

Spark Fundamentals

Getting started with Spark

Contents

GETTING STARTED WITH SPARK 3

1.1 SETTING UP THE LAB ENVIRONMENT 4

1.2 STARTING UP THE SPARK SHELL USING SCALA 4

1.3 MORE ON RDD OPERATIONS USING SCALA..... 9

1.4 STARTING UP THE SPARK SHELL USING PYTHON..... 13

1.5 MORE ON RDD OPERATIONS USING PYTHON 18

1.6 USING SPARK CACHING 21

SUMMARY 22

Getting started with Spark

Spark is built around speed and the ease of use. In this section, you will see for yourself how easy it is to get started using Spark with IBM BigInsights. You will be doing some interactive analysis with the Spark Shell, which comes in two flavors. It is available in either Scala or Python. Scala runs on the Java VM and is thus a good way to use existing Java libraries.

Spark's primary abstraction is a distributed collection of items called a Resilient Distributed Dataset or RDD. In a subsequent lab exercise, you will learn more about the details of RDD. In this lab exercise, you'll get to see quickly how it works. RDDs have actions, which return values, and transformations, which return pointers to new RDD. In this lab exercise, you can decide if you wish to do the exercises in the Scala shell or the Python shell – both goes through the same steps, just slightly different syntax.

After completing this hands-on lab, you should be able to:

- Start the Spark shell with Scala and Python
- Perform basic RDD actions and transformations
- Use caching to speed up repeated operations

Allow 30-45 minutes to complete this section of lab.

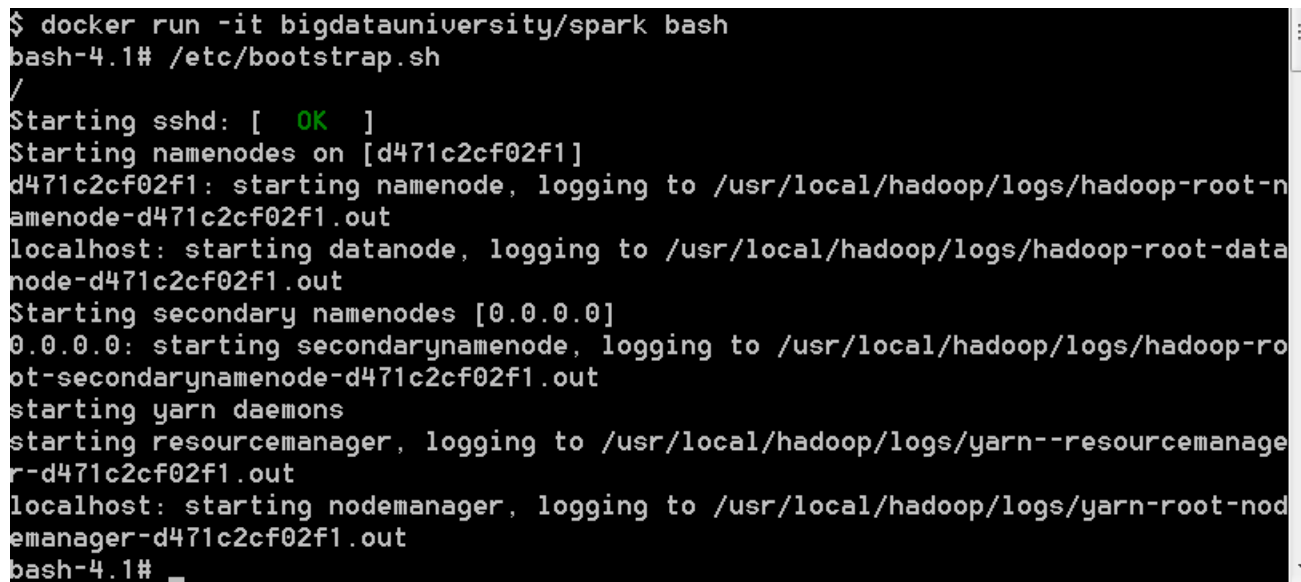
1.1 Setting up the lab environment

The lab was also designed with Spark 1.1, but tested with Spark 1.3.1. The lab environment is using the docker image.

One thing to note: sometimes after you run a command (either Scala or Python), the terminal may seem like got stuck. When that happens, just hit the Enter key to get it back to the prompt.

- ___1. Once you have loaded the docker image, you need to start up the Hadoop service. Type in:

```
/etc/bootstrap.sh
```



```
$ docker run -it bigdatauniversity/spark bash
bash-4.1# /etc/bootstrap.sh
/
Starting sshd: [ OK ]
Starting namenodes on [d471c2cf02f1]
d471c2cf02f1: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-n
amenode-d471c2cf02f1.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-data
node-d471c2cf02f1.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-ro
ot-secondarynamenode-d471c2cf02f1.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn--resourcemanage
r-d471c2cf02f1.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nod
emanager-d471c2cf02f1.out
bash-4.1#
```

1.2 Starting up the Spark Shell using Scala

Spark's shell provides a simple way to learn the APIs. In this section, you will get to use both Scala and Python to work with Spark's API. The first half will cover the shell using Scala. The second half will show the same steps, but using Python instead.

- ___1. Copy the README.md file from the /opt/ibm/labfiles directory into the input/tmp directory on the HDFS. If you don't have the *labfiles* directory, go back to the BDU page and download and set up those files. In the docker window, create the input/tmp directory.

```
hdfs dfs -mkdir input/tmp
```

- ___2. Copy the README.md file to the newly created hdfs directory.

```
hdfs dfs -put /opt/ibm/labfiles/README.md input/tmp
```

- ___3. Make sure you successfully copied the README file by doing a listing of the tmp directory.

```
hdfs dfs -ls input/tmp
```

- __4. Start the Spark shell with this command:

```
$SPARK_HOME/bin/spark-shell
```

```

MINGW32/c/Users/IBM_ADMIN
:49108
15/04/27 16:54:53 INFO util.Utils: Successfully started service 'HTTP file server' on port 49108.
15/04/27 16:54:53 INFO spark.SparkEnv: Registering OutputCommitCoordinator
15/04/27 16:54:53 INFO server.Server: jetty-8.y.z-SNAPSHOT
15/04/27 16:54:53 INFO server.AbstractConnector: Started SelectChannelConnector@0.0.0.0:4040
15/04/27 16:54:53 INFO util.Utils: Successfully started service 'SparkUI' on port 4040.
15/04/27 16:54:53 INFO ui.SparkUI: Started SparkUI at http://d471c2cf02f1:4040
15/04/27 16:54:53 INFO executor.Executor: Starting executor ID <driver> on host localhost
15/04/27 16:54:53 INFO executor.Executor: Using REPL class URI: http://172.17.0.9:51689
15/04/27 16:54:53 INFO util.AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@d471c2cf02f1:42592/user/HeartbeatReceiver
15/04/27 16:54:53 INFO netty.NettyBlockTransferService: Server created on 57801
15/04/27 16:54:53 INFO storage.BlockManagerMaster: Trying to register BlockManager
15/04/27 16:54:53 INFO storage.BlockManagerMasterActor: Registering block manager localhost:57801 with 265.4 MB RAM, BlockManagerId(<driver>, localhost, 57801)
15/04/27 16:54:53 INFO storage.BlockManagerMaster: Registered BlockManager
15/04/27 16:54:53 INFO repl.SparkILoop: Created spark context..
Spark context available as sc.
15/04/27 16:54:54 INFO repl.SparkILoop: Created sql context (with Hive support).
SQL context available as sqlContext.

scala> _

