CSC 343, Final Project
Spring 2019

This course has covered various aspects of the ***Hadoop*** ecosystem, including **Map Reduce**, **Hive/Impala**, and **Spark**, which are frameworks or tools for processing and analyzing large distributed datasets stored on HDFS. For your Final Project, you will select one of these frameworks or tools to either analyze a dataset of your choice or to gain experience with a new technology.

If analyzing a dataset, you can pick any dataset that is of interest to you, as long as it allows you to answer interesting questions. Many datasets are available from the links below:

1. https://www.kaggle.com/datasets
2. http://rs.io/100-interesting-data-sets-for-statistics/

Let me know if you need help finding a dataset.

1. Use the **Oozie workflow** schedule system to chain two ***Map Reduce*** jobs using Hadoop Streaming. The first job counts the total number of words in a directory, for words that are at least 4 characters long, and the second job sorts these words from most common to least common, with your final output in the form *word*: *count*. To set up an Oozie workflow, the easiest option is to use Hue, and select Workflows → Editors → Workflows and then the Create button to create a new workflow. Note that the jar file to run Map Reduce is */usr/jars/hadoop-streaming-2.6.0-cdh5.7.0.jar*, and this needs to be moved to HDFS. For this option, you will turn in a docker image appropriate configured so that the workflow can be executed.

2. Analyze a dataset of your choice using **Impala**. The dataset must contain at least one *partitioned* table, at least two aggregate queries (e.g., *count* or *average*) over all records, and at least two aggregate queries over different groups. For this option, you will turn in the following: 1) a docker image with data uploaded and a script containing queries that can be executed using the Impala shell (*impala shell -f queries.sql*), 2) the Data Definition Language (DDL) script, which contains SQL for creating all tables, 3) a report, approximately 1 page, that describes the dataset used, the structure of the database, and what you have learned from your analysis of it. Information about partitioning: https://www.cloudera.com/documentation/enterprise/latest/topics/impala_partitioning.html

3. Analyze a dataset of your choice using **PySpark**. The dataset must contain at least two 'tables', and your analysis must contain at least two aggregate transformations using a pair RDD. Persistence should be used so that the same RDD is not created more than once. Your code must also generate at least two graphs, which is perhaps most easily accomplished using the *pandas* module. Pandas Data Frames can be used to turn a dictionary into a data frame (a table), where the keys of the dictionary become columns of the table. For barplot examples, see: https://tinyurl.com/y2gzglrd. For this option, you will turn in the following: (1) a zip file of your data, (2) your PySpark script, and (3) a report, at least one page long, that describes the dataset used and what you have learned from your analysis of it, including graphs generated from your analysis.

4. Complete Lab #10 using **Scala**, which is an object-oriented and functional programming language that is commonly used with Spark. Scala runs on the Java Virtual Machine (JVM), and compared to PySpark is more efficient and safer (https://www.kdnuggets.com/2018/05/apache-spark-python-scala.html) . A brief Scala tutorial is available here: https://docs.scala-lang.org/tutorials/tour/basics.html.html

a. You can access Scala from the gdancik/cloudera image by typing *spark-shell* at the command line. This will open the Scala shell that includes a SparkContext object named *sc*.
b. *Scala* distinguishes between *values* and *variables*. Values are immutable and cannot be re-assigned; variables are not immutable and can be reassigned (see tutorial for details)
c. Anonymous functions are defined using the following syntax:
    - Python syntax: `lambda x: x + 1`
    - Scala syntax: `(x:Int) => x + 1`

d. Spark operations are generally the same between PySpark and Scala, as demonstrated in the example below:

```
// create a list
val l = List(1,2,3,4)

// create RDD from list and apply map to increase values by 1
val rdd = sc.parallelize(l).map((x:Int) => x + 1)

// return RDD elements as an Array
rdd.collect()
```

5. You may choose another specialized Hadoop- or Big Data-related analysis, with my approval.

Additional Information
See the accompanying rubric for additional information.