Exercise 5

Get started with Apache Spark and Python

Prior Knowledge
Unix Command Line Shell
Simple Python

Learning Objectives
Understand the Spark system
Use the Spark Python shell to interactively work with data
Submit Spark jobs locally and using YARN
Write SparkSQL code in Python
WordCount!

Software Requirements
(see separate document for installation of these)

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

Part A. Spark Python Shell (pySpark)

1. We are going to do a wordcount against a set of books downloaded from Project Gutenberg. Wordcount is the definitive Big Data program (sort of Hello World for Big Data) and its frankly embarrassing that we haven't done one yet.

2. Apache Spark has a useful Python shell, which we can use to interactively test and run code. Since we have our data in HDFS, *we need to ensure HDFS is running.* (Follow the instructions from the Hadoop lab).

3. Let's load some books into HDFS. In a terminal window (Ctrl-Alt-T)

```
hadoop fs -mkdir  /user/oxclo/books
hadoop fs -put ~/datafiles/books/* /user/oxclo/books/
```

4. Now, change to the Spark directory:
```
cd ~/spark-1.5.1
```

5. Now start the Spark Python command line tool – pyspark
```
bin/pyspark
```

a. You should see a lot of log come up, ending in something like:

```
15/10/25 23:39:52 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.
```

6. Now let's load some data. We already have a SparkContext object defined in the shell (in a program you need to define one, which we will see later)

7. Unfortunately some of the input is handled as Unicode by Python and we want to get rid of that. So let's start:

```
import unicodedata
```

8. Then type (on one line):

```
def u2a(u): return str(unicodedata.normalize('NFKD',u).
encode('ascii','ignore'))
```

9. We also want to remove any non-alphanumeric characters:

```
def strip(s): return ''.join(filter(str.isalpha, s))
```

10. Now we would like to load the books from HDFS. Now let's load some data. We already have a SparkContext object defined in the shell (in a program you need to define one, which we will see later)

```
books = sc.textFile("hdfs://localhost:54310/user/oxclo/books/*")
```

11. Let's split the lines into words:

```
split = books.flatMap(lambda line: line.split())
```

12. Now let's transform from Unicode to ascii

```
asc = split.map(u2a)
```

13. And remove non-alpha characters

```
stripped = asc.map(strip)
```

14. And we should put everything to lower case while we are cleaning it up

```
lower = stripped.map(lambda a: a.lower())
```

15. Finally we are ready to do the classic "WordCount" Map Reduce. We first create a simple <K,V> pair of <word, count>. In the map phase, the count is always 1, since we haven't yet reduced this.

```
numbered = lower.map(lambda word: (word, 1))
```

16. Next we reduce by adding the counts together for the same words:

```
wordcount = numbered.reduceByKey(lambda a,b: a+b)
```

17. Finally, we need to collect the results and print them. In Spark, they may be distributed across different RDD partitions on different machines, so the collect() method brings them together.

