Exercise 7a

Get started with Cassandra and import data

Prior Knowledge
Unix Command Line Shell
HDFS
Simple Python
Spark Python
Simple SQL syntax

Learning Objectives
Understand Cassandra's CQL shell
Integrate Python, Cassandra and Spark
Load data from CSV into Cassandra using Spark Python

Software Requirements
(see separate document for installation of these)

- Apache Spark 1.5.1
- Python 2.7.x
- Apache Cassandra 2.1.11
- Nano text editor or other text editor

Part A

1. Make sure Cassandra is running
    a. In a Terminal window (Crtl-Alt-T) type:
    service cassandra status
    b. You should see
    * Cassandra is running
    c. If not, try
    sudo service cassandra start
    and then check the status again
2. Now you can ask Cassandra about its own situation:
    nodetool status

    You should see something like:

```
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens       Owns    Host ID
Rack
UN  127.0.0.1  668.72 MB  256          ?       56a82160-ee44-47aa-810a-
d8b56b751a92  rack1

Note: Non-system keyspaces don't have the same replication settings, effective
ownership information is meaningless
```

3. You can also try:
   nodetool info
   You should see something like:

```
ID                     : 56a82160-ee44-47aa-810a-d8b56b751a92
Gossip active          : true
Thrift active          : false
Native Transport active: true
Load                   : 668.72 MB
Generation No          : 1447268202
Uptime (seconds)       : 144
Heap Memory (MB)       : 424.54 / 1984.00
Off Heap Memory (MB)   : 3.08
Data Center            : datacenter1
Rack                   : rack1
Exceptions             : 0
Key Cache              : entries 23, size 1.87 KB, capacity 99 MB, 26 hits, 54
requests, 0.481 recent hit rate, 14400 save period in seconds
Row Cache              : entries 0, size 0 bytes, capacity 0 bytes, 0 hits, 0
requests, NaN recent hit rate, 0 save period in seconds
Counter Cache          : entries 0, size 0 bytes, capacity 49 MB, 0 hits, 0
requests, NaN recent hit rate, 7200 save period in seconds
Token                  : (invoke with -T/--tokens to see all 256 tokens)
```

4. Now you can start the Cassandra Shell:
   Type:
   `cqlsh`

   You should see:

```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.2.3 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
cqlsh>
```

5. Let's create a new database (Keyspace):
   a. Type (all on a single line)

```
CREATE KEYSPACE TEST WITH REPLICATION = { 'class' :
'SimpleStrategy', 'replication_factor' : 1 };
```

   b. Check it worked:
      Type:

```
desc keyspace test;
```

   c. You should see:

```
CREATE KEYSPACE test WITH replication = {'class':
'SimpleStrategy', 'replication_factor': '1'}  AND
durable_writes = true;
```

6. Now we need to select to use that keyspace:
   `use test;`

7. The command prompt will change to:
   cqlsh:test>

8. Let's create a simple (key, value) table

a. Type:
```
create table kv ( key text, value text, primary key (key));
```

b. Now type
```
desc kv;
```

c. You should see:
```
cqlsh:test> desc kv;

CREATE TABLE test.kv (
    key text PRIMARY KEY,
    value text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL",
"rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompactionStra
tegy'}
    AND compression = {'sstable_compression':
'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

d. Add some simple values:
```
insert into kv (key, value) values ('a','1');
insert into kv (key, value) values ('b','2');
insert into kv (key, value) values ('c','3');
```

e. Now type:
```
select * from kv;
```

You should see:
```
key | value
-----+-------
   a |     1
   c |     3
   b |     2

(3 rows)
```

9. You can also do other simple SQL of course
```
cqlsh:test> select * from kv where key='a' ;

 key | value
-----+-------
   a |     1

(1 rows)
```


10. Now exit the cqlsh:
    `exit`

11. Congratulations! You have Cassandra running and working.

PART B – Stress testing Cassandra

12. Now let's run a performance test on Cassandra.
    a. We will use the cassandra-stress tool which is part of the Cassandra distribution.

    b. First we need to write some data into Cassandra using the tool
    c. `cassandra-stress write n=100000`

    d. You should see:

