CS585/DS503 Project 3

Total Points: 215

Release Date: 02/12/2022

Due Date: 14/12/2022 Wednesday (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

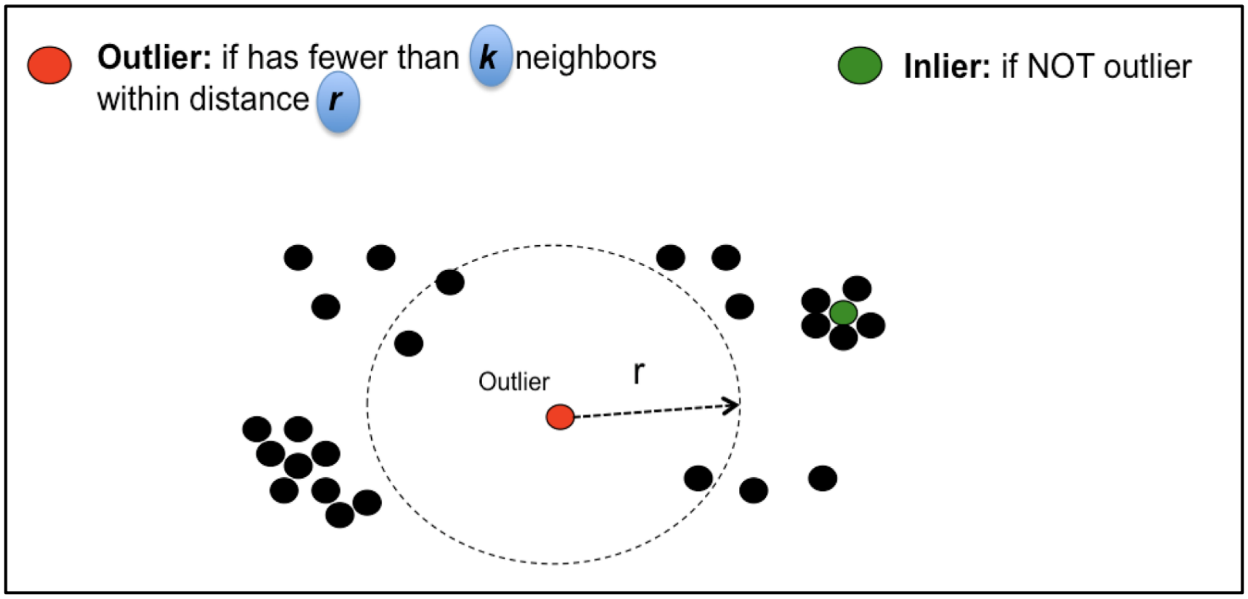
How to Submit

Use the Canvas system to submit your files

Problem 1 (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. If used more than one job, then for each extra job you will lose 15 points.
3. Your code should assume distributed processing, not a single map, and not a single reduce. If you assumed a single map and single reduction, then you will lose 25 points

Hint: Think of diving the space into 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 an outlier or not only based on the points in p’s segment.

Problem 2 (Page Rank) [50 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 the 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 cases.

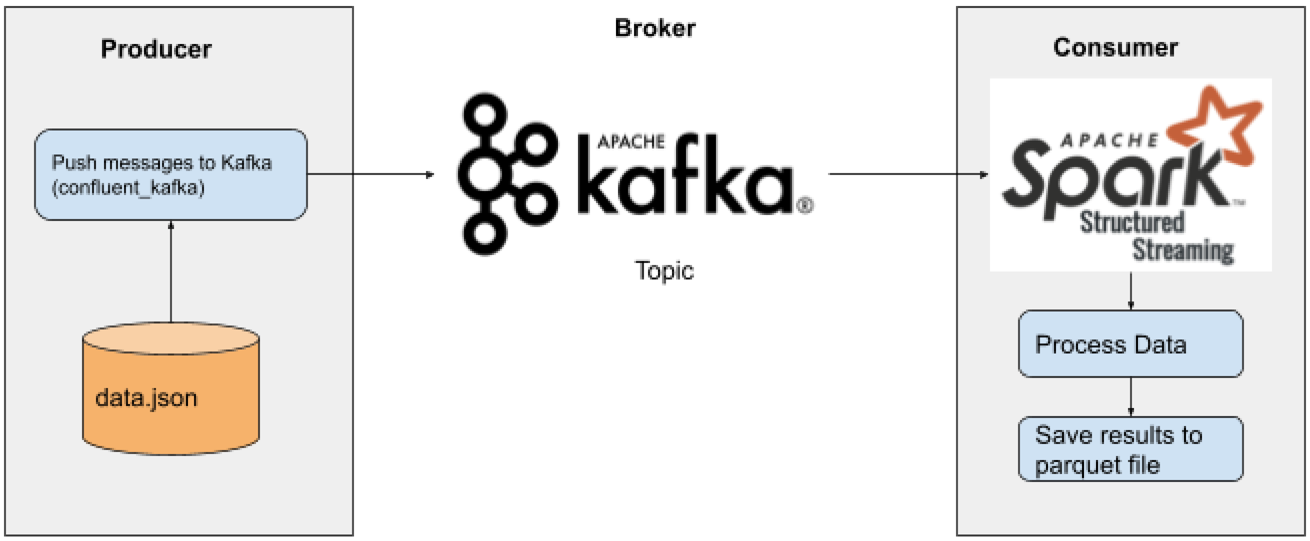
Your Starting Point: your code should read the file and that will be the static “Linkage” table. You should extract all DISTINCT values from the 1st and 2nd columns, and assign to each a rank of “1”. This will be your Rank table (R0).