```

- __5. The shell provides code assist. For example, type in "sc." followed by the Tab key to get the list of options associated with the spark context:

```

scala> sc.
accumulable          accumulableCollection  accumulator           addFile               addJar
addSparkListener    appName              asInstanceOfOf        broadcast             cancelAllJobs
cancelJobGroup       clearCallSite        clearFiles             clearJars             clearJobGroup
defaultMinPartitions defaultMinSplits     defaultParallelism    emptyRDD             files
getAllPools         getCheckpointDir     getConf               getExecutorMemoryStatus getExecutorStorageStatus
getLocalProperty    getPersistentRDDs    getPoolForName        getRDDStorageInfo    getSchedulingMode
hadoopConfiguration hadoopFile           hadoopRDD             initLocalProperties  isInstanceOf
isLocal              jars                 makeRDD               master               newAPIHadoopFile
newAPIHadoopRDD      objectFile           parallelize            runApproximateJob    runJob
sequenceFile         setCallSite          setCheckpointDir       setJobDescription    setJobGroup
setLocalProperty     sparkUser            startTime             stop                 submitJob
tachyonFolderName    textFile             toString              union                 version
wholeTextFiles
scala> sc.

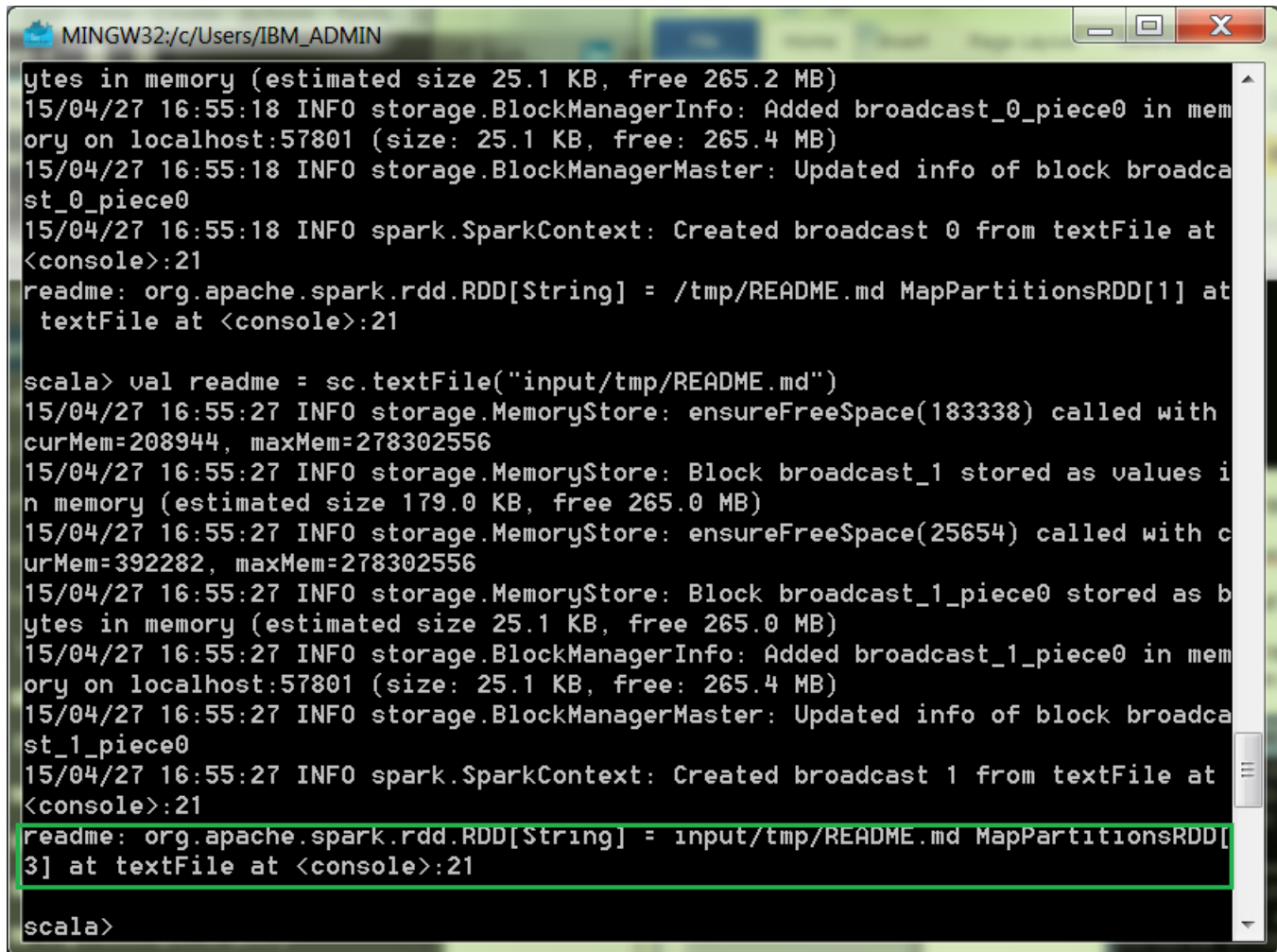
```

- __6. Note the path of the readme text file is on the HDFS. Type in:

```
val readme = sc.textFile("input/tmp/README.md")
```

The parameter of the `textFile` method is the location within the `/tmp` directory on the HDFS.

This was a RDD transformation, thus it return a pointer to a RDD, which we have named as *readme*. You can see from the returned line that the *readme* is a pointer to the RDD.



```

MINGW32/c/Users/IBM_ADMIN
ytes in memory (estimated size 25.1 KB, free 265.2 MB)
15/04/27 16:55:18 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in mem
ory on localhost:57801 (size: 25.1 KB, free: 265.4 MB)
15/04/27 16:55:18 INFO storage.BlockManagerMaster: Updated info of block broadca
st_0_piece0
15/04/27 16:55:18 INFO spark.SparkContext: Created broadcast 0 from textFile at
<console>:21
readme: org.apache.spark.rdd.RDD[String] = /tmp/README.md MapPartitionsRDD[1] at
textFile at <console>:21

scala> val readme = sc.textFile("input/tmp/README.md")
15/04/27 16:55:27 INFO storage.MemoryStore: ensureFreeSpace(183338) called with
curMem=208944, maxMem=278302556
15/04/27 16:55:27 INFO storage.MemoryStore: Block broadcast_1 stored as values i
n memory (estimated size 179.0 KB, free 265.0 MB)
15/04/27 16:55:27 INFO storage.MemoryStore: ensureFreeSpace(25654) called with c
urMem=392282, maxMem=278302556
15/04/27 16:55:27 INFO storage.MemoryStore: Block broadcast_1_piece0 stored as b
ytes in memory (estimated size 25.1 KB, free 265.0 MB)
15/04/27 16:55:27 INFO storage.BlockManagerInfo: Added broadcast_1_piece0 in mem
ory on localhost:57801 (size: 25.1 KB, free: 265.4 MB)
15/04/27 16:55:27 INFO storage.BlockManagerMaster: Updated info of block broadca
st_1_piece0
15/04/27 16:55:27 INFO spark.SparkContext: Created broadcast 1 from textFile at
<console>:21
readme: org.apache.spark.rdd.RDD[String] = input/tmp/README.md MapPartitionsRDD[
3] at textFile at <console>:21

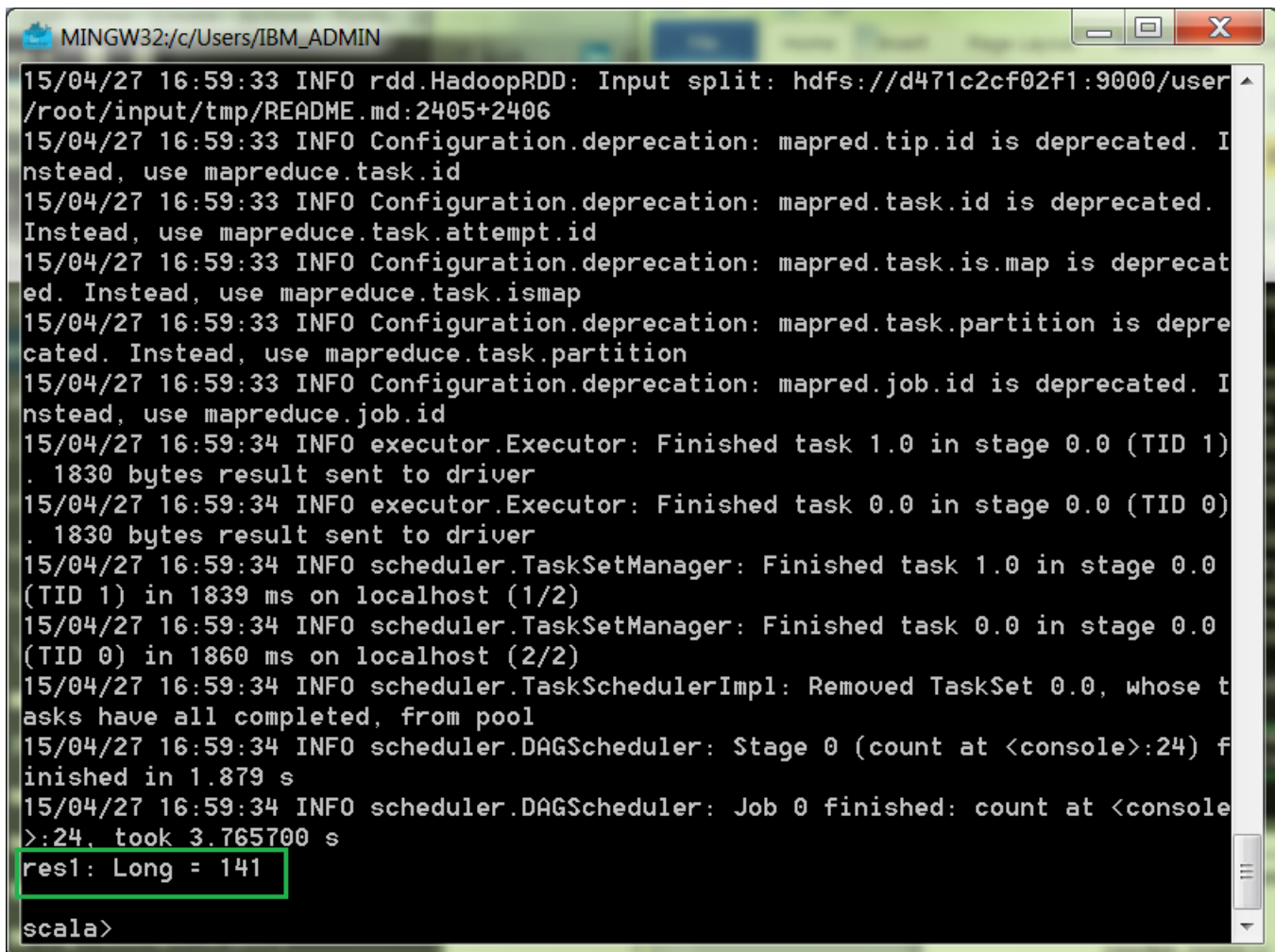
scala>

