CS585/DS503 Project 2

Total Points: 200

Release Date: 31/10/2022

Due Date: 13/11/2022 Sunday (11:59 PM)

Teams: The project is to be done in teams of two

Short Description

In this project, you will write java map-reduce jobs that implement advanced operations in Hadoop as well as learn to write Scala/SparkSQL code for executing jobs over Spark infrastructure. For Spark, you can either work in a Cluster Mode (where files are read/written over HDFS), or Local Mode (where files are read/written over the 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.

What to Submit

- You will submit a single zip file containing the code needed to answer the problems above
- Include a pdf report file containing any required documentation

How to Submit

Use the Canvas system to submit your files

Problem 1 (Spatial Join) [50 points]

Spatial join is a common type of join in many applications that manage multi-dimensional data. A typical example of spatial join is to have two datasets: Dataset P (set of points in two-dimensional space) as shown in Figure 1a, and Dataset R (set of rectangles in two-dimensional space) as shown in Figure 1b. The spatial join operation is to join these two datasets and report any pair (rectangle r, point p) where p is contained inside r (or on the boundaries of r).

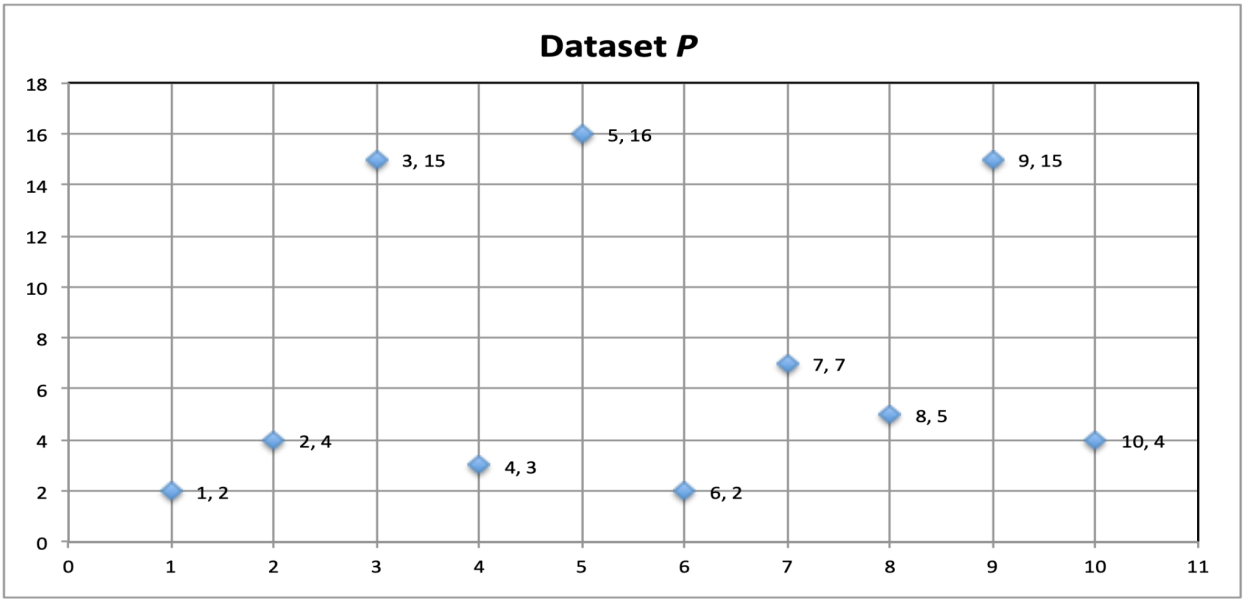


Figure 1a: Set of 2D Points

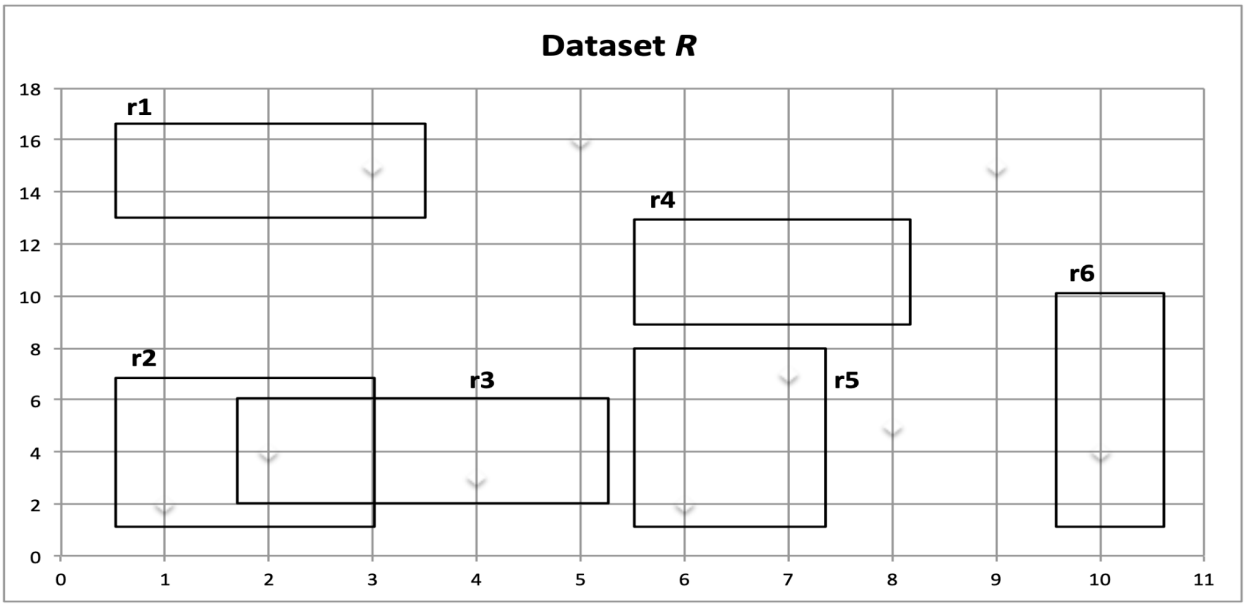


Figure 1b: Set of 2D Rectangles

For example, the join between the two datasets shown in Figure 1, will result in.
<r1, (3,15)>
<r2, (1,2)>
<r2, (2,4)>
<r3, (2,4)>
<r3, (4,3)>
<r5, (6,2)>
<r5, (7,7)>
<r6, (10,4)>

Step 1 (Create the Datasets) **[10 Points]**

- Your task in this step is to create the two datasets P (set of 2D points) and R (set of 2D rectangles)
- Assume the space extends from 1…10,000 in both the x and y-axis. Each line in file P should contain one point, and each line in file R should contain one rectangle
- Scale each dataset P and R to be at least 100MB
- Choose the appropriate random function (of your choice) to create the points. When the data is created it must be random with no specific order
