CS585: Big Data Management

Project 3

Total Points: 130

Release Date: 10/22/2019

Due Date: 11/5/2019 (11:59PM)

Teams: Project to be done in teams of two.

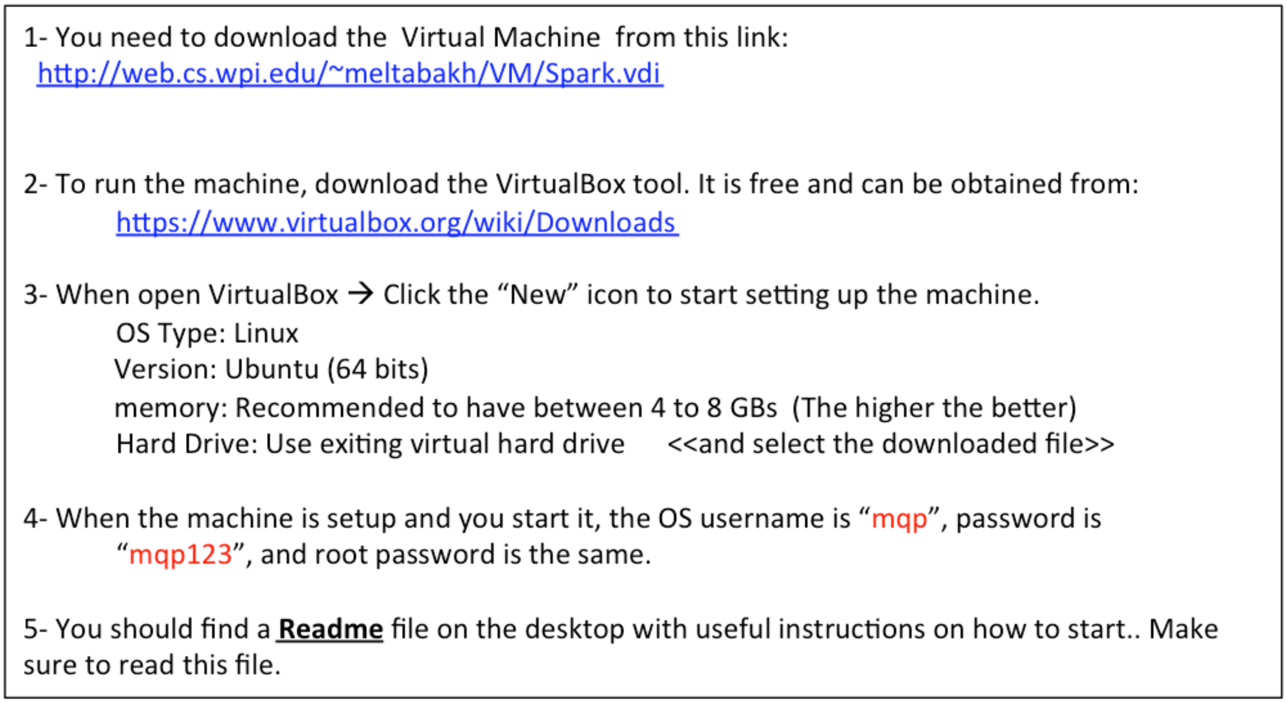
Short Description

In this project, you will write Scala/SparkSQL code for executing jobs over Spark infrastructure.

You can either work on a *Cluster Mode* (where files are read/written over HDFS), or *Local Mode* (where files are read/written over local file system). This should not affect your code design, i.e., in either case your code should be designed with the highest degree of parallelization possible.

Spark Virtual Machine

You can either build your own VM or work with the one provided to you (See below).



1- You need to download the Virtual Machine from this link:
http://web.cs.wpi.edu/~meltabakh/VM/Spark.vdi

2- To run the machine, download the VirtualBox tool. It is free and can be obtained from:
https://www.virtualbox.org/wiki/Downloads

3- When open VirtualBox → Click the "New" icon to start setting up the machine.
OS Type: Linux
Version: Ubuntu (64 bits)
memory: Recommended to have between 4 to 8 GBs (The higher the better)
Hard Drive: Use exiting virtual hard drive <<and select the downloaded file>>

4- When the machine is setup and you start it, the OS username is "mqp", password is "mqp123", and root password is the same.

