Lab - 3

Objectives:

1. Introduction of *divide-and-conquer* using Spark's DAG execution model.

Instructions:

1. Start your VM and open the Zeppelin Notebook Web UI.
2. Create a new note preferably named **Lab 3**.

Word Count:

Using RDD:

1. First, we use `**SparkContext**` to read a text file (***lorem_ipsum.txt***), like so:

```
val paragraphs = sc.textFile("/path/to/file")
```

2. The previous step will return a RDD containing the text file. So, each element of the RDD will be having one paragraph of the file. The following code snippet will give you an idea:

```
// to view data prefer take() over show()
paragraphs.take(1)
// Output:
// Array[String] = Array(Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Phasellus sed velit eget turpis aliquam varius.
Phasellus iaculis eget nibh at pharetra. Morbi lectus magna, tincidunt
eget augue in, volutpat ullamcorper lacus. Aenean sollicitudin sapien
at nisi sodales, eget maximus eros maximus. Vivamus ac est molestie,
facilisis lacus non, interdum felis. Vestibulum lobortis tristique
pretium. Praesent ornare eros in vulputate cursus. Integer sit amet
pharetra diam, vitae semper ligula. Aliquam scelerisque magna interdum
purus efficitur aliquet. Aliquam commodo turpis nunc, vel imperdiet ex
pharetra semper. Pellentesque posuere pretium magna eget lacinia.)
```

3. The following introduces does a word count of the file:

```
// flatMap -> transforms a 2D object to 1D object
// map -> maps a function to every element of a RDD
// stripSuffix -> a Scala String method to remove a suffix
// reduceByKey -> reduces the contents of values of a key-value pair
RDD
val wordCount = paragraphs.flatMap(line => line.split(" "))
               .map(word => (word.stripSuffix(".").stripSuffix(","), 1))
```

```
                .reduceByKey((a, b) => a + b)
// collects all the results at the driver node
wordCount.collect()
```

4. Try to get the word with the highest count. *(Hint: Have a look at `swap` method for Scala Tuple and `sortByKey` in RDD API)*

Using Dataset: (supposed to be slower than RDD due to groupByKey)

```
val linesDataset = spark.read.textFile("/path/to/file").as[String]
val wordCount = linesDataset
.flatMap(_.split(" "))
.map(_.stripSuffix(".").stripSuffix(","))
.groupByKey(word => word)
.count
.collect()
```

Try doing the Word Count example using PySpark.

Alternative procedure for analysis of `**trucks.csv**`:

The previous week used an PySpark to query and visualize the data. Since, by now you are familiar with the basic concepts and abstractions of Spark, we are gonna perform the same analysis using Scala. We are only going to introduce you to the basics of Scala so that you are good to go for using Spark. If you feel adventurous, explore some more ways to perform the same task. Once, you are comfortable with this, we are gonna move to some comparably large dataset.

1. First, we are gonna read the file using `**SparkSession.DataFrameReader**` so that we get a DataFrame, like so:

```
// val defines a constant in Scala
val trucksDf = spark.read.option("header",
"true").csv("/path/to/trucks.csv")
```

2. Then we view all of the column names, using:

```
trucksDf.columns

// Output:
// Array[String] = Array(driverid, truckid, model, jun13_miles,
jun13_gas, may13_miles, may13_gas, apr13_miles, apr13_gas ...
```

3. We can run a simple query to view the *driverid* and *jun13_miles* from *trucksDf*, like so:

```
trucksDf.select("driverid", "jun13_miles").show(15)
```

```
// Output:
//+--------+-----------+
//|driverid|jun13_miles|
//+--------+-----------+
//|      A1|       9217|
//|      A2|      12058|
//|      A3|      13652|
//|      A4|      12687|
//|      A5|      10233|
//|      A6|      14488|
//|      A7|      10938|
//|      A8|      11392|
//|      A9|      12601|
//|     A10|      13699|
//|     A11|      12447|
//|     A12|      10006|
//|     A13|       9740|
//|     A14|      10608|
//|     A15|       9650|
//+--------+-----------+
```

4. To visualize the same chart like the previous lab, we run:

```
// first we generate a TempView from DataFrame, to perform SQL queries
trucksDf.createTempView("trucks")

// then we perform the SQL query to get plot
%sql
SELECT jun13_miles, jun13_gas
FROM trucks
```

Analyzing `**omniture-logs.tsv**`:

Next, we are going to look at a rather large dataset (~ 70 MB). The dataset is a log stash containing purchase history (of customers in USA) for a particular e-commerce company.

This section requires you to load the dataset and analyze using whichever method you find suitable. After analyzing the dataset, you need to answer a few questions.

- Which state has the highest number of purchases?
- Which city of that state has the highest purchase density?

Report:

1) Analyze omniture logs and answer the above 2 questions (attach screenshots of code)
2) Visit ip-address:18081. Look at the spark UI for any of the jobs. How many worker nodes are present and how is the data/processing split between them? (attach screenshots of the same)