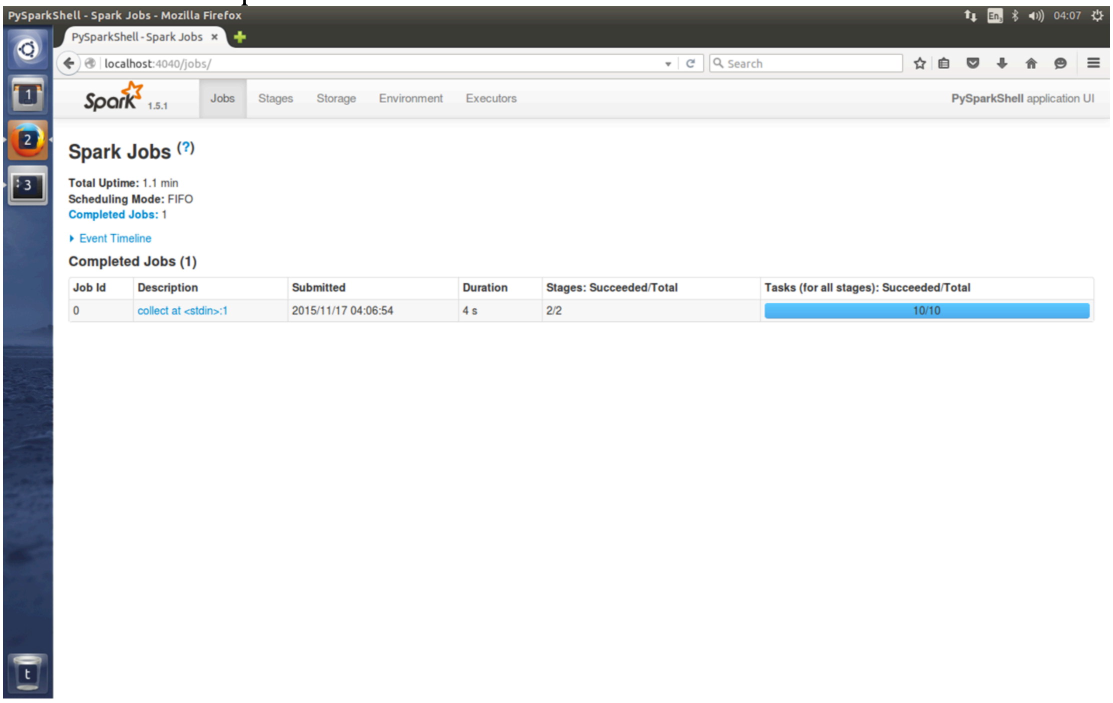
```
for k,v in wordcount.collect(): print k,v
```

18. You should see a lot of word counts go flying past.

19. Congratulations!

20. While the pyspark is still running browse to http://localhost:4040

21. You will see the Spark web console:



PySparkShell - Spark Jobs - Mozilla Firefox
04:07
PySparkShell - Spark Jobs
localhost:4040/jobs/
Search
Spark 1.5.1
Jobs
Stages
Storage
Environment
Executors
PySparkShell application UI

Spark Jobs (?)

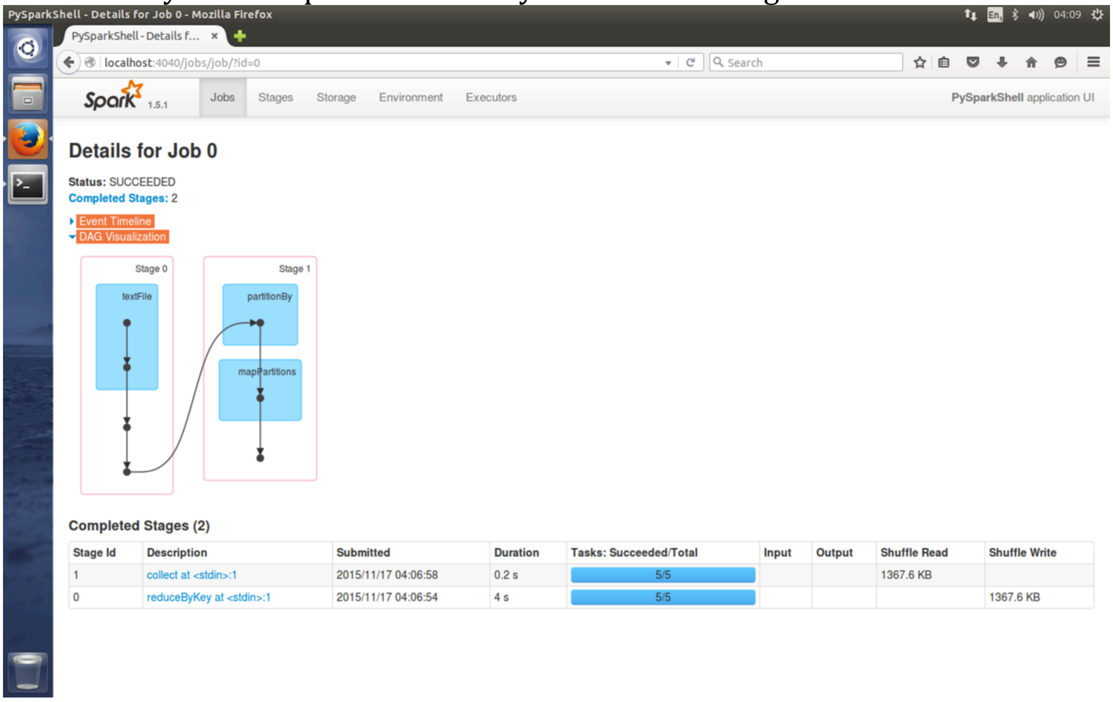
Total Uptime: 1.1 min
Scheduling Mode: FIFO
Completed Jobs: 1

