

# Lab - 2

## Objectives:

1. Introduction to Virtualized Storage. How does HDFS work?
2. Distributed resource management using Apache Ambari.
3. Loading data into HDFS.
4. Visualizing loaded data in Zeppelin Notebook, using Spark SQL.

## Instructions:

Downloading required files:

1. Download the files from the following URLs
  - a. <https://github.com/synchon/IITM-CS4830/raw/master/lab2/omniture-logs.tsv>
  - b. [https://raw.githubusercontent.com/synchon/IITM-CS4830/master/lab2/season-1\\_213\\_csv.csv](https://raw.githubusercontent.com/synchon/IITM-CS4830/master/lab2/season-1_213_csv.csv)
  - c. <https://raw.githubusercontent.com/synchon/IITM-CS4830/master/lab2/trucks.csv>
2. SSH into your VM.
3. Once done, you need to add **127.0.0.1** as an entry to the **/etc/hosts** file, like so:

```
echo '127.0.0.1 sandbox-hdp.hortonworks.com' | sudo tee -a /etc/hosts
```

Loading data into HDFS:

1. In the **Ambari Services** (in the left sidebar), click on **HDFS**.
2. Access the Hadoop Dashboard on your browser at: **<public-ip>:50070** (change the IP address to the public IP of the VM).
3. On the navbar, click on **Utilities** and then select **Browse the file system**.
4. Play around with the interface and get yourself familiar with files residing in the HDFS.

Visualizing data using Zeppelin Notebook and Spark SQL:

1. Note down the metrics like number of blocks, under-replicated blocks/ namenode Heap etc...
2. From the **Ambari Services**, click on **blue icon** (shown below) and open File-view



3. Upload each of the files using the interface to the “/tmp/data” directory (create a folder called /data and upload each of the files)
4. Note down the change in metrics like block size etc.. (explain these changes in the report)
5. To open Zeppelin Notebook, open your browser and access : **<public-ip>:9995** (change the IP address to the public IP of the VM).
6. Click **Create new note** and name the note as **Lab 2**. Leave the **Default Interpreter** as **spark2**.

Below is an example of scala code that reads data from hdfs and displays the same

The Zeppelin Notebook interface shows a notebook titled 'test1'. The code cell contains the following Scala code:

```
val hiveContext = new org.apache.spark.sql.SparkSession.Builder().getOrCreate()
val riskFactorDataFrame = spark.read.format("csv").option("header", "true").load("hdfs:///tmp/data/trucks.csv")
riskFactorDataFrame.createOrReplaceTempView("trucks")
```

The output shows the SparkSession and DataFrame creation details:

```
hiveContext: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@6fc4c743
riskFactorDataFrame: org.apache.spark.sql.DataFrame = [driverid: string, truckid: string ... 109 more fields]
```

Below the output, it states: "Took 1 sec. Last updated by anonymous at January 31 2019, 12:12:24 PM."

The Zeppelin Notebook interface shows a notebook titled 'test1'. The code cell contains the following SQL query:

```
val ans = hiveContext.sql("SELECT driverid, jun13_miles FROM trucks LIMIT 15");
ans.show();
```

The output displays a table with 2 columns: driverid and jun13\_miles. The data is as follows:

