

Project in Spark 2017

Adrianna Janik

Ion Mosnoi

Lei Guo

October 16, 2017

1 TASK

Firstly we uncompressed the data stored in ling-spam.zip folder with *Extract all* command. Secondly we open Virtual Box machine with Hortonworks, we signed in with maria_dev user-name and maria_dev password on Ambari available under 127.0.0.1:8080 ip address. We have selected *Files view*, than navigated to /tmp folder and created directories tmp/ling-spam/ham and ling-spam/spam. Following that we logged in with ssh credentials to Hortonworks machine

```
$ssh root@127.0.0.1 -p 2222
```

In the meantime upload to the virtual machine ling-spam.zip with:

```
$sudo scp -P 2222 ../ling-spam.zip root@127.0.0.1:/tmp/
```

We unzipped ling-spam.zip with:

```
$unzip ling-spam.zip -d /tmp/ling-spam
```

We putted files into /tmp/ling-spam/ folder in hdfs with:

```
$hdfs dfs -put ../ling-spam/ham /tmp/ling-spam/ham
```

```
$hdfs dfs -put ../ling-spam/spam /tmp/ling-spam/spam
```

2 TASK

Installation of sbt:

```
$wget http://dl.bintray.com/sbt/rpm/sbt-0.13.12.rpm
```

Edit file /etc/yum.repos.d/sandbox.repo:

```
~[sandbox]
~name=Sandbox repository (tutorials)
~gpgcheck=0
~enabled=0
~baseurl=http://dev2.hortonworks.com.s3.amazonaws.com/repo/dev/master/Utils/
```

```
$yum clean all
```

```
$yum update
```

```
$sudo yum localinstall sbt-0.13.12.rpm
```

```
$sbt -update
```

```
$sudo scp -P 2222 -r ../spamTopWords/* root@127.0.0.1:/tmp/spamTopWords/
```

```
$sbt package
```

3 TASK

Firstly we created Spark Context with:

```
/* Spark Context */
```

```
object Spark {
    val sc = new SparkContext(new SparkConf().setAppName("test").setMaster("local[*]"))
}
```

Then we called function *probaWordDir* with defined spark context as well as folder name for which we want to count words.

```
val context = Spark.sc // declaration of Spark context
```

```
var files , numFiles = probaWordDir(context)("hdfs:///tmp/ling-spam/ham/ham/")
```

```
// call function with context and folder name and save results to files and
```

```
// numFiles variable containing as follow probabilities of words occurrences and number of
```

```
eta = sc.wholeTextFiles(filesDir).collect().nd{lstlisting}
```

```
\begin{lstlisting}[language=scala]
```

4 TASK

5 TASK

6 SCALA USEFUL FUNCTIONS

- `wholeTextFiles` - lets you read a directory containing multiple small text files, and returns each of them as (filename, content) pairs. This is in contrast with `textFile`, which would return one record per line in each file.

- `map(func)` - return a new distributed dataset formed by passing each element of the source through a function `func`.

- `flatMapValues` -

```
rdd.flatMapValues(x => (x to 5))
```

It is applied on an rdd (1,2),(3,4),(3,6) and the output of the transformation is (1,2),(1,3),(1,4),(1,5),(3,4),(3,5)

`faltMapValues` works on each value associated with key. In above case x to 5 means each value will be incremented till 5.

Taking first pair where you have 1,2 , here key is 1 and value is 2 so there after applying transformation it will become (1,2),(1,3),(1,4),(1,5).

- `mapValues` -

```
val m = Map( 'a' -> 2, 'b' -> 3 )
```

```
// both
```

```
m.mapValues(_ * 5)  
m.transform( (k,v) => v * 5 )  
// deliver the same result.
```

- `filter(func)` - return a new dataset formed by selecting those elements of the source on which `func` returns true.

- `case` -

```
import scala.util.Random
```

```
val x: Int = Random.nextInt(10)
```

```
x match {  
  case 0 => "zero"  
  case 1 => "one"  
  case 2 => "two"  
  case _ => "many"
```

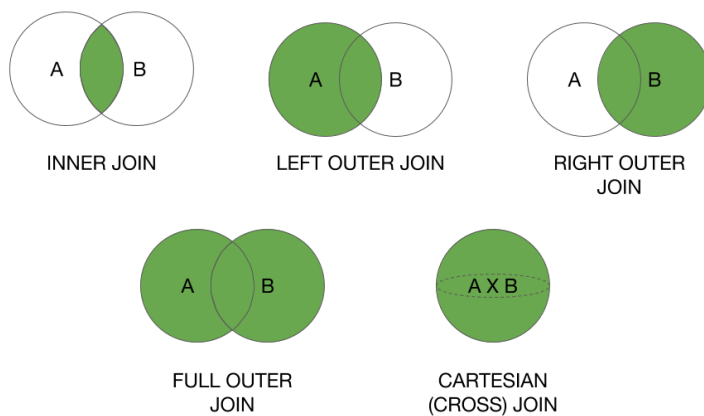
```
}
```

```
def matchTest(x: Int): String = x match {  
  case 1 => "one"  
  case 2 => "two"  
  case _ => "many"  
}  
matchTest(3) // many  
matchTest(1) // one
```

- **reduce** - aggregate the elements of the dataset using a function `func` (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
- **reduceByKey** - when called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function `func`, which must be of type (V,V) => V. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.
- **Ordering.by** - **Ordering** is a trait whose instances each represent a strategy for sorting instances of a type. **Ordering**'s companion object defines many implicit objects to deal with subtypes of `AnyVal` (e.g. `Int`, `Double`), `String`, and others. To sort instances by one or more member variables, you can take advantage of these built-in orderings using **Ordering.by** and **Ordering.on**:

```
import scala.util.Sorting  
val pairs = Array(( 'a', 5, 2), ( 'c', 3, 1), ( 'b', 1, 3))  
  
// sort by 2nd element  
Sorting.quickSort(pairs)(Ordering.by[(String, Int, Int), Int](_. _2))  
  
// sort by the 3rd element, then 1st  
Sorting.quickSort(pairs)(Ordering[(Int, String)].on[  
  (String, Int, Int)]( _. _3, _. _1))
```

- **fullOuterJoin** -
- **join** - when called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through **leftOuterJoin**, **rightOuterJoin**, and **fullOuterJoin**.
- **leftOuterJoin** -
- **getOrElse** -



- **math.log** - The object `Math` contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

def `log (x: Double): Double`

- **toDouble** -

```
scala> "100".toDouble
res1: Double = 100.0
```

- **toSet** -

```
scala> val arr = Array("a", "b", "c")
arr: Array[java.lang.String] = Array(a, b, c)
```

```
scala> arr.toSet
res1: scala.collection.immutable.Set[java.lang.String] = Set(a, b, c)
```

- **takeOrdered** - return the first n elements of the RDD using either their natural order or a custom comparator.
- **foreach** - run a function `func` on each element of the dataset. This is usually done for side effects such as updating an accumulator variable (see below) or interacting with external storage systems.
- **swap** -

```
scala> val pair = (1,2)
pair: (Int, Int) = (1, 2)
```

```
scala> val swappedPair = pair.swap
swappedPair: (Int, Int) = (2, 1)
```