▸ Event Timeline

Completed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at <stdin>:1	2015/11/17 04:06:54	4 s	2/2	10/10

22. Click on the blue link "collect at stdin"
    This shows you how Spark converted your code into stages:


PySparkShell - Details for Job 0 - Mozilla Firefox
04:09
PySparkShell - Details f...
localhost:4040/jobs/job/?id=0
Search
Spark 1.5.1
Jobs
Stages
Storage
Environment
Executors
PySparkShell application UI
Stage 0
textFile
Stage 1
partitionBy
mapPartitions

Details for Job 0

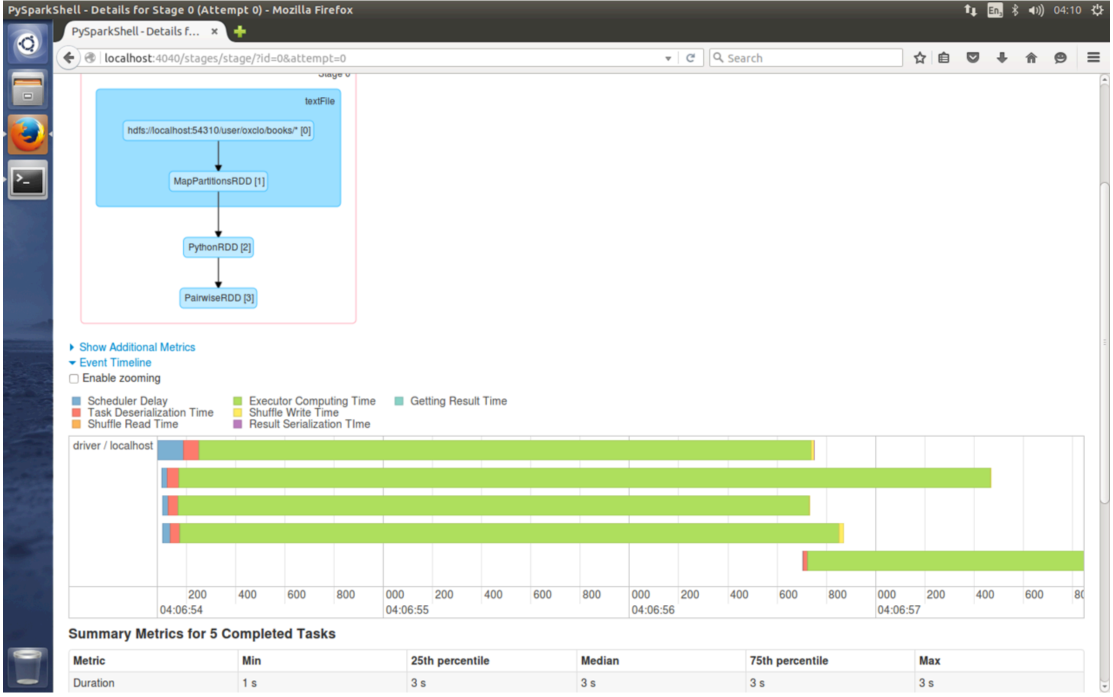
Status: SUCCEEDED
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	collect at <stdin>:1	2015/11/17 04:06:58	0.2 s	5/5			1367.6 KB	
0	reduceByKey at <stdin>:1	2015/11/17 04:06:54	4 s	5/5				1367.6 KB