```

- ___7. Let's perform some RDD actions on this text file. Count the number of items in the RDD using this command:

```
readme.count()
```

You should see that this RDD action returned a value of 141.



```

MINGW32:/c/Users/IBM_ADMIN
15/04/27 16:59:33 INFO rdd.HadoopRDD: Input split: hdfs://d471c2cf02f1:9000/user
/root/input/tmp/README.md:2405+2406
15/04/27 16:59:33 INFO Configuration.deprecation: mapred.tip.id is deprecated. I
nstead, use mapreduce.task.id
15/04/27 16:59:33 INFO Configuration.deprecation: mapred.task.id is deprecated.
Instead, use mapreduce.task.attempt.id
15/04/27 16:59:33 INFO Configuration.deprecation: mapred.task.is.map is deprecate
d. Instead, use mapreduce.task.ismap
15/04/27 16:59:33 INFO Configuration.deprecation: mapred.task.partition is depre
cated. Instead, use mapreduce.task.partition
15/04/27 16:59:33 INFO Configuration.deprecation: mapred.job.id is deprecated. I
nstead, use mapreduce.job.id
15/04/27 16:59:34 INFO executor.Executor: Finished task 1.0 in stage 0.0 (TID 1)
. 1830 bytes result sent to driver
15/04/27 16:59:34 INFO executor.Executor: Finished task 0.0 in stage 0.0 (TID 0)
. 1830 bytes result sent to driver
15/04/27 16:59:34 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0
(TID 1) in 1839 ms on localhost (1/2)
15/04/27 16:59:34 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0
(TID 0) in 1860 ms on localhost (2/2)
15/04/27 16:59:34 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose t
asks have all completed, from pool
15/04/27 16:59:34 INFO scheduler.DAGScheduler: Stage 0 (count at <console>:24) f
inished in 1.879 s
15/04/27 16:59:34 INFO scheduler.DAGScheduler: Job 0 finished: count at <console
>:24, took 3.765700 s
res1: Long = 141
scala>

```

__8. Let's run another action. Run this command to find the first item in the RDD:

```
readme.first()
```

```

MINGW32/c/Users/IBM_ADMIN
ytes in memory (estimated size 1955.0 B, free 265.0 MB)
15/04/27 17:00:03 INFO storage.BlockManagerInfo: Added broadcast_3_piece0 in mem
ory on localhost:57801 (size: 1955.0 B, free: 265.4 MB)
15/04/27 17:00:03 INFO storage.BlockManagerMaster: Updated info of block broadca
st_3_piece0
15/04/27 17:00:03 INFO spark.SparkContext: Created broadcast 3 from broadcast at
DAGScheduler.scala:839
15/04/27 17:00:03 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from S
tage 1 (input/tmp/README.md MapPartitionsRDD[3] at textFile at <console>:21)
15/04/27 17:00:03 INFO scheduler.TaskSchedulerImpl: Adding task set 1.0 with 1 t
asks
15/04/27 17:00:03 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 1.0
(TID 2, localhost, ANY, 1318 bytes)
15/04/27 17:00:03 INFO executor.Executor: Running task 0.0 in stage 1.0 (TID 2)
15/04/27 17:00:03 INFO rdd.HadoopRDD: Input split: hdfs://d471c2cf02f1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:00:03 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 2)
. 1809 bytes result sent to driver
15/04/27 17:00:03 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0
(TID 2) in 25 ms on localhost (1/1)
15/04/27 17:00:03 INFO scheduler.DAGScheduler: Stage 1 (first at <console>:24) f
inished in 0.025 s
15/04/27 17:00:03 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose t
asks have all completed, from pool
15/04/27 17:00:03 INFO scheduler.DAGScheduler: Job 1 finished: first at <console
>:24, took 0.037941 s
res2: String = # Apache Spark
scala>

```

- ___9. Now let's try a transformation. Use the *filter* transformation to return a new RDD with a subset of the items in the file. Type in this command:

```
val linesWithSpark = readme.filter(line => line.contains("Spark"))
```

Again, this returned a pointer to a RDD with the results of the filter transformation.

- ___10. You can even chain together transformations and actions. To find out how many lines contains the word "Spark", type in:

```
readme.filter(line => line.contains("Spark")).count()
```



```

MINGW32:/c/Users/IBM_ADMIN
15/04/27 17:00:41 INFO scheduler.TaskSchedulerImpl: Adding task set 2.0 with 2 t
asks
15/04/27 17:00:41 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 2.0
(TID 3, localhost, ANY, 1318 bytes)
15/04/27 17:00:41 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 2.0
(TID 4, localhost, ANY, 1318 bytes)
15/04/27 17:00:41 INFO executor.Executor: Running task 0.0 in stage 2.0 (TID 3)
15/04/27 17:00:41 INFO executor.Executor: Running task 1.0 in stage 2.0 (TID 4)
15/04/27 17:00:41 INFO rdd.HadoopRDD: Input split: hdfs://d471c2cf02f1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:00:41 INFO rdd.HadoopRDD: Input split: hdfs://d471c2cf02f1:9000/user
/root/input/tmp/README.md:2405+2406
15/04/27 17:00:41 INFO executor.Executor: Finished task 1.0 in stage 2.0 (TID 4)
. 1830 bytes result sent to driver
15/04/27 17:00:41 INFO executor.Executor: Finished task 0.0 in stage 2.0 (TID 3)
. 1830 bytes result sent to driver
15/04/27 17:00:41 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 2.0
(TID 4) in 27 ms on localhost (1/2)
15/04/27 17:00:41 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 2.0
(TID 3) in 30 ms on localhost (2/2)
15/04/27 17:00:41 INFO scheduler.DAGScheduler: Stage 2 (count at <console>:24) f
inished in 0.030 s
15/04/27 17:00:41 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose t
asks have all completed, from pool
15/04/27 17:00:41 INFO scheduler.DAGScheduler: Job 2 finished: count at <console
>:24, took 0.043838 s
res3: Long = 21
scala>

```

__11. Do not close the shell. The rest of the exercises will build upon this one.

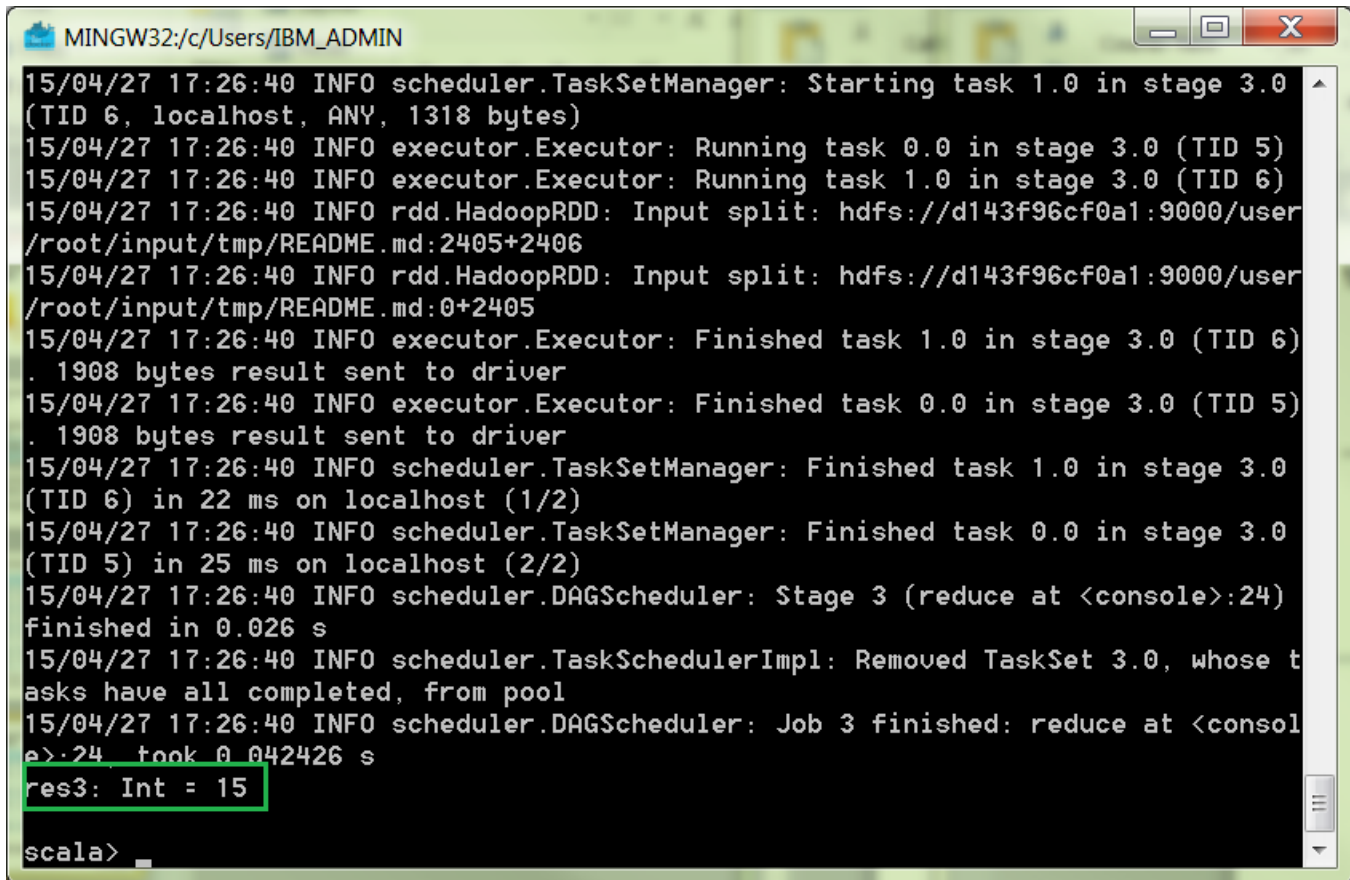
1.3 More on RDD Operations using Scala

This section builds upon the previous section. In this section, you will see that RDD can be used for more complex computations. You will find the line from that readme file with the most words in it.

