Project 3 **CS585/DS 503: Big Data Management** **Spring 2019**

"Working with Spark, Scala and SparkSQL"

Total Points: 100 points
Given Out: Monday, 18th March, 2019
Due Date: Saturday, 30th March, 2019 (5:00PM)

Teams: Project is to be done in teams -- Project3-team.
Team members will be assigned by CS585/DS503 staff.

Project Overview

In this project, you will write Scala and SparkSQL code for executing jobs on the Spark infrastructure. You can either work on a *Cluster Mode* (files are read/written over HDFS) or a *Local Mode* (files are read/written over the local file system). In either case, your code should be designed with a high degree of parallelization.

Project Resources

Refer to Spark resources at: *https://spark.apache.org/docs/latest/quick-start.html*
And online book "learning spark: lightning-fast big data analysis" by O'Reilly publisher (pdf)/

Project Submission

You will submit below as one **single zip file** via CANVAS, namely:

1. Submit a file containing all your code from **Scala, SparkSQL, etc**. This includes a **report (.pdf or .doc)** containing all required documentation and explanations of your solutions.
2. Provide **clear install and execution instructions**, along with data sets used (a sample of the data files, if too large, or the data generator). If your install script and environment description are not clear and the TA cannot run your code, there may be some point deductions – or in the best case some delay in grading your project.
3. Indicate the **skills of each team member prior to this project, skills acquired while working on this project, and the relative contributions**[1] of each team member to this project in your report. Team members agree to a joint statement that is submitted as a part of the above project report – and that reflects the division of team effort.
4. E**ach of you will independently** submit your peer team comments to CS585/DS503 staff via https://goo.gl/forms/ip0Yw7YbSDG5sXAe2. This is your personal assessment of your own contributions and effort as well as the contributions of your team member to this project. *These comments will be treated confidentially. You will not be given a grade, until you submit your survey. You are also invited to talk to the CS585/DS503 about your team and/or project, as needed.*

[1] For instance, if each team member has done the project independently, and then only at the end you pulled the best of the material together, you need to say so. If you have closely collaborated and done the same amount of effort working side by side, report this.

Problem 1 SparkSQL for Processing Purchase Transactions [50 points]

Write a program that creates two datasets (files): ***Customers*** and ***Purchases***. Each line in *Customers* file represents one customer, and each line in *Purchases* file represents one purchase transaction. The attributes within each line are comma separated.

The ***Customers*** C dataset should have the following attributes for each customer:

ID: unique sequential number (integer) from 1 to 50,000 *(50,000 lines)*
Name: random sequence of characters of length between 10 and 20 *(careful:no commas)*
Age: random number (integer) between 10 to 70
CountryCode: random number (integer) between 1 and 100
Salary: random number (float) between 100 and 10,000,000

The ***Purchases*** P dataset should have the following attributes for each purchase transaction:

TransID: unique sequential number (integer) from 1 to 5,000,000 (5M purchases)
CustID: References one of the customer IDs, i.e., on Avg. a customer has 100 transactions.
TransTotal: Purchase amount as random number (float) between 10 and 1000
TransNumItems: Number of items as random number (integer) between 1 and 10
TransDesc: Text of characters of length between 20 and 50 *(careful: no commas)*

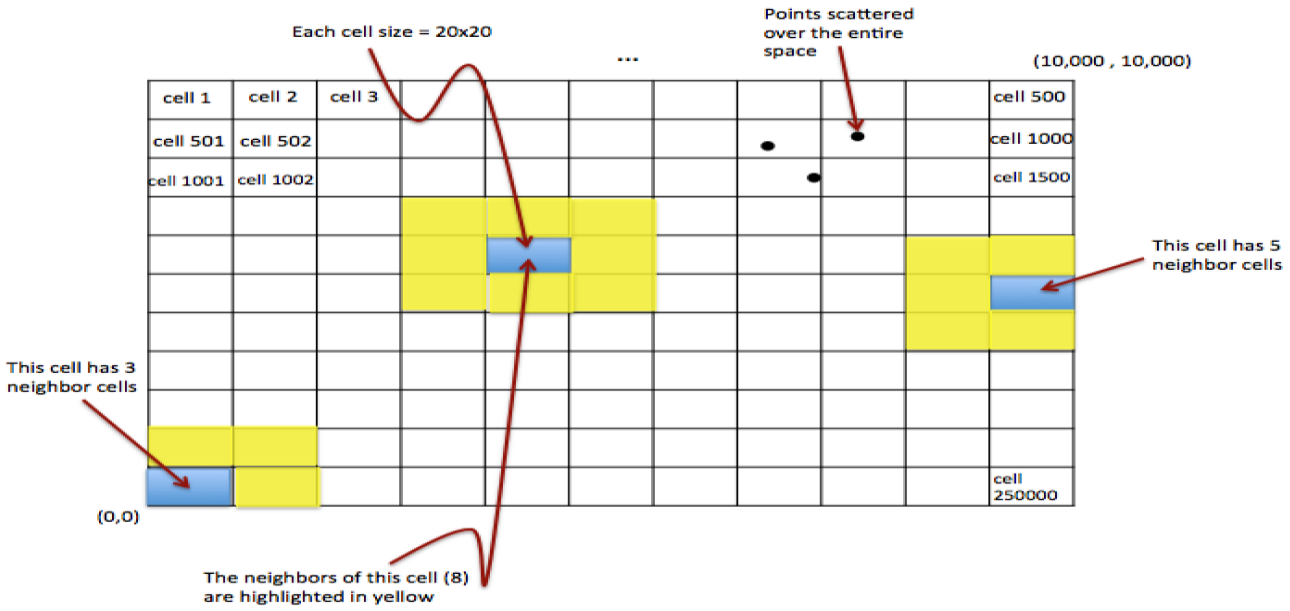
Note: The column names will NOT be stored in the file. From the order of the columns, you will know what each column represents. The values in each line are comma separated; thus do not use a comma inside of each value, such as the description of the transaction, etc.

