Lab: Getting Started with Apache Spark

About this Lab

Objective:	Read and manipulate HDFS files with Spark.
File locations:	/root/devph/labs/Spark
Successful outcome:	You will have processed several HDFS file via Spark Core.
Before you begin:	Your HDP 2.3 cluster should be up and running within your VM.
Related lesson:	***Programming with Apache Spark***

Lab Steps

1) Execute a WordCount with Spark.

a. If not already done, open a Terminal in your VM and type "ssh sandbox".
b. Copy the constitution.txt file to HDFS.

```
[root@sandbox ~]# cd ~/devph/labs/Spark
[root@sandbox Spark]# hdfs dfs -mkdir spark
[root@sandbox Spark]# hdfs dfs -put
~/devph/labs/demos/constitution.txt spark/
[root@sandbox Spark]# hdfs dfs -ls spark
Found 1 items
-rw-r--r--   1 root hdfs      45489 2015-11-09 09:39
spark/constitution.txt
```

c. Launch the Python Spark Shell. NOTE: "INFO" comments will be removed from the output in this lab guide going forward.

```
[root@sandbox ~]# pyspark
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2

... lines removed ...

Welcome to
____              __
/ __/__  ___ _____/ /__
_\ \/ _ \/ _ `/ __/  '_/
/__ / .__/\_,_/_/ /_/\_\   version 1.4.1
/_/

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

d. Read in the file as a RDD.

```
>>> baseFile =
sc.textFile("hdfs://sandbox:8020/user/root/spark/constitution.txt"
)
>>> baseFile.take(1)
[u'We the People of the United States, in Order to form a more
perfect ']
```

e. Break the full lines down into a collection of words.

```
>>> justWords = baseFile.flatMap(lambda line: line.split(' '))
>>> justWords.take(5)
[u'We', u'the', u'People', u'of', u'the']
```

f. Map the words with a count of "1" for each.

```
>>> mappedWords = justWords.map(lambda word: (word, 1))
>>> mappedWords.take(5)
[(u'We', 1), (u'the', 1), (u'People', 1), (u'of', 1), (u'the', 1)]
```

g. Count up the words; sorting them in reverse.

```
>>> wordCounts = mappedWords.reduceByKey(lambda a,b:
a+b).sortByKey(ascending=False)
>>> wordCounts.take(10)
[(u'years;', 1), (u'years', 9), (u'year,', 1), (u'year', 1),
(u'written', 6), (u'writs', 1), (u'writing,', 1), (u'would', 2),
(u'work', 1), (u'witnesses', 2)]
```

h. Chain all the method invocations into a single operation, as is more the normal usage pattern. NOTE: Type as a single line without the new line or "\" characters.

```
>>> asOneLine =
sc.textFile("hdfs://sandbox:8020/user/root/spark/constitution.txt"
) \
   .flatMap(lambda line: line.split(' ')) \
   .map(lambda word: (word, 1)) \
   .reduceByKey(lambda a,b: a+b) \
   .sortByKey(ascending=False)
>>> asOneLine.take(10)
[(u'years;', 1), (u'years', 9), (u'year,', 1), (u'year', 1),
(u'written', 6), (u'writs', 1), (u'writing,', 1), (u'would', 2),
(u'work', 1), (u'witnesses', 2)]
```

i. Exit out of the pyspark REPL.

```
>>> quit()
```

2) On a simple customer file, find the top 5 states with the most male customers.

a. Upload customer.csv and explore its format of name, gender, state and duration.

```
[root@sandbox Spark]# hdfs dfs -put customer.csv spark
[root@sandbox Spark]# hdfs dfs -tail spark/customer.csv
celia,F,Maryland,3.97
Evalyn,F,Pennsylvania,2.1
Jeneva,F,Nebraska,9.26
Kelsey,F,Minnesota,8.68
Daine,F,Nebraska,6.34

   ... lines removed ...