__1. Using the scala shell, copy and paste:

```
readme.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)
```

There are two parts to this. The first maps a line to an integer value, the number of words in that line. In the second part reduce is called to find the line with the most words in it. The arguments to *map* and *reduce* are Scala function literals (closures), but you can use any language feature or Scala/Java library.



```

MINGW32:/c/Users/IBM_ADMIN
15/04/27 17:26:40 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 3.0
(TID 6, localhost, ANY, 1318 bytes)
15/04/27 17:26:40 INFO executor.Executor: Running task 0.0 in stage 3.0 (TID 5)
15/04/27 17:26:40 INFO executor.Executor: Running task 1.0 in stage 3.0 (TID 6)
15/04/27 17:26:40 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:2405+2406
15/04/27 17:26:40 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:26:40 INFO executor.Executor: Finished task 1.0 in stage 3.0 (TID 6)
. 1908 bytes result sent to driver
15/04/27 17:26:40 INFO executor.Executor: Finished task 0.0 in stage 3.0 (TID 5)
. 1908 bytes result sent to driver
15/04/27 17:26:40 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 3.0
(TID 6) in 22 ms on localhost (1/2)
15/04/27 17:26:40 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 3.0
(TID 5) in 25 ms on localhost (2/2)
15/04/27 17:26:40 INFO scheduler.DAGScheduler: Stage 3 (reduce at <console>:24)
finished in 0.026 s
15/04/27 17:26:40 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose t
asks have all completed, from pool
15/04/27 17:26:40 INFO scheduler.DAGScheduler: Job 3 finished: reduce at <consol
e>:24, took 0.042426 s
res3: Int = 15
scala>

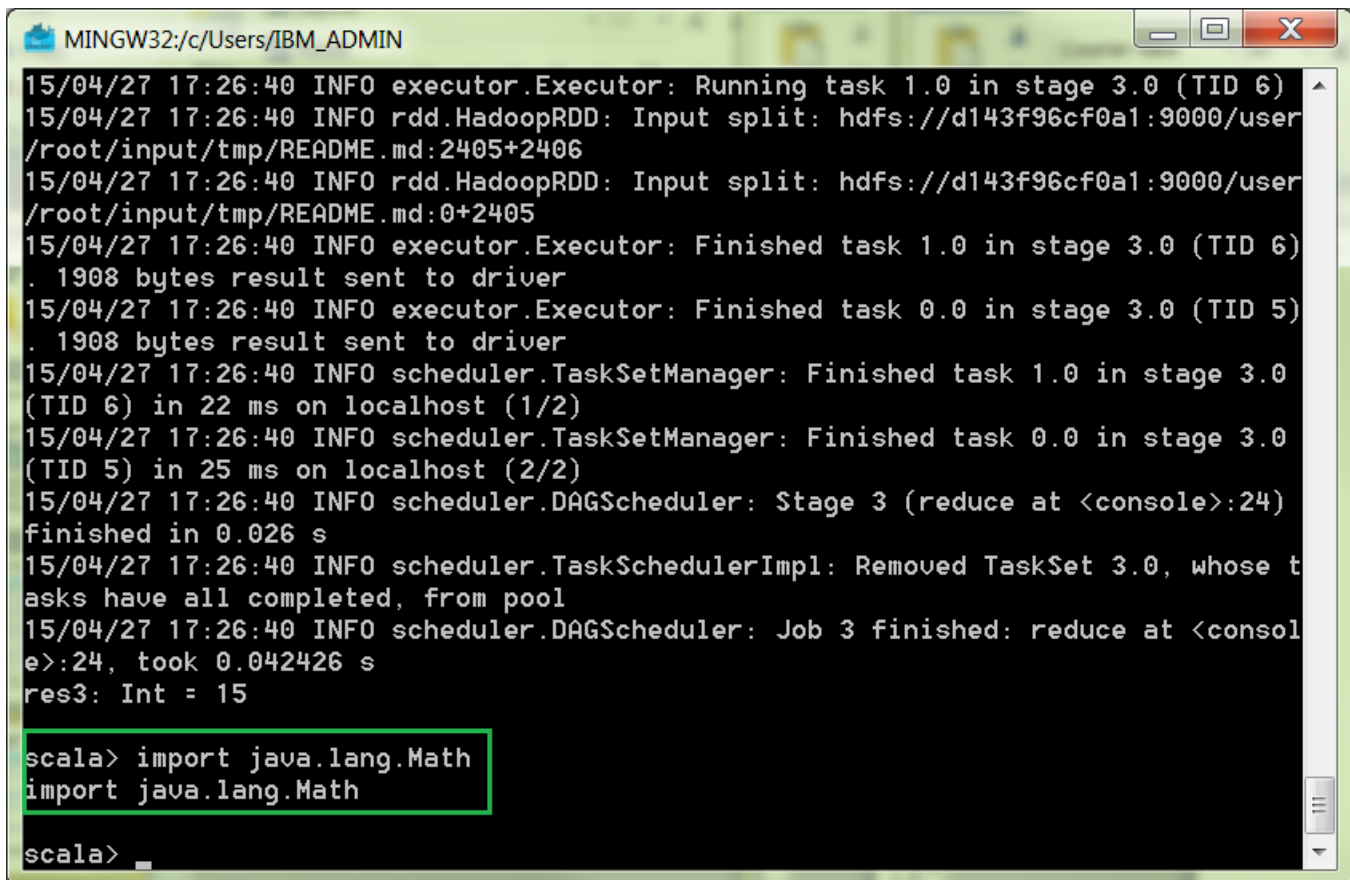
```

Line 15 contains the most words in it.

In the next step, you use the `Math.max()` function to show that you can indeed use a Java library instead.

__2. Import in the `java.lang.Math` library. Copy/paste:

```
import java.lang.Math
```



```
MINGW32:/c/Users/IBM_ADMIN
15/04/27 17:26:40 INFO executor.Executor: Running task 1.0 in stage 3.0 (TID 6)
15/04/27 17:26:40 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:2405+2406
15/04/27 17:26:40 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:26:40 INFO executor.Executor: Finished task 1.0 in stage 3.0 (TID 6)
. 1908 bytes result sent to driver
15/04/27 17:26:40 INFO executor.Executor: Finished task 0.0 in stage 3.0 (TID 5)
. 1908 bytes result sent to driver
15/04/27 17:26:40 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 3.0
(TID 6) in 22 ms on localhost (1/2)
15/04/27 17:26:40 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 3.0
(TID 5) in 25 ms on localhost (2/2)
15/04/27 17:26:40 INFO scheduler.DAGScheduler: Stage 3 (reduce at <console>:24)
finished in 0.026 s
15/04/27 17:26:40 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose t
asks have all completed, from pool
15/04/27 17:26:40 INFO scheduler.DAGScheduler: Job 3 finished: reduce at <consol
e>:24, took 0.042426 s
res3: Int = 15

scala> import java.lang.Math
import java.lang.Math

scala> _
```

__3. Now copy and paste in the following to run with the max function:

```
readme.map(line => line.split(" ").size).reduce((a, b) => Math.max(a, b))
```

```

Select MINGW32:/c/Users/IBM_ADMIN
15/04/27 17:27:40 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 4.0
(TID 8, localhost, ANY, 1318 bytes)
15/04/27 17:27:40 INFO executor.Executor: Running task 0.0 in stage 4.0 (TID 7)
15/04/27 17:27:40 INFO executor.Executor: Running task 1.0 in stage 4.0 (TID 8)
15/04/27 17:27:40 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:2405+2406
15/04/27 17:27:40 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:27:40 INFO executor.Executor: Finished task 0.0 in stage 4.0 (TID 7)
. 1908 bytes result sent to driver
15/04/27 17:27:40 INFO executor.Executor: Finished task 1.0 in stage 4.0 (TID 8)
. 1908 bytes result sent to driver
15/04/27 17:27:40 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0
(TID 7) in 25 ms on localhost (1/2)
15/04/27 17:27:40 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 4.0
(TID 8) in 25 ms on localhost (2/2)
15/04/27 17:27:40 INFO scheduler.DAGScheduler: Stage 4 (reduce at <console>:25)
finished in 0.026 s
15/04/27 17:27:40 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose t
asks have all completed, from pool
15/04/27 17:27:40 INFO scheduler.DAGScheduler: Job 4 finished: reduce at <consol
e>:25, took 0.038541 s
res4: Int = 15
scala>

```

- __4. Spark has a MapReduce data flow pattern. We can use this to do a word count on the readme file. Copy and paste in:

```
val wordCounts = readme.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a,b) => a + b)
```

Here we combined the *flatMap*, *map*, and the *reduceByKey* functions to do a word count of each word in the readme file.

