

Exercise 6a

More Apache Spark and Python

Prior Knowledge

Unix Command Line Shell

Simple Python

Learning Objectives

Using Spark on EC2

Accessing S3 files on Spark

Reading CSV files in Spark

Seeing the differences between Spark and Hadoop by performing the Wind Analysis in Spark

Spark SQL

Software Requirements

(see separate document for installation of these)

- EC2 credentials
- Git

Part A. Starting Spark in EC2

1. Do you remember the Access Key and Secret Key from Exercise 1? You need those now.

2. In a terminal window type:

```
export AWS_ACCESS_KEY_ID=<your access key here>
export AWS_SECRET_ACCESS_KEY=<your secret key here>
```

3. In Spark before 2.0.0, the EC2 scripts came bundled with Spark itself. These are now separate, so we will install them.

```
cd ~
git clone https://github.com/amplab/spark-ec2.git
```

4. Now change into the Spark EC2 directory:

```
cd ~/spark-ec2
```

5. Now let's launch a Spark cluster in EC2. Replace XX with your user details so these match the locations of your key files and so you can identify your own spark cluster. All one line:

```
./spark-ec2 -k oxclo20 -i /home/oxclo/oxclo20.pem  
--region eu-west-1 --hadoop-major-version 2  
-s 1 launch oxclo20-spark-cluster
```

The -s 1 indicates that there is just one slave (you could launch more but that might be expensive).

The default instance type is m3.large, which costs US\$0.133/hour and each cluster is at least two servers. So it comes down to how long you are running. If you can run a massive job on twenty servers and complete in one hour, then \$3 is going to be cheap. If you leave just 3 servers running for a week, doing not much, that is going to waste \$67. However, in my opinion this is all cheaper than using Amazon's own EMR facility, which shuts down the servers as soon as the job is over and then you pay for an hour even if the job failed instantly!

Hint:

If you have a key problem at this stage it might be to do with the time on your Ubuntu VM

You should see output like:

```
Setting up security groups...  
Searching for existing cluster my-spark-cluster in region  
eu-west-1...  
Spark AMI: ami-1ae0166d  
Launching instances...  
Launched 1 slave in eu-west-1a, regid = r-52c4f5ff  
Launched master in eu-west-1a, regid = r-c2c7f66f  
Waiting for AWS to propagate instance metadata...  
Waiting for cluster to enter 'ssh-ready' state.....  
  
Warning: SSH connection error. (This could be temporary.)  
Host: ec2-52-16-96-164.eu-west-1.compute.amazonaws.com  
SSH return code: 255  
SSH output: ssh: connect to host ec2-52-16-96-164.eu-  
west-1.compute.amazonaws.com port 22: Connection refused
```

You may also see:

```
Warning: SSH connection error. (This could be temporary.)  
Host: ec2-54-171-175-114.eu-west-1.compute.amazonaws.com  
SSH return code: 255  
SSH output: ssh: connect to host ec2-54-171-175-114.eu-west-  
1.compute.amazonaws.com port 22: Connection refused
```

Don't worry about this either.

6. Maybe go grab a coffee ☺ This takes a while
7. After a while the system will start logging a lot more as the setup on EC2 starts happening.

Eventually you will see:

```
Setting up ganglia  
RSYNC'ing /etc/ganglia to slaves...  
ec2-52-31-197-16.eu-west-1.compute.amazonaws.com  
Shutting down GANGLIA gmond: [FAILED]  
Starting GANGLIA gmond: [OK]  
Shutting down GANGLIA gmond: [FAILED]  
Starting GANGLIA gmond: [OK]  
Connection to ec2-52-31-197-16.eu-west-1.compute.amazonaws.com closed.  
Shutting down GANGLIA gmetad: [FAILED]  
Starting GANGLIA gmetad: [OK]  
Stopping httpd: [FAILED]  
Starting httpd: httpd: Syntax error on line 154 of /etc/httpd/conf/httpd.conf: Cannot load  
/etc/httpd/modules/mod_authz_core.so into server: /etc/httpd/modules/mod_authz_core.so: cannot open  
shared object file: No such file or directory [FAILED]  
[timing] ganglia setup: 00h 00m 02s  
Connection to ec2-52-16-96-164.eu-west-1.compute.amazonaws.com closed.  
Spark standalone cluster started at http://ec2-52-16-96-164.eu-west-1.compute.amazonaws.com:8080  
Ganglia started at http://ec2-52-16-96-164.eu-west-1.compute.amazonaws.com:5080/ganglia  
Done!
```

8. Ignore the Ganglia errors.
9. Find the Spark URL in the log above and go to that page in your browser.
You might want to leave this open as we'll need it later.

