**The 7 Ways to Code WordCount in Spark 2.0**

For illustration purposes, I am using text file that contains the 4 lines of the Humpty Dumpty rhyme.

Humpty Dumpty sat on a wall,

Humpty Dumpty had a great fall.

All the king's horses and all the king's men

Couldn't put Humpty together again.

All examples start by reading the file, then separating the words in each line, filtering out all other words except for the two words Humpty & Dumpty, and last performing the count. In each snippet the result is printed on the console rather than saving it into an hdfs file. The result of the 7 examples is always Dumpty occurring 2 times and Humpty 3 times:

[Dumpty,2]

[Humpty,3]

Each of the snippets illustrates a specific Spark construct or API functionality related to either RDDs, DataFrames, Datasets or Spark SQL.

## Example 1: Classic Word Count using filter & reduceByKey on RDD

val dfsFilename = "/input/humpty.txt"

val readFileRDD = spark.sparkContext.textFile(dfsFilename)

val wcounts1 = readFileRDD.flatMap(line=>line.split(" "))

.filter(w => (w =="Humpty") || (w == "Dumpty"))

.map(word=>(word, 1))

.reduceByKey(\_ + \_)

wcounts1.collect.foreach(println)

In this example each line in the file is read as an entire string into an RDD. Then each line is split into words. The split command generates an array of words for each line. The flatMap command flattens the array and groups them together to produce a long array that has all the words in the file. Then the array is filtered and only the two words are selected. Then each of the two words is mapped into a key/value pair. Last the reduceByKey operation is applied over the key/value pair to count the words’ occurrence in the text.

## Example 2: Word Count Using groupBy on RDD

val dfsFilename = "/input/humpty.txt"

val readFileRDD = spark.sparkContext.textFile(dfsFilename)

val wcounts2 = readFileRDD.flatMap(line=>line.split(" "))

.filter(w => (w =="Humpty") || (w == "Dumpty"))

.groupBy(\_.toString)

.map(ws => (ws.\_1,ws.\_2.size))

wcounts2.collect.foreach(println)

This example is similar to the first example. The two only differ in the usage of groupBy operation which generates a key/value pair that contains the word as a key and a sequence of the same word repeated as a value. Then a new key/value pair is produced that uses the sequence size as a count of the occurrence of the word. It is important to note that the filter function (predicate) is applied on each word and only the words that satisfy the condition are passed to the groupBy operation.

## Example 3: Word Count Using Dataframes, Rows and groupBy

import org.apache.spark.sql.Row

val dfsFilename = "/input/humpty.txt"

val readFileDF = spark.sparkContext.textFile(dfsFilename)

val wordsDF = readFileDF.flatMap(\_.split(" ")).toDF

val wcounts3 = wordsDF.filter(r => (r(0) =="Humpty") || (r(0) == "Dumpty"))

.groupBy("Value")

.count()

wcounts3.collect.foreach(println)

This example is totally different from the first two examples. Here we use DataFrames instead of RDD to work with the text as indicated with the “toDF” command. The returned DataFrame is made of a sequence of Rows; for in Spark 2.0, DataFrames are just Datasets of Rows. Because of the split operation, each row is made of one element, which allows us to access that entry using the field index=0. Also, similar to 2nd example we are using the gourpBy operation which is followed by count to perform the word count. The count command gives DataFrames their edge over RDDs.

If you are wondering how can we use the column name "Value" in the groupBy operation, the reason is simple; when you define a Dataset/DataFrame with one column the Spark Framework on run-time generates a column named "Value" by default if the programmer does not define one. The filter operation above can also be written in another way and the first element in the array within the row can be accessed via “r.getString(0)”.

val wcounts3 = wordsDF.filter(r => (r.getString(0) =="Humpty") || (r.getString(0) == "Dumpty"))

.groupBy("Value")

.count()

## Example 4: Word Count Using Dataset

import spark.implicits.\_

val dfsFilename = "/input/humpty.txt"

val readFileDS = spark.read.textFile(dfsFilename)

val wcounts4 = readFileDS.flatMap(\_.split(" "))

.filter(w => (w =="Humpty") || (w == "Dumpty"))

.groupBy("Value")

.count()

wcounts4.show()

We use in this example Datasets instead of DataFrames to read the text file then we apply a filter and groupBy operation followed by count. The code here is easy to read and very intuitive; it blends well with the Scala paradigm.

## Example 5: Word Count Using Spark SQL on Dataset & TempView

import spark.implicits.\_

val dfsFilename = "/input/humpty.txt"

val readFileDS = spark.sqlContext.read.textFile(dfsFilename)

val wordsDS = readFileDS.flatMap(\_.split(" ")).as[String]

wordsDS.createOrReplaceTempView("WORDS")

val wcounts5 = spark.sql("SELECT Value, COUNT(Value) FROM WORDS WHERE Value ='Humpty' OR Value ='Dumpty' GROUP BY Value")

wcounts5.show

Here we create a Temporary View called WORDS that we query using a standard Spark SQL Select statement.

## Example 6: Word Count Using Case Class, Dataset and Value Attribute

case class CWord (Value: String)

import spark.implicits.\_

val dfsFilename = "/input/humpty.txt"

val readFileDS = spark.sqlContext.read.textFile(dfsFilename).flatMap(\_.split(" "))

val CWordsDS = readFileDS.as[CWord]

val wcounts6 = CWordsDS.filter (w => (w.Value == "Humpty") || (w.Value == "Dumpty"))

.groupBy("Value")

.count()

wcounts6.collect.foreach(println)

n this example we utilize the power of the "strongly typed" Datasets by providing the schema as a case classs using the "as" operator. Thus, instead of using type agnostic Rows, we use Scala’s case classes to describe the row contents and access the attribute. Note that the case class CWord has a column name 'Value'. Using any other name will cause a run-time error in Spark because your CWord column name should be identical to the default column name 'Value' generated by Spark on run-time when reading the text file.

In case you wish to use the “where” operator with an SQL style predicate you can define the query clause as a double quoted string as follows.

val wcounts6 = CWordsDS.where("Value = 'Humpty' OR Value = 'Dumpty'")

.groupBy("Value")

.count()

## Example 7: Word Count Using Case Class, Dataset, agg operation and $column-name

case class CWord (Value: String)

import spark.implicits.\_

val dfsFilename = "/input/humpty.txt"

val readFileDS = spark.sqlContext.read.textFile(dfsFilename).flatMap(\_.split(" "))

val CWordsDS = readFileDS.as[CWord]

val wcounts7 = CWordsDS.where( ($"Value" === "Humpty") || ($"Value" === "Dumpty") )

.groupBy($"Value")

.agg(count($"Value"))

wcounts7.collect.foreach(println)

This example is similar to the previous one, however instead of a string predicate in the “where” clause we use the '$' symbol to indicate a column name. We also use the agg command which is the generalized operation for all aggregate functions in Spark. The above can also be written by replacing the $"Value" with CWordsDS("Value") as follows:

val wcounts7 = CWordsDS.where( (CWordsDS($"Value") === "Humpty") || ((CWordsDS($"Value") === "Dumpty") )

.groupBy((CWordsDS($"Value"))

.agg(count((CWordsDS($"Value"))