```
INFO  19:27:23 Did not find Netty's native epoll transport in the classpath, defaulting to
NIO.
INFO  19:27:23 Using data-center name 'datacenter1' for DCAwareRoundRobinPolicy (if this
is incorrect, please provide the correct datacenter name with DCAwareRoundRobinPolicy
constructor)
INFO  19:27:23 New Cassandra host localhost/127.0.0.1:9042 added
Connected to cluster: Test Cluster
Datatacenter: datacenter1; Host: localhost/127.0.0.1; Rack: rack1
Created keyspaces. Sleeping 1s for propagation.
Sleeping 2s...
Warming up WRITE with 50000 iterations...
Running WRITE with 200 threads for 100000 iteration
type,      total ops,    op/s,    pk/s,   row/s,    mean,     med,     .95,     .99,
.999,     max,   time,   stderr, errors,  gc: #,  max ms,  sum ms,  sdv ms,      mb
total,         15657,   15820,   15820,   15820,    15.0,    11.7,    38.6,    93.7,
129.9,   135.5,    1.0,  0.00000,      0,       0,       0,       0,       0,       0
total,         32473,   14669,   14669,   14669,    13.5,     9.4,    35.8,    96.0,
288.7,   316.8,    5.7,  0.04323,      0,       0,       0,       0,       0,       0
total,         98307,   16813,   16813,   16813,    11.8,     7.1,    33.6,   125.0,
164.2,   172.2,    6.9,  0.05918,      0,       1,     115,     115,       0,     293
total,        100000,   15868,   15868,   15868,    11.5,     8.0,    31.6,    86.9,
88.9,    89.9,    7.0,  0.05107,      0,       0,       0,       0,       0,       0


Results:
op rate                   : 14256 [WRITE:14256]
partition rate            : 14256 [WRITE:14256]
row rate                  : 14256 [WRITE:14256]
latency mean              : 14.2 [WRITE:14.2]
latency median            : 9.5 [WRITE:9.5]
latency 95th percentile   : 36.4 [WRITE:36.4]
latency 99th percentile   : 107.0 [WRITE:107.0]
latency 99.9th percentile : 254.2 [WRITE:254.2]
latency max               : 316.8 [WRITE:316.8]
Total partitions          : 100000 [WRITE:100000]
Total errors              : 0 [WRITE:0]
total gc count            : 2
total gc mb               : 592
total gc time (s)         : 0
avg gc time(ms)           : 102
stdev gc time(ms)         : 13
Total operation time      : 00:00:07
END
```

13. Now you can try a full test:
    `cassandra-stress mixed n=100000`

14. At what thread count did you get the highest throughput? And the lowest latency?

PART C – Loading data from CSV files into Cassandra

15. Firstly, we need to create a database and a table in which to store our data. Start up the **cqlsh** again and type the following commands:

```
CREATE KEYSPACE wind
WITH replication = {'class': 'SimpleStrategy',
'replication_factor': '1'};

USE wind;

CREATE TABLE winddata (
          stationid text,
          time timestamp,
          direction float,
          temp float,
          velocity float,
          PRIMARY KEY (stationid, time)
      );
```

16. In order to load the CSV files into Cassandra, we are going to use two Spark packages to help us. The first is the Databricks Spark CSV reader, and the second is the Cassandra plugin for Pyspark.

    a. To use these, we need to start Pyspark with the correct command line. Start a terminal window and change into the Spark directory:

    ```
    cd spark-1.5.1
    ```

    b. Now type the following:

    ```
    bin/pyspark --packages \
    TargetHolding:pyspark-cassandra:0.1.5,\
    com.databricks:spark-csv_2.11:1.2.0
    ```

    c. You should see an inordinate amount of log before you see:

    ```
    15/11/12 14:53:48 INFO BlockManagerMaster: Registered BlockManager
    Welcome to
          ____              __
         / __/__  ___ _____/ /__
        _\ \/ _ \/ _ `/ __/  '_/
       /__ / .__/\_,_/_/ /_/\_\   version 1.5.1
          /_/

    Using Python version 2.7.6 (default, Jun 22 2015 17:58:13)
    SparkContext available as sc, SQLContext available as sqlContext.
    ```

17. Now we need to set up our imports:
    In the shell type (or cut and paste from http://freo.me/oxclo-spark-cass)