10. Let's login to the master:

```
./spark-ec2 -k oxcloXX -i /home/oxclo/oxcloXX.pem \  
--region eu-west-1 \  
login oxcloXX-spark-cluster
```

You see something like:

```
Last login: Wed Nov 18 09:27:01 2015 from ip-172-31-9-125.eu-  
west-1.compute.internal
```

```
 _ _|_(_ _|_ )  
 _ | ( _|_ / Amazon Linux AMI  
 __| \__|__|
```

```
https://aws.amazon.com/amazon-linux-ami/2013.03-release-notes/  
Amazon Linux version 2015.09 is available.
```

11. This basically just SSH's you into the master. You could do the same from the EC2 console as before.



12. We are going to need your AWS credentials in this session as well so that spark can access S3 resources. In this session type

expo

Type

14. Now start pyspark once again. This time we are going to add in a Spark Package that supports easy reading of CSV files (one line):

bin/pyspark
 packages com.databricks:spark-csv_2.11:1.5.0

You should see:

Note that this is running an older version of Spark to the one deployed on the VM, but everything is very similar.

15. We are going to use Spark's SQL support which in turn uses Apache Hive. This combined with the CSV package we saw earlier makes it very easy to work with data.

First let's tell spark we are using SQL:

```
from pyspark.sql import SQLContext  
  
sqlContext = SQLContext(sc)
```



16. Now let's load the data into a DataFrame. (one line)

```
df = sqlContext.  
read.format('com.databricks.spark.csv').  
options(header='true', inferSchema='true').  
load('s3n://oxclo-wind/2015/*')
```

17. You should see a lot of log go by.

18. The df object we have is not an RDD. It is basically a SQL construct. But we can easily convert it into an RDD.

```
winds = df.rdd
```

19. You can see the structure of each row by:

```
winds.first()
```

20. You will see something like:

```
oxclo@oxclo: ~/spark-ec2  
16/09/08 11:29:11 INFO MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.4 KB, free 586.9 KB)  
16/09/08 11:29:11 INFO BlockManagerInfo: Added broadcast_9_piece0 in memory on 172.31.39.235:46696 (size: 5.4 KB, free: 511.4 MB)  
16/09/08 11:29:11 INFO SparkContext: Created broadcast 9 from broadcast at DAGScheduler.scala:1006  
16/09/08 11:29:11 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 4 (PythonRDD[17] at RDD at PythonRDD.scala:43)  
16/09/08 11:29:11 INFO TaskSchedulerImpl: Adding task set 4.0 with 1 tasks  
16/09/08 11:29:11 INFO TaskSetManager: Starting task 0.0 in stage 4.0 (TID 14, ip-172-31-43-152.eu-west-1.compute.internal, partition 0,PROCESS_LOCAL, 2627 bytes)  
16/09/08 11:29:11 INFO BlockManagerInfo: Added broadcast_9_piece0 in memory on ip-172-31-43-152.eu-west-1.compute.internal:33263 (size: 5.4 KB, free: 3.9 GB)  
16/09/08 11:29:11 INFO BlockManagerInfo: Added broadcast_8_piece0 in memory on ip-172-31-43-152.eu-west-1.compute.internal:33263 (size: 9.0 KB, free: 3.9 GB)  
16/09/08 11:29:11 INFO TaskSetManager: Finished task 0.0 in stage 4.0 (TID 14) in 530 ms on ip-172-31-43-152.eu-west-1.compute.internal (1/1)  
16/09/08 11:29:11 INFO TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool  
16/09/08 11:29:11 INFO DAGScheduler: ResultStage 4 (runJob at PythonRDD.scala:393) finished in 0.532 s  
16/09/08 11:29:11 INFO DAGScheduler: Job 4 finished: runJob at PythonRDD.scala:393, took 0.553450 s  
Row(Station_ID=u'SF04', Station_Name=u'Lincoln High School', Location_Label=u'2162 24th Ave', Interval_Minutes=5, Interval_End_Time=u'2015-01-5 07:50', Wind_Velocity_Mtr_Sec=0.979, Wind_Direction_Variance_Deg=0.31, Wind_Direction_Deg=57.69, Ambient_Temperature_Deg_C=6.297, Global_Horizontal_Irradiance=0.706)  
>>>
```

