Spark in Scala

Lab 01: Spark Basics Walkthrough

This file gets you started in Spark in Scala.

Turn down the logging level

By default Spark versions before 2.0, many of Spark's logging parameters are set to `INFO` leading to extremely verbose messages printed to the console. In fact, if you don't turn these down, the prompt will soon scroll off the screen!

On Amazon EMR

If you are running a version of Spark before 2.0, edit `/usr/lib/spark/conf/log4j.properties` on the cluster with your text editor of choice (`emacs` , `vi` , `nano`) to selectively change `INFO` entries to `WARN` . To change them all easily:

```
sed -i.bak 's/INFO/WARN/' /usr/lib/spark/conf/log4j.properties
```

On Vagrant

If you are running a version of Spark before 2.0, edit `/vagrant/latest-spark/conf/log4j.properties` on the cluster with your text editor of choice (`emacs` , `vi` , `nano`) to selectively change `INFO` entries to `WARN` . To change them all easily:

```
sed -i.bak 's/INFO/WARN/' /vagrant/latest-spark/conf/log4j.properties
```

Launch Spark's interactive environment

Begin by starting the Spark shell from the Linux Shell command line

```
spark-shell --master yarn-client         // start in client mode
```

At the `scala>` prompt, examine the Spark Context you were handed when you started Spark by typing `sc`. Throughout this lab, we'll show what you type immediately after the `scala>` prompt followed by what the spark-shell returns.

```
scala> sc
res7: org.apache.spark.SparkContext = org.apache.spark.SparkContext@2bc52b08

scala> sc.isLocal
res8: Boolean = false

scala>
```

Note that tab completion works by default in the Scala shell to show which methods and fields are available on the SparkContext sc. Try some of them out to explore the SparkContext:

```
scala> sc.appName
res11: String = Spark shell

scala> sc.getExecutorMemoryStatus
res12: scala.collection.Map[String,(Long, Long)] = Map(172.31.57.176:45326 -> (434582323,43458

scala> sc.hadoopConfiguration
res13: org.apache.hadoop.conf.Configuration = Configuration: core-default.xml, core-site.xml, r

scala> sc.version
res14: String = 2.2.0

scala> spark
res15: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@50aa482f
```

Play with an RDD

Now create a simple, small RDD by hand:

```
val smallprimes = sc.parallelize(Array(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97))
```

Try some basic operations on the RDD. Note that a function takes no arguments, no parentheses are needed after the function name.

```
scala> smallprimes.count
res23: Long = 25

scala> smallprimes.min
res24: Int = 2

scala> smallprimes.max
res25: Int = 97

scala> smallprimes.sum
res26: Double = 1060.0

scala> smallprimes.collect
res27: Array[Int] = Array(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, (
```

Read data from HDFS

A more realistic scenario is to create an RDD from data read from disk. Spark can natively access HDFS and S3 in addition to the local file system. Try reading in the stock quote data from HDFS:

```
val rdd = sc.textFile("hdfs:///data/stocks-flat/input")
rdd.first
rdd.count
```

Simple Transformations and Actions

Let's first filter the data set so we only see AAPL records:

```
val aapl = rdd.filter( line => line.contains("AAPL") )
aapl.count
```