```
from pyspark_cassandra import CassandraSparkContext
from pyspark import SparkContext, SparkConf
import time
from datetime import datetime
from pyspark.sql import SQLContext
```

18. Next we need to stop the existing SparkContext. We want to create a special Cassandra-aware context and you can't have two running at the same time:

```
sc.stop()
```

19. Next we need to initialize a CassandraSparkContext pointing to our local Cassandra:

```
conf = SparkConf() \
    .setAppName("PySpark Cassandra Test")\
    .setMaster("local")\
    .set("spark.cassandra.connection.host", "127.0.0.1")

sc = CassandraSparkContext(conf=conf)

sqlContext = SQLContext(sc)
```

20. Before the next step make sure that HDFS is running!
    Remember you can test it with:
    `hadoop fs -ls -R /`
    and start it with
    `start-dfs.sh`

21. Now lets load the CSV files into a SQL Dataframe:

```
df = sqlContext.read.format('com.databricks.spark.csv').\
options(header='true', inferschema='true').\
load('hdfs://localhost:54310/user/oxclo/wind/*')
```

22. Take a look at the data in df:

```
df.first()
```

    After the log, you should see something like:

```
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=40.31, Wind_Direction_Deg=57.69,
Ambient_Temperature_Deg_C=6.297,
Global_Horizontal_Irradiance=0.706)
```

23. We have two challenges. Firstly we want to map the Interval_End_Time into something we can put in Cassandra. Cassandra expects a Python datetime.datetime object.

    This chunk of python will convert the string date/time into that:

```
convertTime = lambda t: \
datetime.fromtimestamp( \
time.mktime(time.strptime(t, "%Y-%m-%d? %H:%M")))
```


24. Secondly, we need to create a Python dictionary with the right names for our Cassandra Table. This function does that. I recommend you cut and paste!

```
toDict = lambda s: \
dict(stationid=s.Station_ID, \
time=convertTime(s.Interval_End_Time), \
direction=s.Wind_Direction_Deg, \
temp=s.Ambient_Temperature_Deg_C, \
velocity=s.Wind_Velocity_Mtr_Sec)
```

25. We need to map this function onto the data:

```
rdd2 = df.rdd.map(toDict)
```

26. Finally, we can do the work:

```
rdd2.saveToCassandra('wind', 'winddata')
```

This will take a bit longer!

27. Check that the data has loaded. In your **cqlsh** window type:

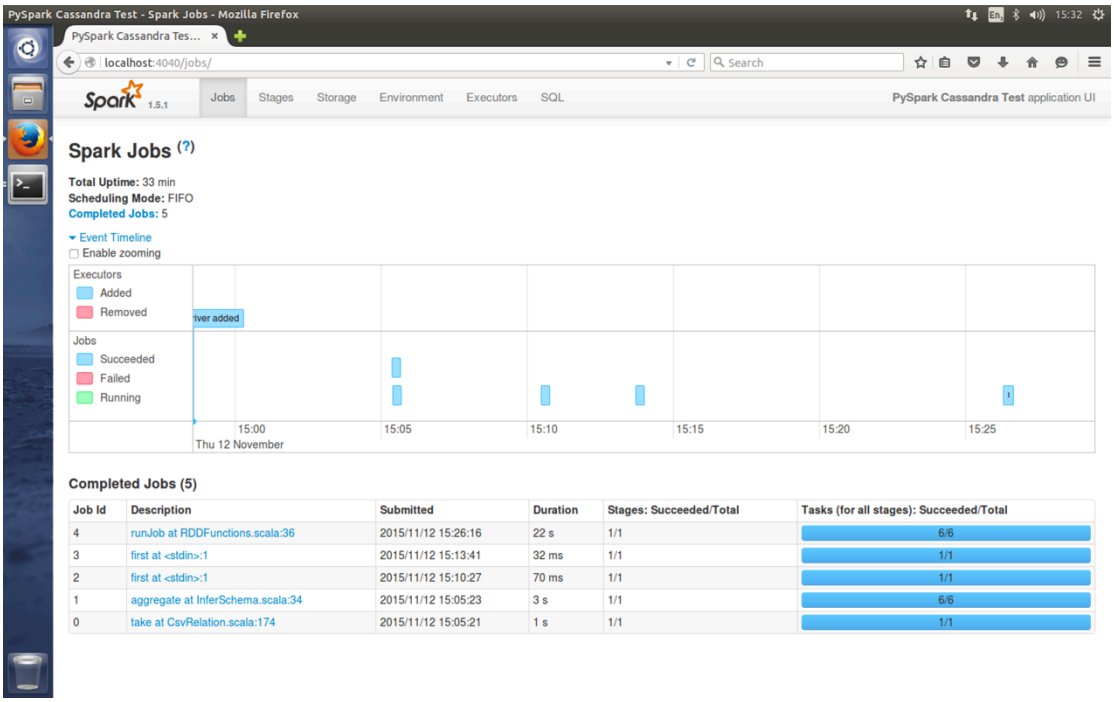
```
select * from wind.winddata limit 15;
```

28. You should see something like:

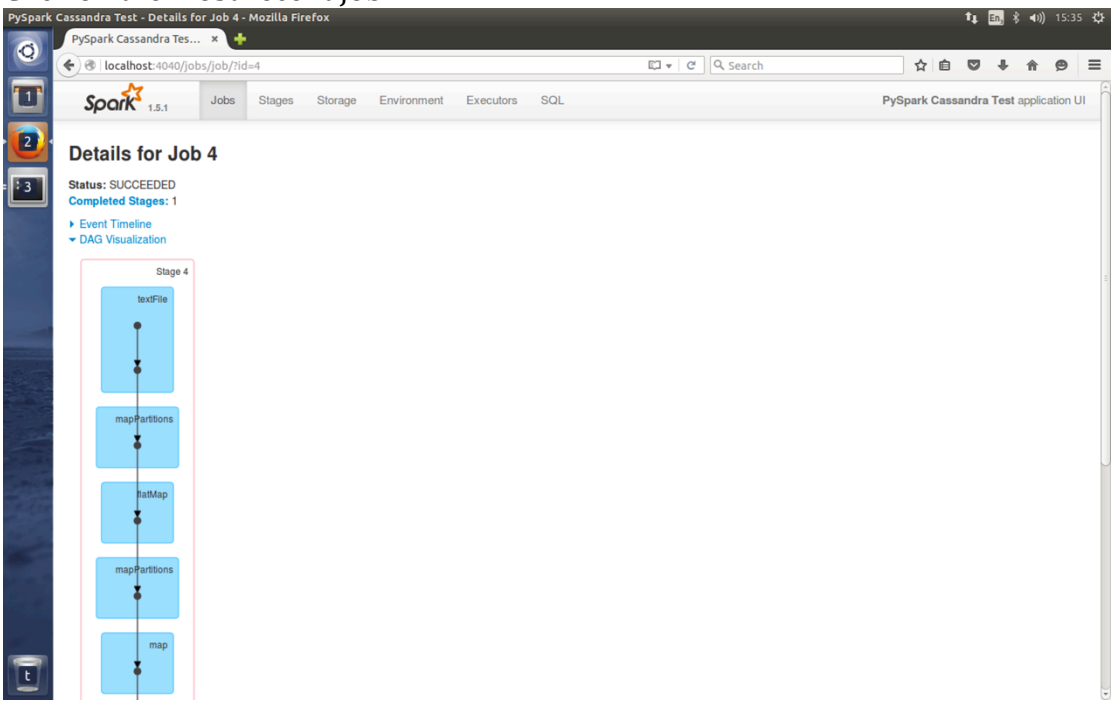
```
 stationid | time                     | direction | temp  | velocity
-----------+--------------------------+-----------+-------+----------
       SF36 | 2015-01-01 00:00:00+0000 |     116.9 | 11.33 |    2.727
       SF36 | 2015-01-01 00:05:00+0000 |     108.5 | 11.25 |    1.814
       SF36 | 2015-01-01 00:10:00+0000 |     113.7 |  11.2 |    2.621
       SF36 | 2015-01-01 00:15:00+0000 |     117.8 | 11.11 |    3.678
       SF36 | 2015-01-01 00:20:00+0000 |     117.3 | 11.07 |    2.842
       SF36 | 2015-01-01 00:25:00+0000 |     117.3 | 11.07 |    2.629
       SF36 | 2015-01-01 00:30:00+0000 |     117.3 | 11.09 |    2.235
       SF36 | 2015-01-01 00:35:00+0000 |     117.2 | 11.09 |    2.043
       SF36 | 2015-01-01 00:40:00+0000 |     117.2 | 11.05 |    1.635
       SF36 | 2015-01-01 00:45:00+0000 |     117.3 | 10.93 |    2.224
       SF36 | 2015-01-01 00:50:00+0000 |     112.5 | 10.86 |    1.822
       SF36 | 2015-01-01 00:55:00+0000 |     108.7 |  10.8 |    0.866
       SF36 | 2015-01-01 01:00:00+0000 |     108.7 | 10.67 |    1.068
       SF36 | 2015-01-01 01:05:00+0000 |     108.6 | 10.54 |    1.393
       SF36 | 2015-01-01 01:10:00+0000 |     108.7 | 10.44 |    1.468