21. Let's do the normal step of mapping the data into a simple <K,V> pair. Each column in the row can be accessed by the syntax e.g. row.Station_ID

We can therefore map our RDD with the following: (one line)

```
mapped = winds.map(lambda s: (s.Station_ID,  
s.Wind_Velocity_Mtr_Sec))
```

22. We can simply calculate the maximum values with this reducer:

```
maxes = mapped.reduceByKey(lambda a, b: a if (a>b) else b)
```

23. And once again collect / print:

```
for (k,v) in maxes.collect(): print k,v
```

24. You can also turn the response of a collect into a Python Map, which is handy. Try this:

```
maxes.collectAsMap()['SF04']
```

25. You will see a bunch of log before the following appears:

```
SF18 10.57
SF36 11.05
SF37 7.079
SF15 7.92
SF04 34.12
SF17 5.767
```

PART B – Using SQL

26. There is an easier way to do all this if you are willing to write some SQL.

27. First we need to give our DataFrame a table name:
`df.registerTempTable('wind')`

28. Now we can use a simple SQL statement against our data.
ALL ON ONE Line type:

```
sqlContext.sql("SELECT Station_ID, avg(Wind_Velocity_Mtr_Sec) as
avg,max(Wind_Velocity_Mtr_Sec) as max from wind group by
Station_ID").show()
```

29. Bingo you should see a lot of log followed by:

Station_ID	avg	max
SF36	2.464172530911313	11.05
SF37	2.260403505500663	7.079
SF04	2.300981748124102	34.12
SF15	1.8214145677504483	7.92
SF17	0.5183500253485376	5.767
SF18	2.2202234391695437	10.57

30. Recap. We have:

- a. Started Spark in EC2
- b. Loaded data from S3
- c. Used SQL to read in CSV files
- d. Explored Map/Reduce on those CSV files
- e. Used SQL to query the data.

31. Go back to the browser view of the Spark console and you can take a look at the jobs that have been run:

The screenshot shows the Spark Master interface at `spark://ec2-52-16-96-164.eu-west-1.compute.amazonaws.com:7077`. The page includes the following sections:

- Alive Workers:** 1 worker listed.
- Memory in use:** 6.3 GB Total, 0.0 B Used.
- Applications:** 0 Running, 5 Completed.
- Drivers:** 0 Running, 0 Completed.
- Status:** ALIVE.
- Workers:** A table showing one worker with ID `worker-20151118093054-172.31.10.244-56180`, Address `172.31.10.244:56180`, State `ALIVE`, Cores `2 (0 Used)`, and Memory `6.3 GB (0.0 B Used)`.
- Running Applications:** An empty table.
- Completed Applications:** A table listing five completed PySparkShell applications, each with Name, Cores, Memory per Node, Submitted Time, User, State, and Duration.

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20151118095430-0004	PySparkShell	2	6.0 GB	2015/11/18 09:54:30	root	FINISHED	39 min
app-20151118094604-0003	PySparkShell	2	6.0 GB	2015/11/18 09:46:04	root	FINISHED	5.0 min
app-20151118094533-0002	PySparkShell	2	6.0 GB	2015/11/18 09:45:33	root	FINISHED	4 s
app-20151118094206-0001	PySparkShell	2	6.0 GB	2015/11/18 09:42:06	root	FINISHED	2.1 min
app-20151118093751-0000	PySparkShell	2	6.0 GB	2015/11/18 09:37:51	root	FINISHED	1.0 min

32. Quit the pyspark shell:
`quit()`

33. Exit the SSH session:
`exit`

34. We must remember to stop our cluster as well (its costing money!)
From Ubuntu terminal where you started the Spark cluster

```
./spark-ec2 --region eu-west-1 destroy oxcloXX-spark-cluster
```

Type y when prompted.

35. Congratulations!