First, load your data into your storage.
Then use SparkSQL to express the following workflows:

1) T1: Filter out (drop) the Purchases from **P** with a total purchase amount above $500
2) T2: Over T1, group the Purchases by the Number of Items purchased, and for each group calculate the average, min and max of total amount spent for purchases in that group.
3) Report back T2 to the client side.
4) T3: Over T1, group the Purchases from P by customer ID for customers between 10 and 20 years of age, and for each group report the customer ID, their age, and total number of items that this person has purchased and total amount she spent.
5) T4: Select all customer pairs IDs C1 and C2 from T3 where C1 is younger in age than customer C2 but C1 spent in total more money than C2 but bought less items.
6) Report back T4 (C1 ID, C2 ID, Age1, Age2, TotalAmount1, TotalAmount2, TotalItemCount1,TotalItemCount2) to the client side

Problem 2: Spark-RDDs: Scala For Computing Relative-Density Scores at Scale [50 points]

Overview: Assume a two-dimensional space that extends from 1…10,000 in each dimension as shown in Figure 1. There are points scattered across the space. The space is divided into pre-defined grid-cells, each of size 20x20. That is, there are 500,000 grid cells. Each cell has a unique ID as the figure. Given an ID of a grid cell, you can calculate the row and the column it belongs to.



Neighbor Definition: For a given grid cell X, N(X) is the set of all neighbor cells of X, which are the cells with which X has a common edge or corner. The figure illustrates examples of neighbors. Each non-boundary grid cell has 8 neighbors. A boundary cell will have a smaller number of neighbors. Since the grid cell size is fixed, the IDs of the neighbor cells of a given cell can be computed.

Example:
N(Cell 1) = {Cell 2, Cell 501, Cell 502}
N(Cell 1002) = {Cell 501, Cell 502, Cell 503, Cell 1001, Cell 1003, Cell 1501, Cell 1502, Cell 1503}

Relative-Density Score: For a grid cell X, I(X) denotes a decimal number that indicates the relative density of cell X compared to that of its neighbors. It is calculated as follows:

$$I(X) = X.count / Average (Y_1.count, Y_2.count, \ldots Y_n.count)$$

where "X.count" means the count of points inside the grid cell X, and $\{Y_1, Y_2, \ldots, Y_n\}$ refers to the set of all neighbor cells of cell X, that is $N(X) = \{Y_1, Y_2, \ldots, Y_n\}$.

Problem P2.A: Create the Datasets.

You can re-use your code from Project 2 for this task P2.A.

First, you create a dataset *P* (set of 2D points) for problem 2. Assume the space extends from 1…10,000 in the both the X and the Y axis. Each line in the file should contain one point in the format (*a*, *b*), where *a* is the value in the X axis, and *b* denotes the value in the Y axis. Choose a random function to create the points. Then you scale this dataset to make it a big dataset of 100MB. Lastly, you store the file into HDFS.

Problem P2.B: Report the TOP 10 grid cells with highest Relative-Density Scores [30 Pts]

In this task, report the top k grid cells X that have the highest **I** (X) score with k=10. You should return these 10 grid cell IDs and their relative-density score. Thus the output file itself is small. You need NOT return the points inside the grid cell.
This problem must be done in two languages, once in (1) Scala and once in (2) python or java, as you prefer. Your code should be fully distributed and scalable! Report how long your job runs.

Problem P2.C: Report the TOP k grid cells w.r.t their Relative-Density Scores [20 Pts]

In this second task, you use the results output by the task P2.B above, namely, the list of reported top 10 grid cells. For each of these reported top 10 grid cells C1, now compute their neighbor cells, and then return the ID of the grid cell C1, its density-score, its number of neighbors, and the cell IDs and relative-density scores of each of its direct neighbor cells.
This problem must be done in two languages, once in (1) Scala and once in (2) python or java. Your code should be fully distributed and scalable! Also report how long your job runs.

- The end –