- For rectangles, you need to also select a point at random (say the bottom-left corner), and then select two random variables that define the height and width of the rectangle. For example, the height random variable can be uniform between [1,20] and the width is also uniform between [1,5]. Therefore, a rectangle is defined as <bottomLeft_x, bottomleft_y, h, w>

Step 2 (MapReduce job for Spatial Join) **[40 Points]**

In this step, you need to write a java map-reduce job that implements the spatial join operation between the two datasets P and R based on the following requirements:

- The program takes an optional input parameter W(x1, y1, x2, y2) that indicates a spatial window (rectangle) of interest within which we want to report the joined objects. If W is given, then any rectangle that is entirely outside W and any point that is outside W should be skipped. If W is omitted, then the entire two sets should be joined
    - For example, referring to Figure 1, if the window parameter is W(1, 3, 3, 20), then the reported joined objects should be:
        <r1, (3,15)>
        <r2, (2,4)>
        <r3, (2,4)>
- You should have a single map-reduce job (many mappers and many reducers but in a single job) to implement the spatial join operation.

Problem 2 (K-Means Clustering) **[50 points]**

K-Means clustering is a popular algorithm for clustering similar objects into K groups (clusters). It starts with an initial seed of K points (randomly chosen) as centers, and then the algorithm iteratively tries to enhance these centers. The algorithm terminates either when two consecutive iterations generate the same K centers, i.e., the centers did not change, or a maximum number of iterations is reached.

Hint: You may reference these links to get some ideas (in addition to the course slides):
http://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm
https://cwiki.apache.org/confluence/display/MAHOUT/K-Means+Clustering

Map-Reduce Job:
Write map-reduce job(s) that implement the K-Means clustering algorithm as given in the course slides.
The algorithm should terminate if either of these two conditions becomes true:

a. The K centers did not change over two consecutive iterations
b. The maximum number of iterations (make it six (6) iterations) has been reached.

Apply the tricks given in class and in the 2nd link above such as:

- Use of a combiner
- Use a single reducer

Dataset: Use the dataset P that you created in Problem 1 as the main input dataset.