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

The below snapshot reads data in Pyspark and visualizes the same consequently

The Zeppelin Notebook interface shows a notebook titled 'test1'. The code cell contains the following Pyspark code:

```
%pyspark
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

dfObj = spark.read.csv("hdfs:///tmp/data/trucks.csv", header=True, mode="DROPMALFORMED")
dfObj.columns

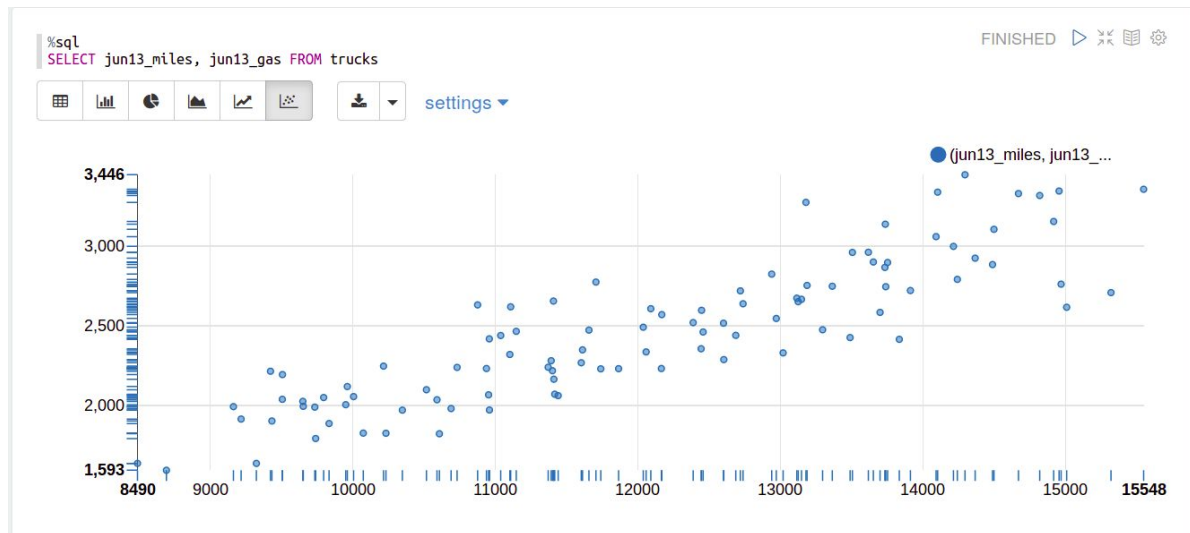
['driverid', 'truckid', 'model', 'jun13_miles', 'jun13_gas', 'may13_miles', 'may13_gas', 'apr13_miles', 'apr13_gas', 'mar13_miles', 'mar13_gas', 'feb13_miles', 'feb13_gas', 'jan13_miles', 'jan13_gas', 'dec12_miles', 'dec12_gas', 'nov12_miles', 'nov12_gas', 'oct12_miles', 'oct12_gas', 'sep12_miles', 'sep12_gas', 'aug12_miles', 'aug12_gas', 'jul12_miles', 'jul12_gas', 'jun12_miles', 'jun12_gas', 'may12_miles', 'may12_gas', 'apr12_miles', 'apr12_gas', 'mar12_miles', 'mar12_gas', 'feb12_miles', 'feb12_gas', 'jan12_miles', 'jan12_gas', 'dec11_miles', 'dec11_gas', 'nov11_miles', 'nov11_gas', 'oct11_miles', 'oct11_gas', 'sep11_miles', 'sep11_gas', 'aug11_miles', 'aug11_gas', 'jul11_miles', 'jul11_gas', 'jun11_miles', 'jun11_gas', 'may11_miles', 'may11_gas', 'apr11_miles', 'apr11_gas', 'mar11_miles', 'mar11_gas', 'feb11_miles', 'feb11_gas', 'jan11_miles', 'jan11_gas', 'dec10_miles', 'dec10_gas', 'nov10_miles', 'nov10_gas', 'oct10_miles', 'oct10_gas', 'sep10_miles', 'sep10_gas', 'aug10_miles', 'aug10_gas', 'jul10_miles', 'jul10_gas', 'jun10_miles', 'jun10_gas', 'may10_miles', 'may10_gas', 'apr10_miles', 'apr10_gas', 'mar10_miles', 'mar10_gas', 'feb10_miles', 'feb10_gas', 'jan10_miles', 'jan10_gas', 'dec09_miles', 'dec09_gas', 'nov09_miles', 'nov09_gas', 'oct09_miles', 'oct09_gas', 'sep09_miles', 'sep09_gas', 'aug09_miles', 'aug09_gas', 'jul09_miles', 'jul09_gas', 'jun09_miles', 'jun09_gas', 'may09_miles', 'may09_gas', 'apr09_miles', 'apr09_gas', 'mar09_miles', 'mar09_gas', 'feb09_miles', 'feb09_gas', 'jan09_miles', 'jan09_gas']
```

