in your browser. From here you can open an existing IPython notebook or create your own notebook. Open the IPython notebook included with this tutorial (the .ipynb file), there will be instructions for completing the rest of the tutorial there.