- __5. To collect the word counts, use the *collect* action.

```
wordCounts.collect()
```

```

MINGW32:/c/Users/IBM_ADMIN
). 4667 bytes result sent to driver
15/04/27 17:29:41 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 6.0
(TID 11) in 125 ms on localhost (1/2)
15/04/27 17:29:41 INFO executor.Executor: Finished task 1.0 in stage 6.0 (TID 12)
). 5111 bytes result sent to driver
15/04/27 17:29:41 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 6.0
(TID 12) in 134 ms on localhost (2/2)
15/04/27 17:29:41 INFO scheduler.DAGScheduler: Stage 6 (collect at <console>:27)
finished in 0.140 s
15/04/27 17:29:41 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 6.0, whose t
asks have all completed, from pool
15/04/27 17:29:41 INFO scheduler.DAGScheduler: Job 5 finished: collect at <conso
le>:27, took 0.400836 s
res5: Array[(String, Int)] = Array((means,1), (under,2), (this,4), (Because,1),
(Python,2), (agree,1), (cluster,1), (its,1), (follows,1), (YARN,3), (general,
2), (have,2), (pre-built,1), (MRv1,1), (locally,1), (changed,1), (locally,2),
(sc.parallelize(1,1), (only,1), (several,1), (This,2), (first,1), (basic,1), (le
arning,1), (documentation,1), (Configuration,1), (MapReduce,2), (CLI,1), (graph
,1), (requests,1), (without,1), ("yarn-client",1), ([params],1), (any,2), (set
ting,2), (application,1), (prefer,1), (SparkPi,2), (<http://spark.apache.org/>,1
), (version,3), (file,1), (documentation,1), (engine,1), (MASTER,1), (entry,1),
(example,3), (are,2), (systems,1), (params,1), (scala,1), (provides,1), (refe
r,1), (MLLib,1), (Interactive,2), (artifact,1), (configure,1), (can,8), (<art...
scala>

```

You can see the partial results using the collect action

- __6. You can quit out of the Scala shell, you have completed this section. To stop the spark contentType in:

CTRL +D

1.4 Starting up the Spark Shell using Python

This section goes over the same tasks as the Scala section, but using the Python shell.

- __1. Load the README file from the local system onto HDFS. If you had done this already, you can skip this step. Execute this command:

```
hdfs dfs -put /opt/ibm/labfiles/README.md input/tmp
```

- __2. Start the Python Spark shell with this command:

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

```
MINGW32:/c:/Users/IBM_ADMIN
```

```
15/04/27 17:40:43 INFO server.AbstractConnector: Started SelectChannelConnector@
0.0.0.0:4040
15/04/27 17:40:43 INFO util.Utils: Successfully started service 'SparkUI' on por
t 4040.
15/04/27 17:40:43 INFO ui.SparkUI: Started SparkUI at http://d143f96cf0a1:4040
15/04/27 17:40:43 INFO executor.Executor: Starting executor ID <driver> on host
localhost
15/04/27 17:40:43 INFO util.AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp
://sparkDriver@d143f96cf0a1:60623/user/HeartbeatReceiver
15/04/27 17:40:43 INFO netty.NettyBlockTransferService: Server created on 37907
15/04/27 17:40:43 INFO storage.BlockManagerMaster: Trying to register BlockManag
er
15/04/27 17:40:43 INFO storage.BlockManagerMasterActor: Registering block manage
r localhost:37907 with 265.4 MB RAM, BlockManagerId(<driver>, localhost, 37907)
15/04/27 17:40:43 INFO storage.BlockManagerMaster: Registered BlockManager
Welcome to

      _--_
     /    \
    /  V  \
   /    \  \
  /      \  \
 /        \  \
/_          \_\

version 1.3.1

Using Python version 2.6.6 (r266:84292, Jan 22 2014 09:42:36)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

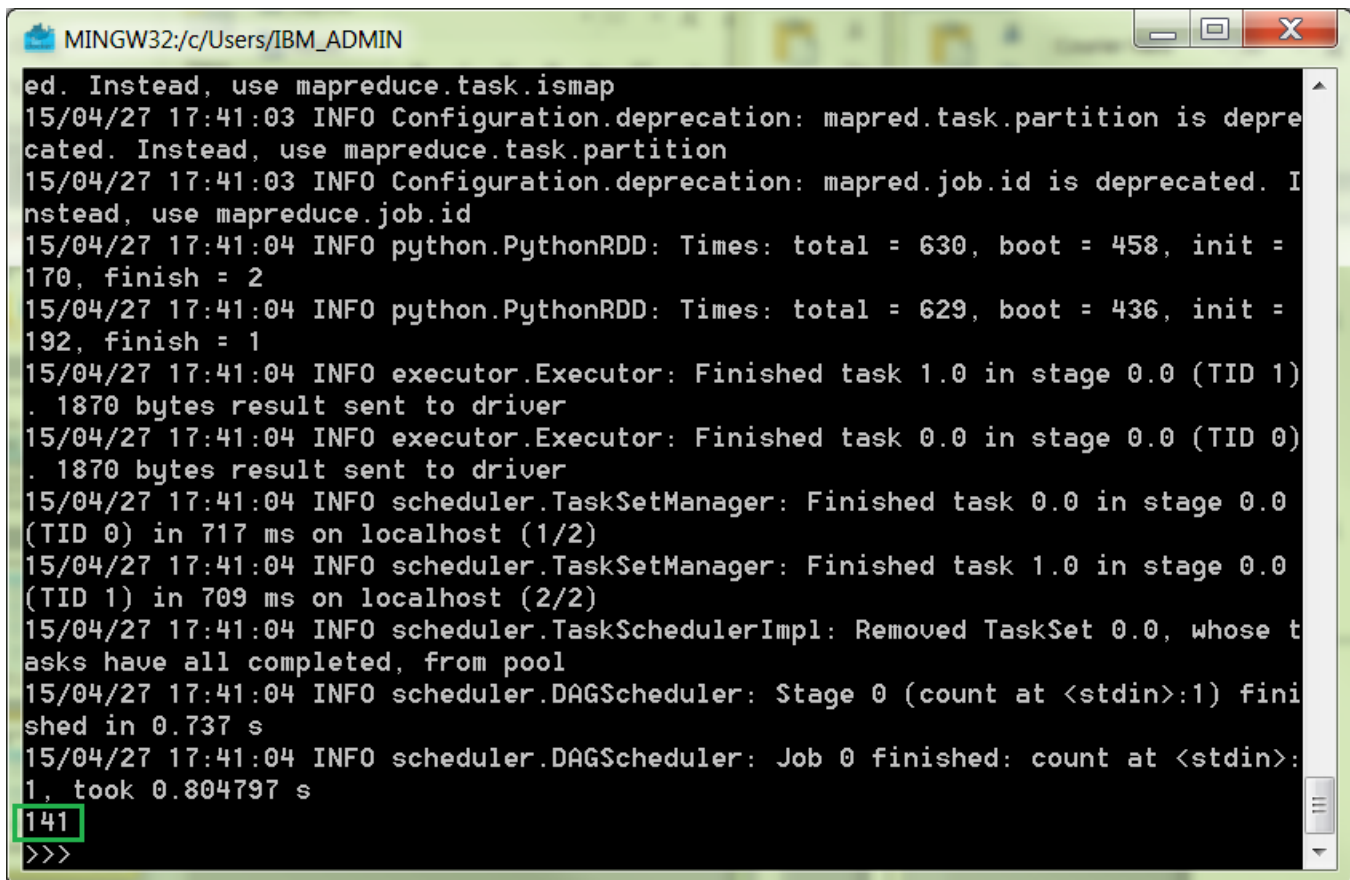
- __3. Let's start with the example of reading in a text file, and thus converting it to a RDD. Note that the path of the readme text file is on the HDFS. Type in:

```
readme = sc.textFile("input/tmp/README.md")
```

This was a RDD transformation, thus it return a pointer to a RDD, which we have named as *readme*.

- ___4. Let's perform some RDD actions on this text file. Count the number of items in the RDD using this command:

```
readme.count()
```



A screenshot of a MINGW32 terminal window titled 'MINGW32:/c/Users/IBM_ADMIN'. The window displays a series of log messages from the Spark framework, including deprecation warnings and task completion status. The logs indicate that a job has finished successfully. At the bottom of the terminal, the number '141' is displayed, which is the result of a previous RDD action. Below the result, the prompt '>>>' is visible, indicating that the terminal is ready for the next command.