The output shows the list of columns in the DataFrame. Below the output, it states: "Took 1 sec. Last updated by anonymous at February 04 2019, 12:17:09 PM."

The second code cell contains the following Pyspark code:

```
%pyspark
from pyspark.sql.types import DoubleType
changedTypedfTmp = dfObj.withColumn("jun13_miles", dfObj["jun13_miles"].cast(DoubleType()))
changedTypedf = changedTypedfTmp.withColumn("jun13_gas", dfObj["jun13_gas"].cast(DoubleType()))
changedTypedf.registerTempTable("trucks")
```

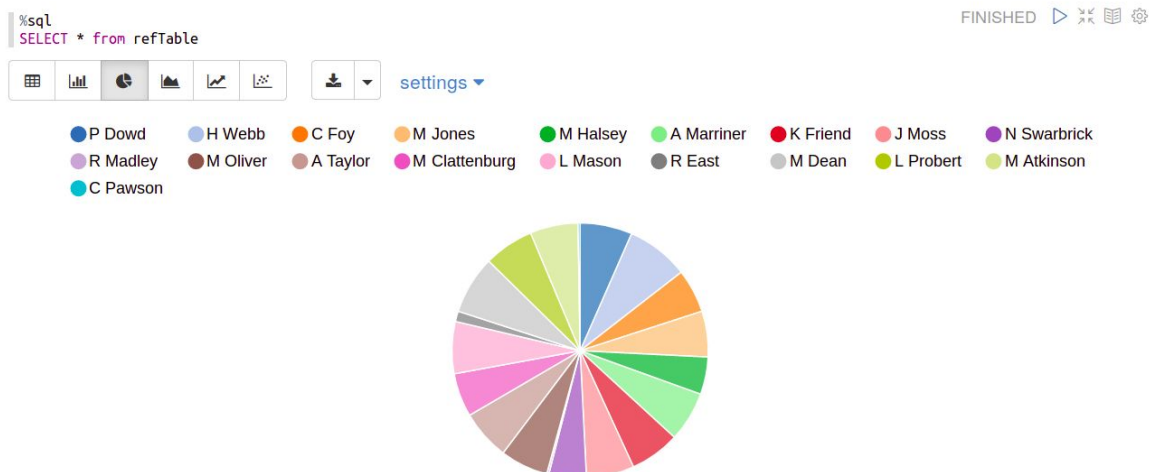
The output shows the result of the type casting and table registration. Below the output, it states: "Took 0 sec. Last updated by anonymous at February 04 2019, 12:17:32 PM."



The above code is also available at:

<https://github.com/synchon/IITM-CS4830/blob/master/lab2/codeSample.md>

- 7 ) Now read the data present in the **season-1213\_csv.csv** from the HDFS. Understand the data and attempt to generate the following plot (For those unfamiliar with SQL have look at the groupby command). This plot describes the number of matches officiated by each referee



In the report, explain how the computations used to generate the above table is executed in the map-reduce framework (what are the map/reduce operations)

Questions to answer:

1. What is the relation of HDFS to CAP theorem
2. We saw Hive,Pig are one of the many services that deals with Yarn. What is the function of these two services
3. Note down metrics like (number of blocks, heap size, under-replicated blocks). Explain the change in metrics after uploading each of the files.
4. Describe the task (of referee vs matches officiated) as a map-reduce task
5. Attach code snippets/plot screenshots of the completed Zeppelin task