Annamae,F,Nebraska,9.11
Racheal,F,Wisconsin,9.65
Ellan,F,Michigan,5.82
```

b. Launch pyspark and read the file.

```
[root@sandbox labs]# pyspark
>>> custFile =
sc.textFile("hdfs://sandbox:8020/user/root/spark/customer.csv")
>>> custFile.take(3)
[u'Irvin,M,Maryland,5.06', u'Owen,M,Illinois,2.01',
u'August,M,Illinois,1.42']
```

c. Filter out just the male customers.

```
>>> justMales = custFile.map(lambda line:
line.split(',')).filter(lambda line: line[1] == 'M')
>>> justMales.take(2)
[[u'Irvin', u'M', u'Maryland', u'5.06'], [u'Owen', u'M',
u'Illinois', u'2.01']]
```

d. Create Key-Value-Pairs (KVP) with state being the key and the number "1" being the value.

```
>>> mapByStateCode = justMales.map(lambda line: (line[2], 1))
>>> mapByStateCode.take(4)
[(u'Maryland', 1), (u'Illinois', 1), (u'Illinois', 1), (u'New
Jersey', 1)]
```

e. Count up the number of customers by state.

```
>>> nbrCustsByState = mapByStateCode.reduceByKey(lambda a,b: a+b)
>>> nbrCustsByState.take(4)
[(u'Wisconsin', 2), (u'New Jersey', 4), (u'Michigan', 8),
(u'Pennsylvania', 2)]
```

f. Flip the KVP so that the count is first and order that from highest to lowest.

```
>>> highToLowCountAndState = nbrCustsByState.map(lambda (a,b):
(b,a)).sortByKey(ascending=False)
>>> highToLowCountAndState.take(6)
[(8, u'Michigan'), (8, u'Maryland'), (7, u'Illinois'), (5,
u'Nebraska'), (4, u'New Jersey'), (4, u'Indiana')]
```

g. Flip the KVP pair back to state and count plus add an index to represent the ordering sequence.

```
>>> stateCountIndexedHighToLow = highToLowCountAndState.map(lambda
(a,b): (b,a)).zipWithIndex()
>>> stateCountIndexedHighToLow.take(6)
[((u'Michigan', 8), 0), ((u'Maryland', 8), 1), ((u'Illinois', 7),
2), ((u'Nebraska', 5), 3), ((u'New Jersey', 4), 4), ((u'Indiana',
4), 5)]
```

h. Eliminate all records except for the top 5.

```
>>> topFive = stateCountIndexedHighToLow.filter(lambda ((a,b),c):
c<5)
>>> topFive.collect()
[((u'Michigan', 8), 0), ((u'Maryland', 8), 1), ((u'Illinois', 7),
2), ((u'Nebraska', 5), 3), ((u'New Jersey', 4), 4)]
```

i. Eliminate the index and the counts to just return the top 5 state names.

```
>>> top5Names = topFive.map(lambda ((a,b),c): a)
>>> top5Names.collect()
[u'Michigan', u'Maryland', u'Illinois', u'Nebraska', u'New
Jersey']
```

j. As before, chain all the method invocations into a single operation as is more the normal usage pattern.

```
>>>
sc.textFile("hdfs://sandbox:8020/user/root/spark/customer.csv") \
   .map(lambda line: line.split(',')) \
   .filter(lambda line: line[1] == 'M') \
   .map(lambda line: (line[2], 1)) \
   .reduceByKey(lambda a,b: a+b) \
   .map(lambda (a,b): (b,a)) \
   .sortByKey(ascending=False) \
   .map(lambda (a,b): (b,a)) \
   .zipWithIndex() \
   .filter(lambda ((a,b),c): c<5) \
   .map(lambda ((a,b),c): a) \
   .collect()
[u'Michigan', u'Maryland', u'Illinois', u'Nebraska', u'New
Jersey']
```

Result:

Successful use of Spark Core and RDD to read files and perform data analysis.