```
ed. Instead, use mapreduce.task.ismap
15/04/27 17:41:03 INFO Configuration.deprecation: mapred.task.partition is depre
cated. Instead, use mapreduce.task.partition
15/04/27 17:41:03 INFO Configuration.deprecation: mapred.job.id is deprecated. I
nstead, use mapreduce.job.id
15/04/27 17:41:04 INFO python.PythonRDD: Times: total = 630, boot = 458, init =
170, finish = 2
15/04/27 17:41:04 INFO python.PythonRDD: Times: total = 629, boot = 436, init =
192, finish = 1
15/04/27 17:41:04 INFO executor.Executor: Finished task 1.0 in stage 0.0 (TID 1)
. 1870 bytes result sent to driver
15/04/27 17:41:04 INFO executor.Executor: Finished task 0.0 in stage 0.0 (TID 0)
. 1870 bytes result sent to driver
15/04/27 17:41:04 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0
(TID 0) in 717 ms on localhost (1/2)
15/04/27 17:41:04 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0
(TID 1) in 709 ms on localhost (2/2)
15/04/27 17:41:04 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose t
asks have all completed, from pool
15/04/27 17:41:04 INFO scheduler.DAGScheduler: Stage 0 (count at <stdin>:1) fini
shed in 0.737 s
15/04/27 17:41:04 INFO scheduler.DAGScheduler: Job 0 finished: count at <stdin>:
1, took 0.804797 s
141
>>>
```

You see that this RDD action returned a value of 141.

- __5. Let's run another action. Run this command to find the first item in the RDD:

```
readme.first()
```

```

MINGW32:/c/Users/IBM_ADMIN
15/04/27 17:43:07 INFO spark.SparkContext: Created broadcast 2 from broadcast at
DAGScheduler.scala:839
15/04/27 17:43:07 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from S
tage 1 (PythonRDD[3] at RDD at PythonRDD.scala:43)
15/04/27 17:43:07 INFO scheduler.TaskSchedulerImpl: Adding task set 1.0 with 1 t
asks
15/04/27 17:43:07 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 1.0
(TID 2, localhost, ANY, 1318 bytes)
15/04/27 17:43:07 INFO executor.Executor: Running task 0.0 in stage 1.0 (TID 2)
15/04/27 17:43:07 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:43:07 INFO python.PythonRDD: Times: total = 15, boot = 2, init = 12,
finish = 1
15/04/27 17:43:07 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 2)
. 1889 bytes result sent to driver
15/04/27 17:43:07 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0
(TID 2) in 30 ms on localhost (1/1)
15/04/27 17:43:07 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose t
asks have all completed, from pool
15/04/27 17:43:07 INFO scheduler.DAGScheduler: Stage 1 (runJob at PythonRDD.scal
a:362) finished in 0.031 s
15/04/27 17:43:07 INFO scheduler.DAGScheduler: Job 1 finished: runJob at PythonR
DD.scala:362, took 0.046056 s
u'# Apache Spark'
>>>

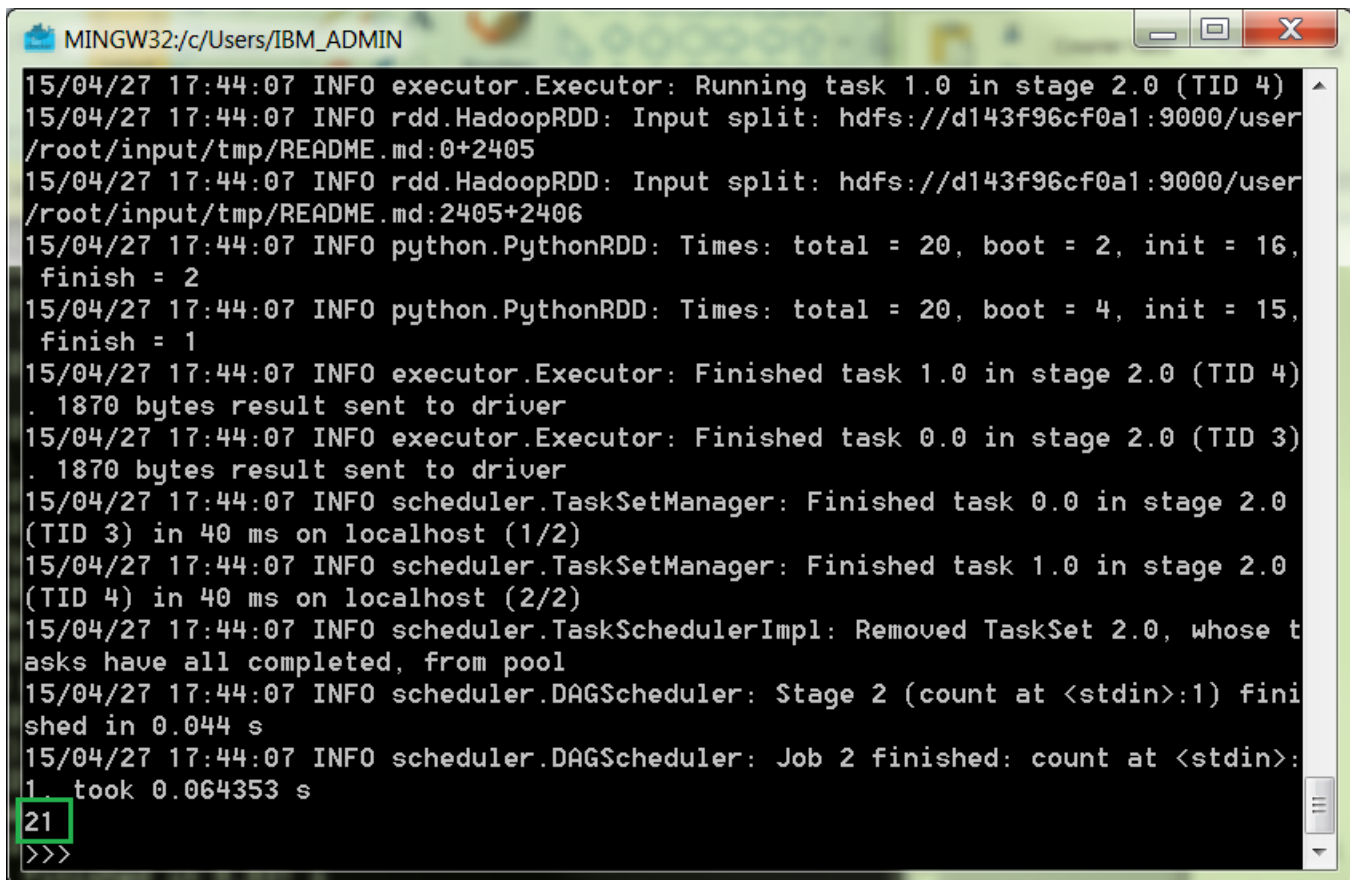
```

- ___6. Now let's try a transformation. Use the *filter* transformation to return a new RDD with a subset of the items in the file. Type in this command:

```
linesWithSpark = readme.filter(lambda line: "Spark" in line)
```

- ___7. You can even chain together transformations and actions. To find out how many lines contains the word "Spark", type in:

```
readme.filter(lambda line: "Spark" in line).count()
```

A screenshot of a Windows command prompt window titled "MINGW32:/c/Users/IBM_ADMIN". The window displays a series of log messages from a Hadoop job. The messages are timestamped "15/04/27 17:44:07" and include information about task execution, input splits, and scheduler activity. The logs show tasks running and finishing in stage 2.0, with results being sent to the driver. The job concludes with a message indicating it finished in 0.064353 seconds. At the bottom of the window, the number "21" is highlighted with a green box, and the prompt ">>>" is visible.