(15 rows)
```

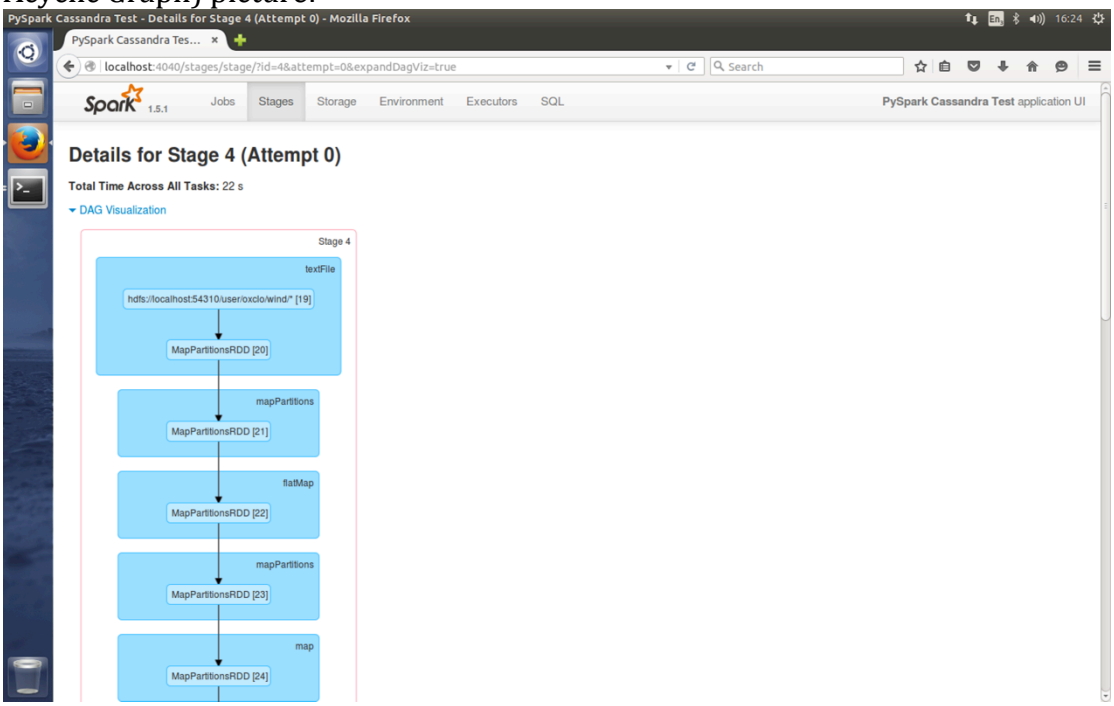

29. Browse to http://localhost:4040
    It will look similar to:



30. Click on the most recent job:



31. You can also get more details by clicking on a stage in the DAG (Directed Acyclic Graph) picture:



32. Congratulations, you have finished this lab.