23. From the stages section you can dig deeper into your execution plan:



▸ Show Additional Metrics
▾ Event Timeline
□ Enable zooming

Summary Metrics for 5 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 s	3 s	3 s	3 s	3 s

24. Quit the pyspark shell by typing `quit()`

25. Now let's run the same code as a "job" instead of interactively.

26. Make a directory for your spark python code:
```
mkdir ~/pysp
cd ~/pysp
```

27. From http://freo.me/oxclo-wc-py copy the code into a file wc.py

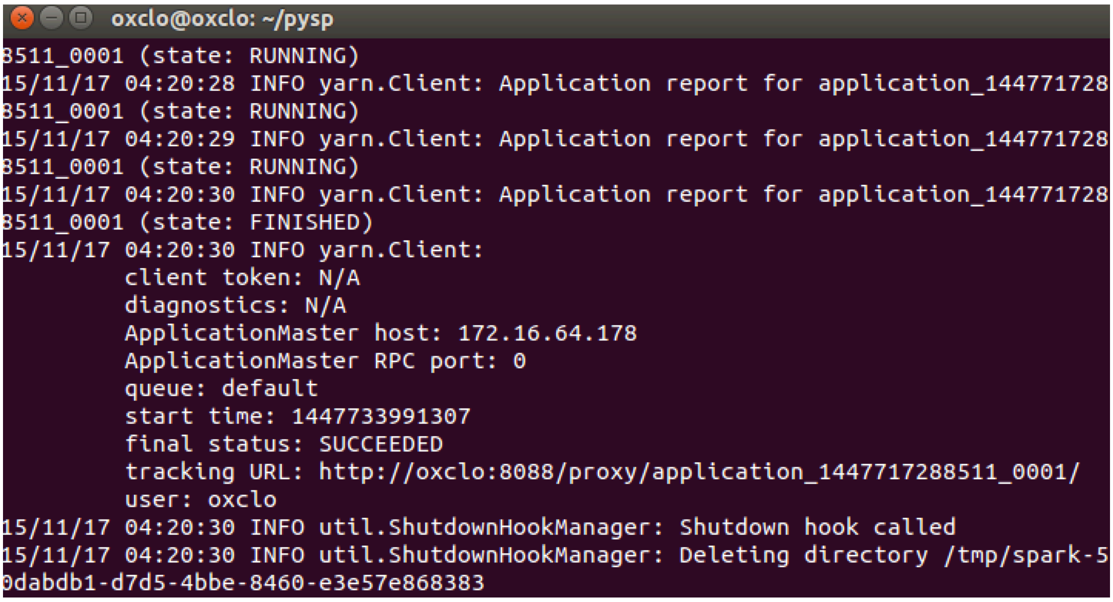
28. Now configure the correct setup so Spark can find the Yarn system:

```
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop/
```

29. Now submit the job via YARN:
```
~/spark-1.5.1/bin/spark-submit --master yarn-cluster wc.py \
"hdfs://localhost:54310/user/oxclo/books/*"
```