5- You should find a **Readme** file on the desktop with useful instructions on how to start.. Make sure to read this file.

Problem 1 SparkSQL (Transaction Data Processing) [30 points]

Use the Transaction dataset **T** that you created in Project 1 and create a Spark workflow to do the following. [Use ***SparkSQL*** to write this workflow.]

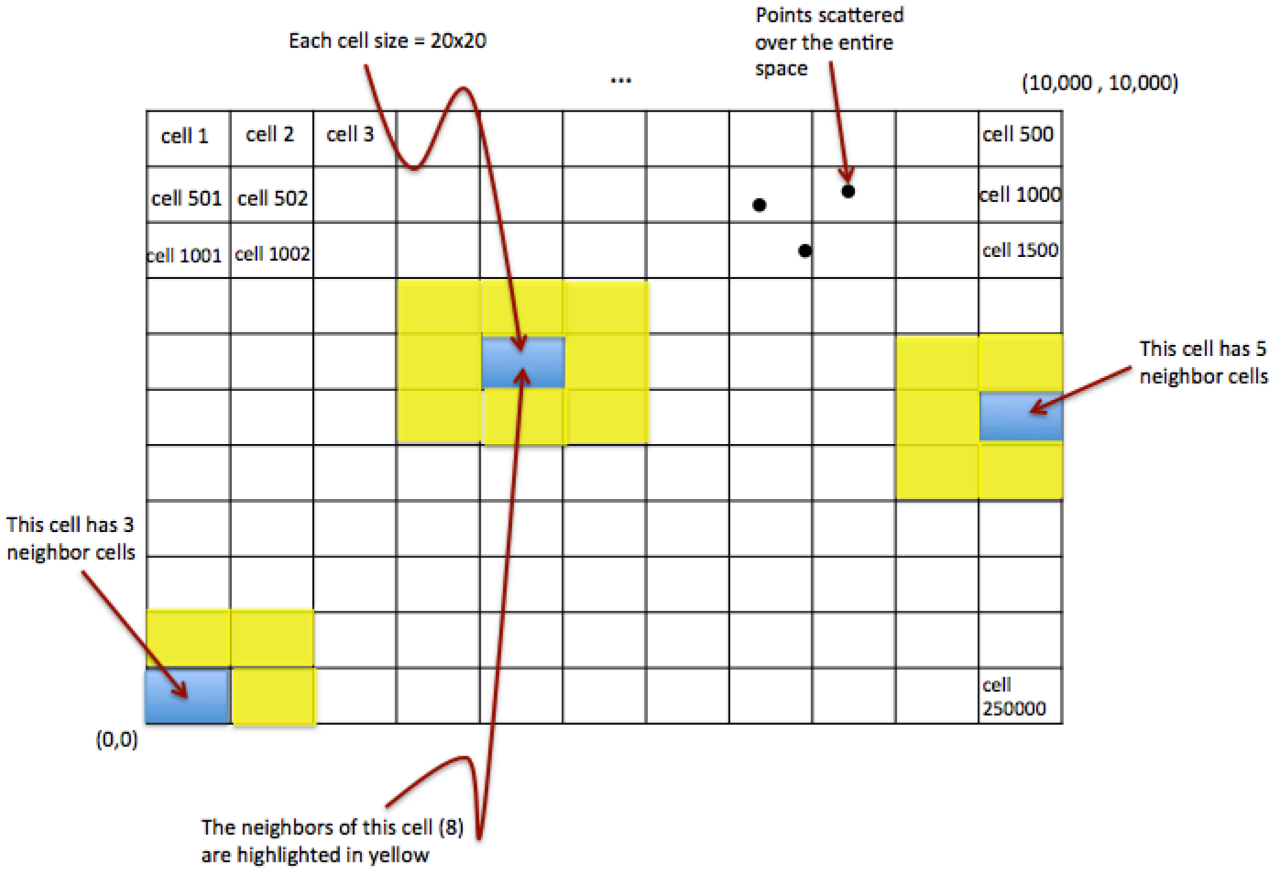
Start with T as a dataframe.