```
15/04/27 17:44:07 INFO executor.Executor: Running task 1.0 in stage 2.0 (TID 4)
15/04/27 17:44:07 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:44:07 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:2405+2406
15/04/27 17:44:07 INFO python.PythonRDD: Times: total = 20, boot = 2, init = 16,
finish = 2
15/04/27 17:44:07 INFO python.PythonRDD: Times: total = 20, boot = 4, init = 15,
finish = 1
15/04/27 17:44:07 INFO executor.Executor: Finished task 1.0 in stage 2.0 (TID 4)
. 1870 bytes result sent to driver
15/04/27 17:44:07 INFO executor.Executor: Finished task 0.0 in stage 2.0 (TID 3)
. 1870 bytes result sent to driver
15/04/27 17:44:07 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 2.0
(TID 3) in 40 ms on localhost (1/2)
15/04/27 17:44:07 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 2.0
(TID 4) in 40 ms on localhost (2/2)
15/04/27 17:44:07 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose t
asks have all completed, from pool
15/04/27 17:44:07 INFO scheduler.DAGScheduler: Stage 2 (count at <stdin>:1) fini
shed in 0.044 s
15/04/27 17:44:07 INFO scheduler.DAGScheduler: Job 2 finished: count at <stdin>:
1, took 0.064353 s
21
>>>
```

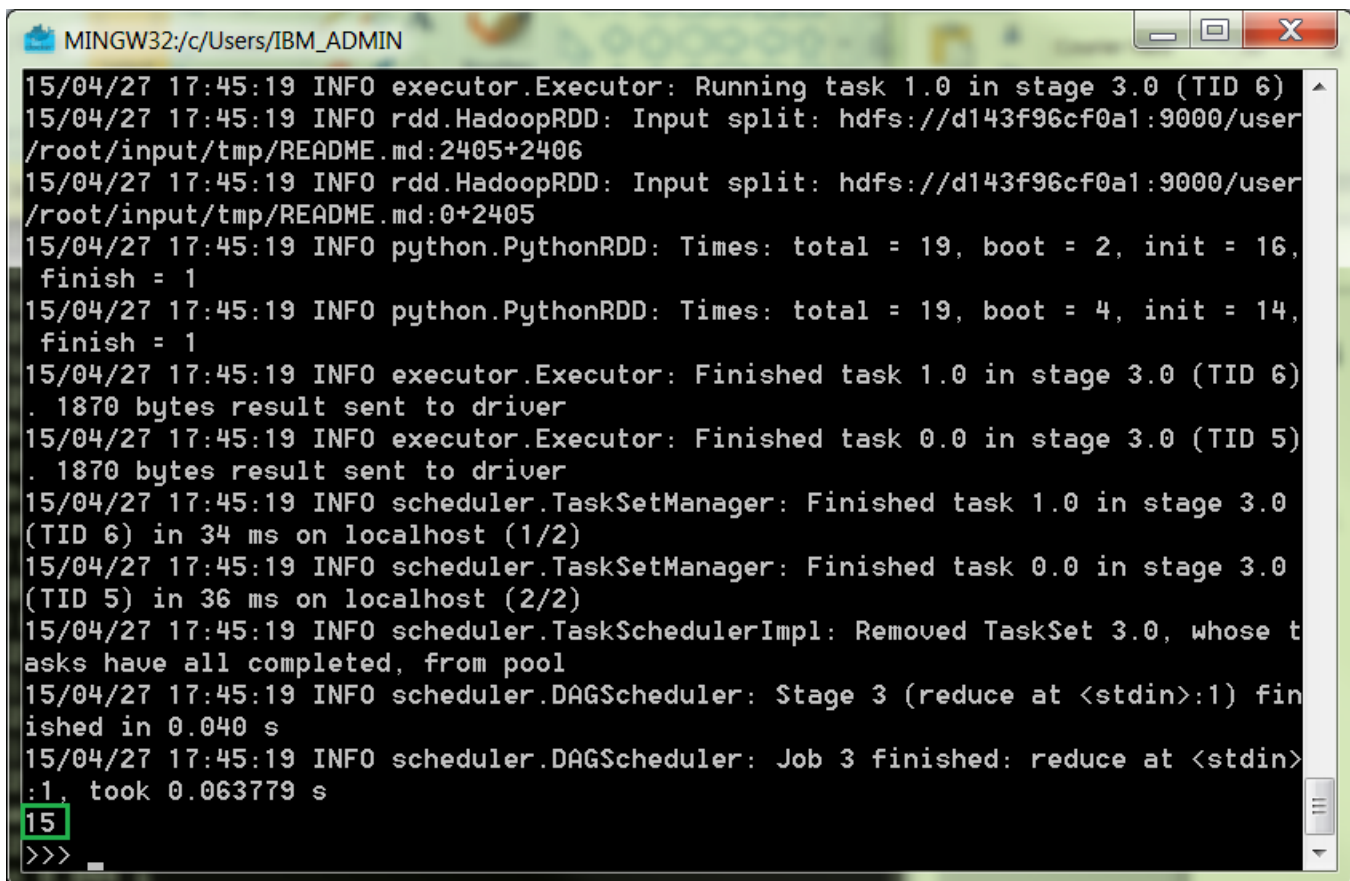
1.5 More on RDD Operations using Python

This section builds upon the previous section. You will have to use the same shell from the previous section. In this section, you will see that RDD can be used for more complex computations. You will find the line from that readme file with the most words in it.

- __1. Using the python shell, copy in:

```
readme.map(lambda line: len(line.split())).reduce(lambda a, b: a if (a > b)
else b)
```

There are two parts to this. The first maps a line to an integer value, the number of words in that line. In the second part reduce is called to find the line with the most words in it. The arguments to *map* and *reduce* are Python anonymous functions (lambdas), but you can use any top level Python functions. In the next step, you'll define a max function to illustrate this feature.



```

MINGW32:/c/Users/IBM_ADMIN
15/04/27 17:45:19 INFO executor.Executor: Running task 1.0 in stage 3.0 (TID 6)
15/04/27 17:45:19 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:2405+2406
15/04/27 17:45:19 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:45:19 INFO python.PythonRDD: Times: total = 19, boot = 2, init = 16,
finish = 1
15/04/27 17:45:19 INFO python.PythonRDD: Times: total = 19, boot = 4, init = 14,
finish = 1
15/04/27 17:45:19 INFO executor.Executor: Finished task 1.0 in stage 3.0 (TID 6)
. 1870 bytes result sent to driver
15/04/27 17:45:19 INFO executor.Executor: Finished task 0.0 in stage 3.0 (TID 5)
. 1870 bytes result sent to driver
15/04/27 17:45:19 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 3.0
(TID 6) in 34 ms on localhost (1/2)
15/04/27 17:45:19 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 3.0
(TID 5) in 36 ms on localhost (2/2)
15/04/27 17:45:19 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose t
asks have all completed, from pool
15/04/27 17:45:19 INFO scheduler.DAGScheduler: Stage 3 (reduce at <stdin>:1) fin
ished in 0.040 s
15/04/27 17:45:19 INFO scheduler.DAGScheduler: Job 3 finished: reduce at <stdin>
:1, took 0.063779 s
15
>>>

```

- __2. Define the max function. You will need to type this in:

```
def max(a, b):
...     if a > b:
```

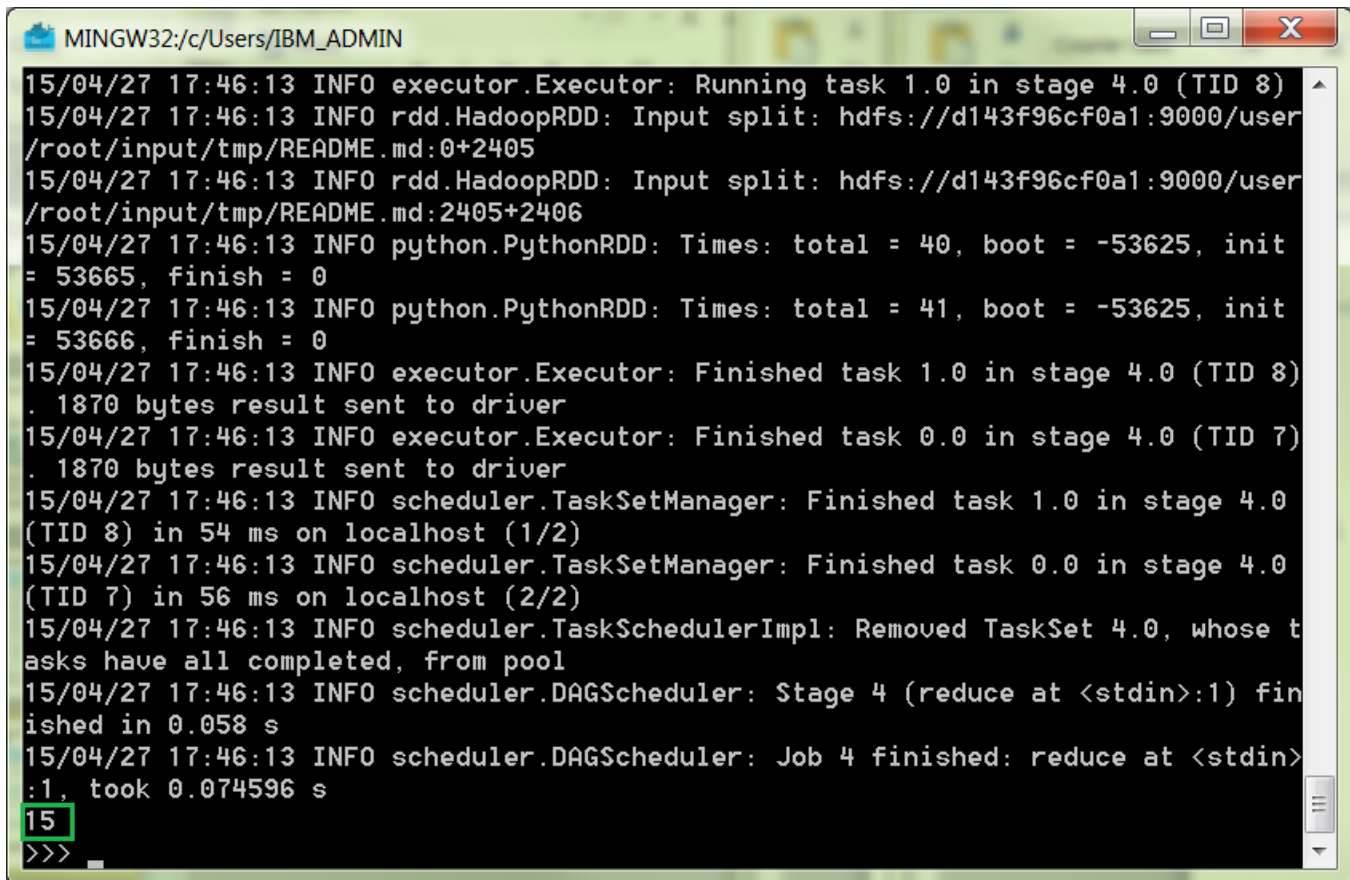
```

...         return a
...     else:
...         return b
...

```

- __3. Now copy in the following to run with the max function:

```
readme.map(lambda line: len(line.split())).reduce(max)
```



```

MINGW32/c/Users/IBM_ADMIN
15/04/27 17:46:13 INFO executor.Executor: Running task 1.0 in stage 4.0 (TID 8)
15/04/27 17:46:13 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:0+2405
15/04/27 17:46:13 INFO rdd.HadoopRDD: Input split: hdfs://d143f96cf0a1:9000/user
/root/input/tmp/README.md:2405+2406
15/04/27 17:46:13 INFO python.PythonRDD: Times: total = 40, boot = -53625, init
= 53665, finish = 0
15/04/27 17:46:13 INFO python.PythonRDD: Times: total = 41, boot = -53625, init
= 53666, finish = 0
15/04/27 17:46:13 INFO executor.Executor: Finished task 1.0 in stage 4.0 (TID 8)
. 1870 bytes result sent to driver
15/04/27 17:46:13 INFO executor.Executor: Finished task 0.0 in stage 4.0 (TID 7)
. 1870 bytes result sent to driver
15/04/27 17:46:13 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 4.0
(TID 8) in 54 ms on localhost (1/2)
15/04/27 17:46:13 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0
(TID 7) in 56 ms on localhost (2/2)
15/04/27 17:46:13 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose t
asks have all completed, from pool
15/04/27 17:46:13 INFO scheduler.DAGScheduler: Stage 4 (reduce at <stdin>:1) fin
ished in 0.058 s
15/04/27 17:46:13 INFO scheduler.DAGScheduler: Job 4 finished: reduce at <stdin>
:1, took 0.074596 s
15
>>>

```

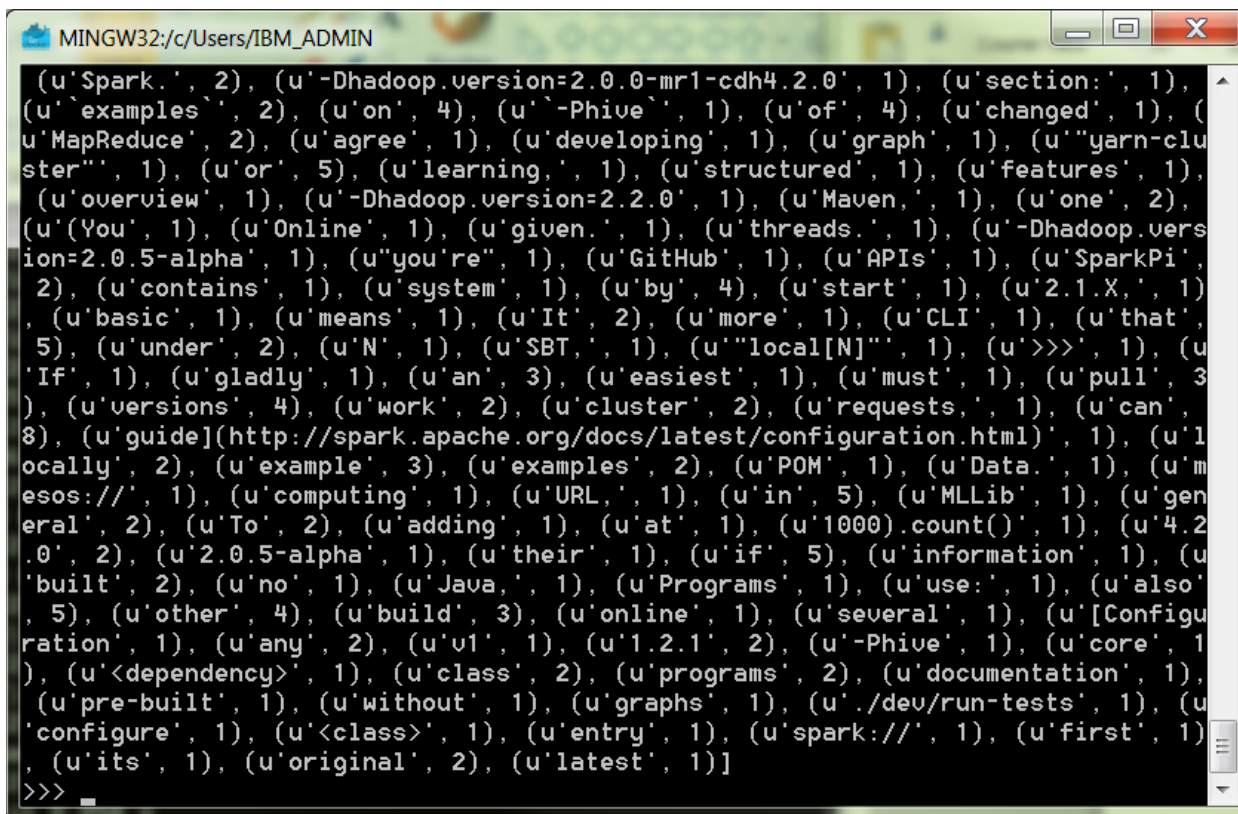
- __4. Spark has a MapReduce data flow pattern. We can use this to do a word count on the readme file. Copy in:

```
wordCounts = readme.flatMap(lambda line: line.split()).map(lambda word: (word,
1)).reduceByKey(lambda a, b: a+b)
```

Here we combined the *flatMap*, *map*, and the *reduceByKey* functions to do a word count of each word in the readme file.

- __5. To collect the word counts, use the *collect* action.

```
wordCounts.collect()
```



```

(u'Spark.', 2), (u'-Dhadoop.version=2.0.0-mr1-cdh4.2.0', 1), (u'section:', 1),
(u'examples', 2), (u'on', 4), (u'-Phive', 1), (u'of', 4), (u'changed', 1), (
u'MapReduce', 2), (u'agree', 1), (u'developing', 1), (u'graph', 1), (u'"yarn-clu
ster"', 1), (u'or', 5), (u'learning', 1), (u'structured', 1), (u'features', 1),
(u'overview', 1), (u'-Dhadoop.version=2.2.0', 1), (u'Maven', 1), (u'one', 2),
(u'You', 1), (u'Online', 1), (u'given.', 1), (u'threads.', 1), (u'-Dhadoop.vers
ion=2.0.5-alpha', 1), (u"you're", 1), (u'GitHub', 1), (u'APIs', 1), (u'SparkPi',
2), (u'contains', 1), (u'system', 1), (u'by', 4), (u'start', 1), (u'2.1.X.', 1)
, (u'basic', 1), (u'means', 1), (u'It', 2), (u'more', 1), (u'CLI', 1), (u'that',
5), (u'under', 2), (u'N', 1), (u'SBT', 1), (u'"local[N]"', 1), (u'>>>', 1), (u
'If', 1), (u'gladly', 1), (u'an', 3), (u'easiest', 1), (u'must', 1), (u'pull', 3
), (u'versions', 4), (u'work', 2), (u'cluster', 2), (u'requests', 1), (u'can',
8), (u'guide](http://spark.apache.org/docs/latest/configuration.html)', 1), (u'l
ocally', 2), (u'example', 3), (u'examples', 2), (u'POM', 1), (u'Data.', 1), (u'm
esos://', 1), (u'computing', 1), (u'URL', 1), (u'in', 5), (u'MLLib', 1), (u'gen
eral', 2), (u'To', 2), (u'adding', 1), (u'at', 1), (u'1000).count()', 1), (u'4.2
.0', 2), (u'2.0.5-alpha', 1), (u'their', 1), (u'if', 5), (u'information', 1), (u
'built', 2), (u'no', 1), (u'Java', 1), (u'Programs', 1), (u'use:', 1), (u'also'
, 5), (u'other', 4), (u'build', 3), (u'online', 1), (u'several', 1), (u'[Configu
ration', 1), (u'any', 2), (u'u1', 1), (u'1.2.1', 2), (u'-Phive', 1), (u'core', 1
), (u'<dependency>', 1), (u'class', 2), (u'programs', 2), (u'documentation', 1),
(u'pre-built', 1), (u'without', 1), (u'graphs', 1), (u'./dev/run-tests', 1), (u
'configure', 1), (u'<class>', 1), (u'entry', 1), (u'spark://', 1), (u'first', 1)
, (u'its', 1), (u'original', 2), (u'latest', 1)]
>>>

```

1.6 Using Spark caching

In this short section, you'll see how Spark caching can be used to pull data sets into a cluster-wide in-memory cache. This is very useful for accessing repeated data, such as querying a small "hot" dataset or when running an iterative algorithm. Both Python and Scala use the same commands, so you can input these into any of the two shells.

- __1. As a simple example, let's mark our *linesWithSpark* dataset to be cached and then invoke the first count operation to tell Spark to cache it. Remember that transformation operations such as `cache` does not get processed until some action like `count()` is called. Once you run the second `count()` operation, you should notice a small increase in speed.

```
linesWithSpark.cache()  
  
linesWithSpark.count()  
  
linesWithSpark.count()
```

It may seem silly to cache such a small file, but for larger data sets across tens or hundreds of nodes, this would still work. The second *linesWithSpark.count()* action runs against the cache and would perform significantly better for large datasets.

- __2. At this time, you can terminate the shells. Use the key combination: CTRL + D

Summary

Having completed this exercise, you should now be able to log in to your environment and use the Spark shell to run simple actions and transformations for Scala and/or Python. You understand that Spark caching can be used to cache large datasets and subsequent operations on it will utilize the data in the cache rather than re-fetching it from HDFS.

NOTES

NOTES

[illegible]



© Copyright IBM Corporation 2015.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.



Please Recycle