Input Parameters: The Java program should accept the HDFS file location containing the initial K centroids as a parameter. This is the file, which will be broadcasted to all mappers in the 1st round. K can be any value within the range of [10…100].

Problem 3 SparkSQL **[30 points]**

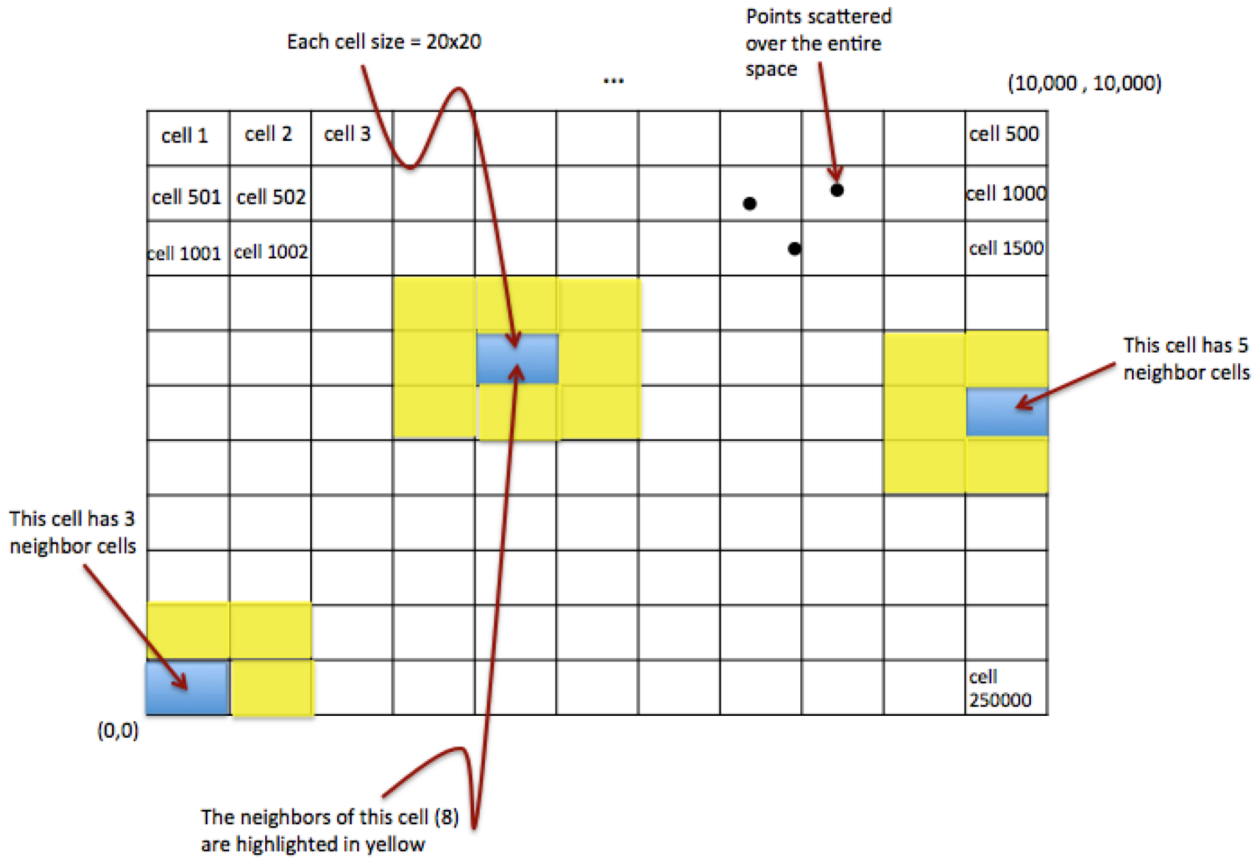
Use the Transaction dataset T that you created in Project 1 and create a Spark workflow to do the following. [Use SparkSQL to write this workflow.]

Start with T as a data frame.

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, and 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 4 Spark-RDDs **[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.



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 (Y1.count, Y2.count, …Yn.count)

Where “X.count” means the count of points inside grid cell X, and {Y1, Y2, …, Yn} are the neighbors of X. That is N(X) = {Y1, Y2, …, Yn}

Step 1 (Create the Datasets) **[10 Points]**

You can re-use your code from Problem 1

- Your task in this step is to create one dataset P (set of 2D points). Assume the space extends from 1…10,000 in 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 on the X-axis, and b is the value on 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 spark code 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.