Spark in Scala

Spark-Wordcount

This exercise walks through the classic *Wordcount* example in Spark in both Scala and Python.

Before running this or other examples, you should turn down Spark's default logging. See the the slides in Spark-in-Scala or Spark-in-Python for how to do this.

Run the Spark shell

Scala:

```
$ spark-shell --master yarn-client
```

Python:

```
$ pyspark --master yarn-client
```

Note the Spark context will already be initialized and available to you as `sc`.

Read the data from HDFS

First, let's load in our Shakespeare text and apply a regular expression to split on non-word characters:

Scala:

```
val text = sc.textFile("hdfs:///data/shakespeare/input/")
val words = text.flatMap( line => line.split("\\W+") )
```

Python:

```
text = sc.textFile("hdfs:///data/shakespeare/input/")
words = text.flatMap( lambda line: line.split() )
```

Construct key-value pairs

Generate the "(word, 1)" key-value pairs:

Scala:

```
val kv = words.map( word => (word.toLowerCase(), 1) )
```

Python:

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

Group by key and sum

Use `reduceByKey` to group the data by key and apply the addition operator to add the values:

Scala:

```
val totals = kv.reduceByKey( (v1, v2) => v1 + v2  )
```

Python:

```
totals = kv.reduceByKey( lambda v1, v2: v1 + v2 )
```

Execute the job and save the results

Scala:

```
scala> totals.saveAsTextFile("hdfs:///user/hadoop/spark-wc")

scala> totals.take(10)
res17: Array[(String, Int)] = Array((pinnace,3), (bone,21), (lug,3), (vailing,3), (bombast,4),

scala> words.max()
res18: String = zwaggered
```

Python:

```
>>> totals.saveAsTextFile('hdfs:///user/hadoop/spark-wc-py')

>>> totals.take(10)
[(u'fawn', 14), (u'sending:', 1), (u'mustachio', 1), (u'philadelphos,', 1), (u'protested,', 1)
```

Examine the output on disk

```
$ hadoop fs -cat spark-wc/part-00000 | head -20
(pinnace,3)
(bone,21)
(lug,3)
(vailing,3)
(bombast,4)
(gaping,11)
(hem,10)
(forsooth,48)
(stinks,1)
(been,738)
(fuller,2)
(jade,16)
(countervail,3)
(jove,98)
(crying,36)
(breath,238)
(battering,3)
(contemptible,5)
(swain,24)
(clients,3)
```

Note the lack of sorting. Spark does not perform the same merge-sort as MapReduce during its shuffles.

Data pipeline syntax

Both Scala and Python allow you to chain function calls making it possible to write both Spark jobs much more compactly:

Scala:

```
val text = sc.textFile("hdfs:///data/shakespeare/input/")
val totals = text.flatMap( line => line.split("\\W+") ).map( word => (word.toLowerCase(), 1) )

totals.saveAsTextFile("hdfs:///user/hadoop/spark-wc2")
```

Python:

```
text = sc.textFile("hdfs:///data/shakespeare/input/")
totals = text.flatMap( lambda line: line.split() ).map( lambda word: (word.lower(), 1) ).reduce

totals.saveAsTextFile('hdfs:///user/hadoop/spark-wc-py2')
```