CS585: Big Data Management

Project 2

Total Points: 200

Release Date: 09/20/2019

Due Date: 10/04/2019 (11:59PM)

Teams: Project 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 more details about Hadoop's Input Formats.

Problem 1 (Spatial Join) [50 points]

Spatial join is a common type of joins 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*).

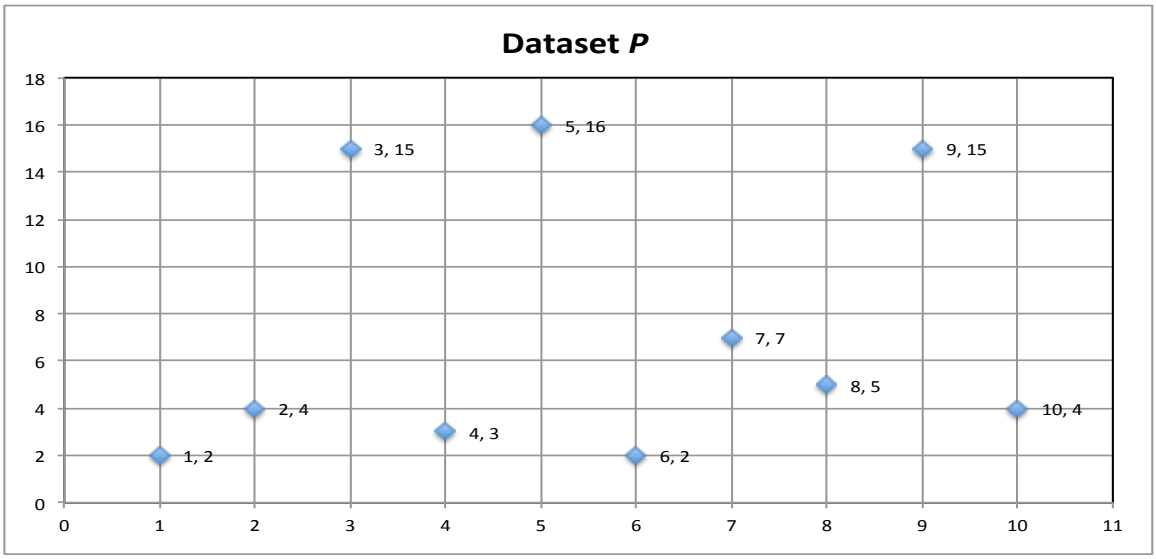


Figure 1a: Set of 2D Points

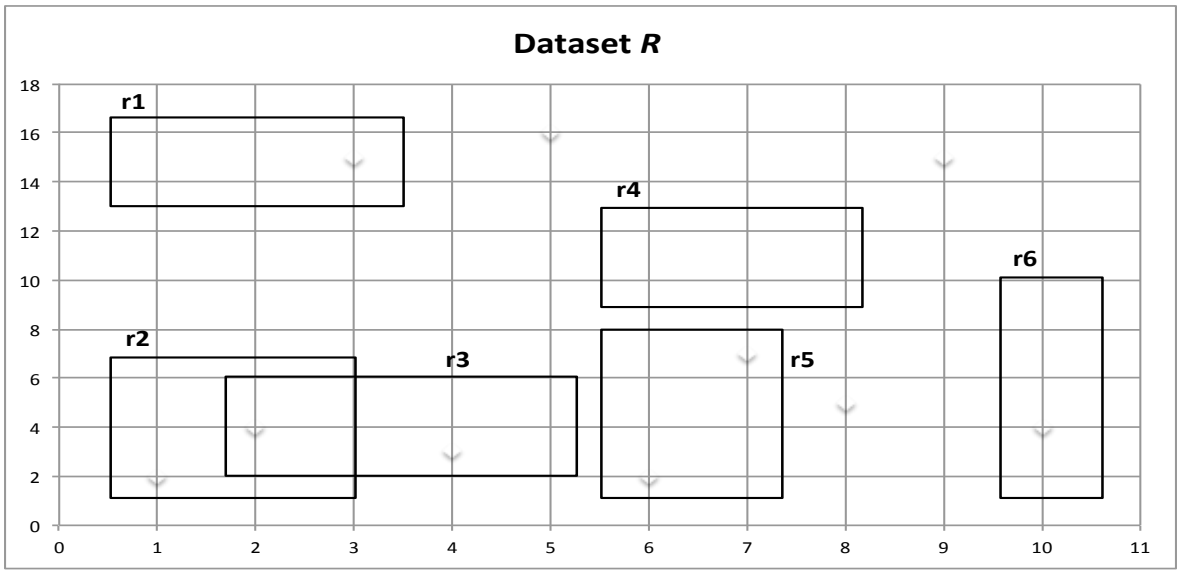


Figure 1b: Set of 2D Rectangles

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 indicate 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.
    - 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 terminates if either of these two conditions become true:

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

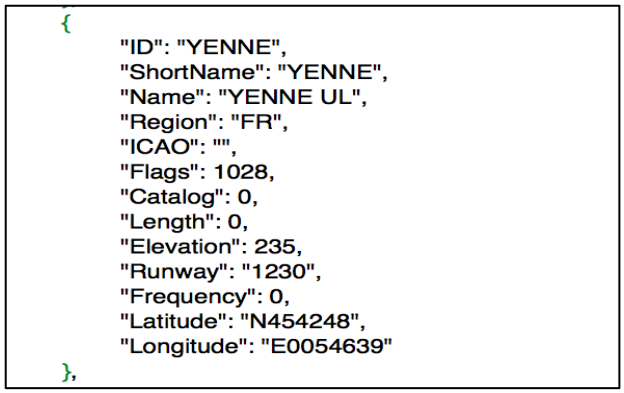
Input Parameters: The Java program should accept the HDFS file location containing the initial K centroids as a parameter. This is the file, which will be broadcasted to all mappers in the 1st round. K can be any value within the range of [10…100].