1) T1: Filter out (drop) the transactions from **T** whose total amount is less than $200
2) T2: Over T1, group the transactions by the Number of Items it has, and for each group calculate the sum of total amounts, the average of total amounts, the min and the max of the total amounts.
3) Report back T2 to the client side
4) T3: Over T1, group the transactions by customer ID, and for each group report the customer ID, and the transactions' count.
5) T4: Filter out (drop) the transactions from **T** whose total amount is less than $600
6) T5: Over T4, group the transactions by customer ID, and for each group report the customer ID, and the transactions' count.
7) T6: Select the customer IDs whose T5.count * 5 < T3.count
8) Report back T6 to the client side

Problem 2 Spark-RDDs (Grid Cells of High Relative-Density Index) [70 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 all around the space. The space is divided into pre-defined grid-cells, each of size 20x20. That is, there is 500x500= 250,000 grid cell in the space. Each cell has a unique ID as indicated in the Figure. Given an ID of a grid cell, you can calculate the row and the column it belongs to using a simple mathematical equation.



Each cell size = 20x20
...
Points scattered over the entire space
(10,000 , 10,000)
cell 1
cell 2
cell 3
cell 500
cell 501
cell 502
cell 1000
cell 1001
cell 1002
cell 1500
This cell has 5 neighbor cells
This cell has 3 neighbor cells
cell 250000
(0,0)
The neighbors of this cell (8) are highlighted in yellow

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 different examples of neighbors. Each non-boundary grid cell has 8 neighbors. However, boundary cells will have less number of neighbors (See the figure). Since the grid cell size is fixed, the IDs of the neighbor cells of a given cell can be computed using a formula (mathematical equations) in a short procedure.

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

I(X) = X.count / Average (Y_1.count, Y_2.count, …Y_n.count)

Where "X.count" means the count of points inside grid cell X, and {Y_1, Y_2, …, Y_n} are the neighbors of X. That is N(X) = {Y_1, Y_2, …, Y_n}

Step 1 (Create the Datasets)[10 Points] **//You can re-use your code from Project 2**

- Your task in this step is to create one dataset *P* (set of 2D points). Assume the space extends from 1…10,000 in the both the X and 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* is the value in the Y axis.
- Scale the dataset to be at least 100MB.
- Choose the appropriate random function (of your choice) to create the points.
- Upload and store the file into HDFS

Step 2 (Report the TOP 50 grid cells w.r.t Relative-Density Index)[40 Points]

In this step, you will write Scala or Java code (it is your choice) to manipulate the file and report the top 50 grid cells (the grid cell IDs not the points inside) that have the highest **I** index. Write the workflow that reports the cell IDs along with their relative-density index.

Your code should be fully parallelizable (distributed) and scalable.

Step 3 (Report the Neighbors of the TOP 50 grid)[20 Points]

Continue over the results from Step 2, and for each of the reported top 50 grid cells, report the IDs and the relative-density indexes of its neighbor cells.

Problem 3 Spark (PageRank Algorithm) [30 Points]

PageRank is a well-known algorithm that is described in class multiple times. Your task in this problem is to implement this algorithm using Spark (either RDDs or DataFrames is your choice).

Dataset: The dataset is provided to you on Canvas system (under Project 3 directory). The file name is “soc-LiveJournal1.txt”. It contains two columns the first column is the “source” and the second column is the “destination”. Some values in “destination” do not appear in the “source” column, which means they do not have outgoing edges. Your code should handle such case.

Your Starting Point: your code should read the file and that will be the static “Linkage” table. You should extract all DISTINCT values from the 1st and 2nd columns, and assign to each a rank of “1”. This will be your Rank table (R0).

Stopping Criteria: Do the iterative algorithm *50 times*. Do not check the output from one iteration to the next iteration. Just keep iterating until you do 50 iterations.

Final Output: After finishing the 50 iterations, report from the final rank table (Say R50) the top 100 nodes along with their final rank.

What to Submit

You will submit a single zip file containing the code needed to answer the problems above. Also include a .doc or .pdf report file containing any required documentation.

How to Submit

Use Canvas system to submit your files.