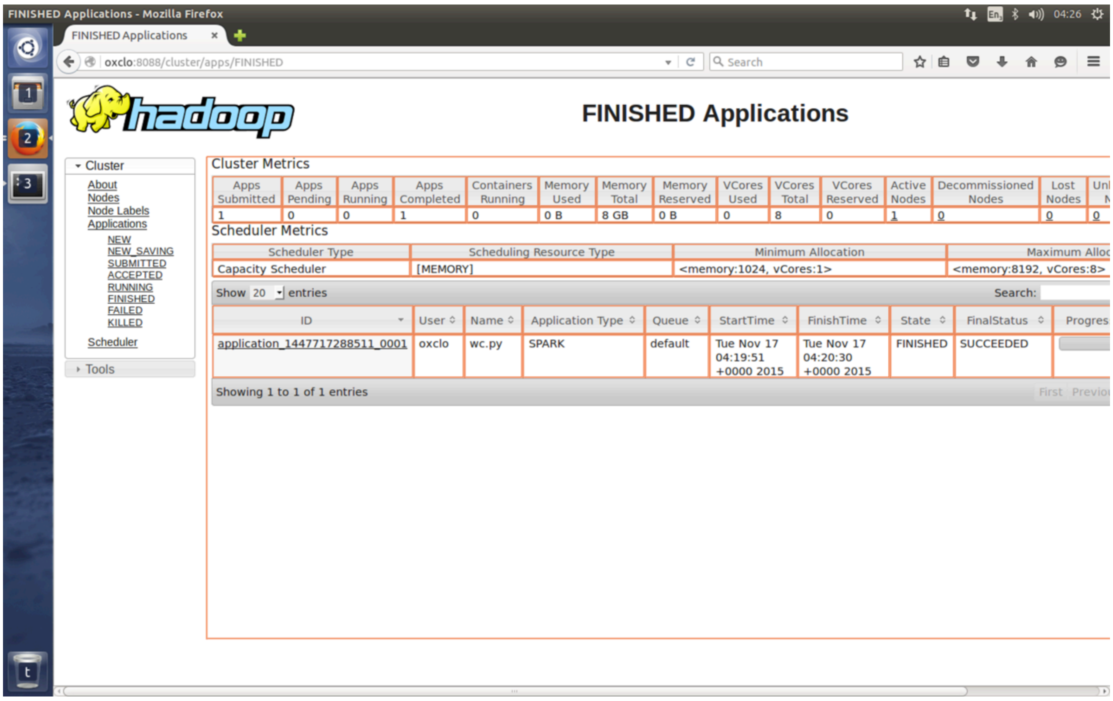
30. You can see the status of the job via the YARN webpage:
http://localhost:8088

31. You should also see a success via the command line:



```
8511_0001 (state: RUNNING)
15/11/17 04:20:28 INFO yarn.Client: Application report for application_144771728
8511_0001 (state: RUNNING)
15/11/17 04:20:29 INFO yarn.Client: Application report for application_144771728
8511_0001 (state: RUNNING)
15/11/17 04:20:30 INFO yarn.Client: Application report for application_144771728
8511_0001 (state: FINISHED)
15/11/17 04:20:30 INFO yarn.Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: 172.16.64.178
         ApplicationMaster RPC port: 0
         queue: default
         start time: 1447733991307
         final status: SUCCEEDED
         tracking URL: http://oxclo:8088/proxy/application_1447717288511_0001/
         user: oxclo
15/11/17 04:20:30 INFO util.ShutdownHookManager: Shutdown hook called
15/11/17 04:20:30 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-5
0dabdb1-d7d5-4bbe-8460-e3e57e868383
```

32. In the YARN web console, click on the FINISHED link in the left hand menu. You should see:



FINISHED Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unl N
1	0	0	1	0	0 B	8 GB	0 B	0	8	0	1	0	0	0