Problem 3 (Custom Input Format) [50 points]

So far, all of the given assignments use text files as input, and hence you use 'TextInputFormat()' to read the files. In this problem, you will learn more about Hadoop input formats and you will write your custom one to read the input data.

Step 1 (Data Sets)

You will use the dataset posted in Canvas System (under Project 2), the file name is "airfield.json". This file has records formatted in JSON format. Each record starts with "{" and ends with "}" (no quotes). All attributes in between form one record. Records are separated with "," after "}". For example, the following figure shows one record:



Upload this file into HDSF.

Step 2 (Map Job with a Custom Input Format)[50 Points]

- Now, to do any job on the above dataset using the standard "TextInputFormat()", the map function must be complex as it needs to collect many lines to form a single record. This complexity will repeat with each written job over the above dataset.
- A better way is to write a custom input format, call it "***JSONInputFormat***". This input format should read many lines from the input file until it gets a complete record (as the one in the figure above), and then converts these lines to a list of comma separated values in a single line, and then passes it to the map function.
  - E.g., each input to the map function should be: *id_value, shortName-value, …, …*
  - In this case, the map function is not even aware that it is reading JSON formatted file.
  - As you see the input line to a map function should have the field values in order and comma separated.

- Your task is to write this new "JSONInputFormat", and use it in a map-reduce job that aggregates the records based on the "Flag" field, and for each flag value report the maximum and minimum elevation values.

- Part of this step is to control the number of mappers that will execute to process the input file. We need to divide the file (independent from the HDFS block size) into 5 splits, which means Hadoop should start 5 mappers to process the file. *//Play with the number of splits to achieve that.*

- ***Hint:*** You need to understand first the "***FileInputFormat***", "***TextInputFormat***", and "***LineRecordReader***" classes. And you can reuse some of them and build your new one as extension.

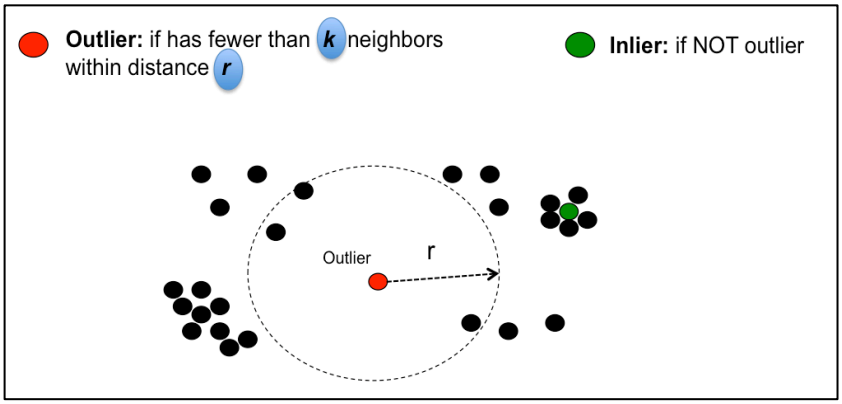
Problem 4 (Distance-Based Outlier Detection Clustering) [50 points]

Outliers are objects in the data that do not conform to the common behavior of the other objects. There are many definitions for outliers. One common definition is "*distance-based outliers*". In this definition (see the figure below), you are given two parameters, radius *r* and threshold *k*, and a point ***p*** is said to be outlier iff:

"Within a circle around p (p is the center) of radius r, less than k neighbors are found"

And point p is said to be inlier (Not outlier) iff:

"Within a circle around p (p is the center) of radius r, more than or equal to k neighbors are found"



Step 1: Dataset

Use the dataset P that you created in Problem 1. And you know the entire space boundaries, i.e., from (1,1) to (10,000, 10,000). The initial points in the HDFS file are totally in random order, and there is no specific organization. For example, referring to Figure 1(a), points (3,15), (10,4), and (4,3) can be in the 1st HDFS block, etc.

Step 2: Reporting Outliers (50 Points)

In this step, you need to write a java map-reduce job that ***reports the outliers*** based on the following requirements:

(1) The program takes two mandatory parameters *r* and *k*. If either is missing, then report an error.
(2) You must use a single map-reduce job (many mappers and many reducers but in a single job) to complete the task.
    a. If used more than one job, then for each extra job you will loose 15 points.
(3) Your code should assume distributed processing, not a single map and not a single reduce.
    a. If you assumed single map and single reduce, then you will loose 25 points

Hint: Think of diving the space in small segments. Try to make the processing of each segment independent from any other segment. That is, for a specific point p, you should be able to decide whether it is outlier or not only based on the points in p's segment.

What to Submit

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

How to Submit

Use Canvas system to submit your files.