Scheduler Metrics

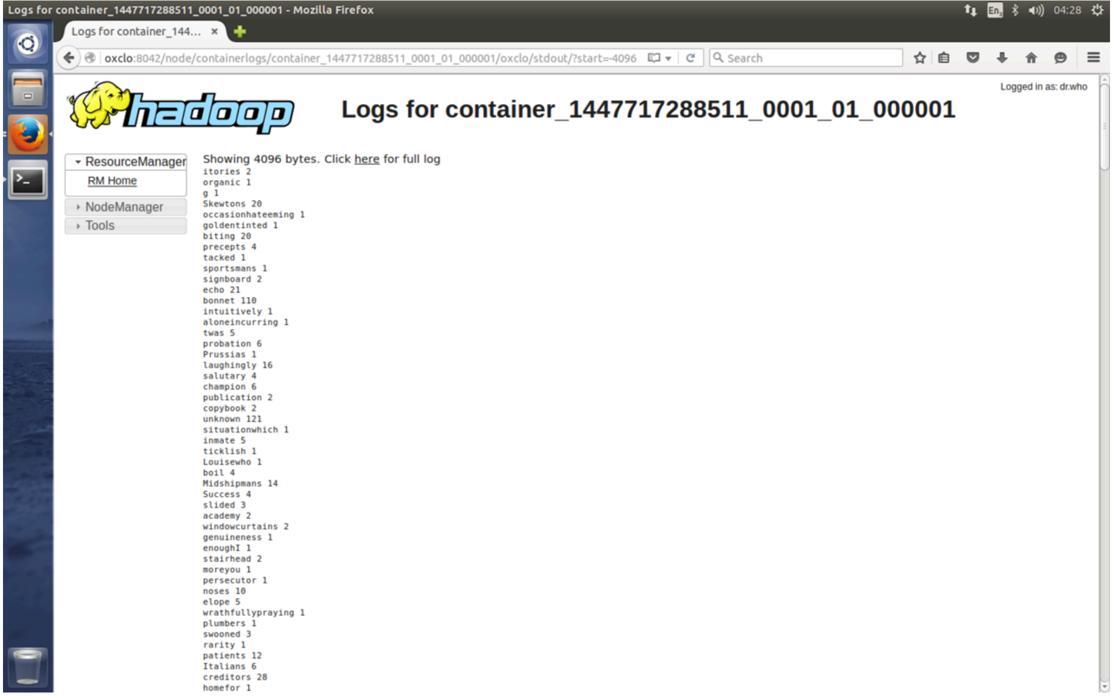
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Alloc
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progres
application_1447717288511_0001	oxclo	wc.py	SPARK	default	Tue Nov 17 04:19:51 +0000 2015	Tue Nov 17 04:20:30 +0000 2015	FINISHED	SUCCEEDED	

Showing 1 to 1 of 1 entries First Previo

33. Click on the application link and then click on the Logs link.
34. You can now find the stdout log from your code which has the result of the MR job:


Logs for container_1447717288511_0001_01_000001 - Mozilla Firefox
Logs for container_144...
oxclo:8042/node/containerlogs/container_1447717288511_0001_01_000001/oxclo/stdout/?start=4096
Search
04:28
Logged in as: dr.who
hadoop
ResourceManager
RM Home
NodeManager
Tools

Logs for container_1447717288511_0001_01_000001

Showing 4096 bytes. Click here for full log

```
itories 2
organic 1
g 1
Skewtons 20
occasionhateeming 1
goldentinted 1
biting 20
precepts 4
tacked 1
sportsmans 1
signboard 2
echo 21
bonnet 110
intuitively 1
aloneincurring 1
twas 5
probation 6
Prussias 1
laughingly 16
salutary 4
champion 6
publication 2
copybook 2
unknown 121
situationwhich 1
inmate 5
ticklish 1
Louisewho 1
boil 4
Midshipmans 14
Success 4
slided 3
academy 2
windowcurtains 2
genuineness 1
enoughI 1
stairhead 2
moreyou 1
persecutor 1
noses 10
elope 5
wrathfullypraying 1
plumbers 1
swooned 3
rarity 1
patients 12
Italians 6
creditors 28
homefor 1
```

35. You can also run jobs locally on a single node without using YARN: The local[*] indicates to use as many threads as you have cores on your system:

```
~/spark-1.5.1/bin/spark-submit --master local[*] wc.py
"hdfs://localhost:54310/user/oxclo/books/*"
```

36. You could also use Spark's own cluster manager or Apache Mesos as other options if you have those set up.

37. Congratulations, the lab is complete!