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, a report from the final rank table (Say R50) the top 100 nodes along with their final rank.

Problem 3 (Kafka Spark Structured Streaming) [50 Points]

Design an end-to-end structured streaming job leveraging Kafka and Spark and use docker to orchestrate services.


Producer
Push messages to Kafka (confluent_kafka)
data.json
Broker
APACHE kafka
Topic
Consumer
APACHE Spark Structured Streaming
Process Data
Save results to parquet file

Make use of the provided template codebase and complete the functions.

Template Codebase: https://github.com/kingspp/cs585_ds503_ss_assignment

Dataset: Bios Dataset (data.json)

Steps:

1. Setup Kafka, Spark, Zookeeper, and Jupyter Notebook docker containers - docker-compose.yml
2. Read data.json from local and push records to Kafka Topic - producer.py
3. Fetch messages from Kafka in micro-batches and prepare a data frame using a pre-defined schema - schema.pkl, consumer.py
4. Perform Data processing operations:

a. age - Calculate the difference between death and birth fields in terms of years (If the death field is null, use the current date to calculate the age) Ex: For John Backus, age=83 (ceil to the nearest) (hint: pyspark has in-built function to calculate time delta - months_between)
b. num_contrib - Number of items in the contrib field. Ex: For John Backus, num_contrib=4
c. min_max_years - For the collection of each award, get the min and max of years. Ex: John Backus, min_max_years=[1967, 1993] (hint: make use of UDF)

5. Save the aggregated results Columns[name, age, num_contrib, min_max_years] to a parquet format

Result:

```
+--------------------+---+------------+------------+
|                name|age|num_contribs|     min_max|
+--------------------+---+------------+------------+
|{null, James, Gos...| 68|           1|[2002, 2007]|
…
…
```

Problem 4 (Mongo DB) [65 Points]

Question 1: [30 Points, 3 Points for each sub-question]

Your task is to design a MongoDB database and apply some CRUD (Create/Read/Update/Delete) operations as follows.

Create a collection named “test”, and insert into this collection the documents found in this link (10 documents): http://docs.mongodb.org/manual/reference/bios-example-collection/

1. Write a CRUD operation(s) that inserts the following new records into the collection:

```
{
   "_id" : 20,
   "name" : {
      "first" : "Alex",
      "last" : "Chen"
   },
   "birth" : ISODate("1933-08-27T04:00:00Z"),
   "death" : ISODate("1984-11-07T04:00:00Z"),
   "contribs" : [
      "C++",
      "Simula"
   ],
   "awards" : [
      {
         "award" : "WPI Award",
         "year" : 1977,
         "by" : "WPI"
      }
   ]
}
```

```
{
   "_id" : 30,
   "name" : {
      "first" : "David",
      "last" : "Mark"
   },
   "birth" : ISODate("1911-04-12T04:00:00Z"),
   "death" : ISODate("2000-11-07T04:00:00Z"),
   "contribs" : [
      "C++",
      "FP",
      "Lisp",
   ],
   "awards" : [
      {
         "award" : "WPI Award",
         "year" : 1963,
         "by" : "WPI"
      },
      {
         "award" : "Turing Award",
         "year" : 1966,
         "by" : "ACM"
      }
   ]
}
```

2. Report all documents of people who got less than 3 awards or have contributed in “FP”
3. Update the document of “Guido van Rossum” to add “OOP” to the contribution list
4. Insert a new filed of type array, called “comments”, into the document of “Alex Chen” storing the following comments: “He taught in 3 universities”, “died from cancer”, “lived in CA”
5. For each contribution by “Alex Chen”, say X, and list the peoples’ names (first and last) who have contribution X. E.g., Alex Chen has two contributions in “C++” and “Simula”. Then, the output should be similar to:
    a. {Contribution: “C++”, People: [{first: “Alex”, last: “Chen”}, {first: “David”, last: “Mark”}]}, { Contribution: “Simula”, ...}
6. Report the distinct organization that gave awards. This information can be found in the “by” field inside the “awards” array. The output should be an array of distinct values, e.g., [“wpi’, “acm’, ...]
7. Delete from all documents any award given in 2011.
8. Report only the names (first and last) of those individuals who won at least two awards in 2001.
9. Report the document with the largest id. First, you need to find the largest _id (using a CRUD statement), and then use that to report the corresponding document.
10. Report only one document where one of the awards is given by “ACM”.

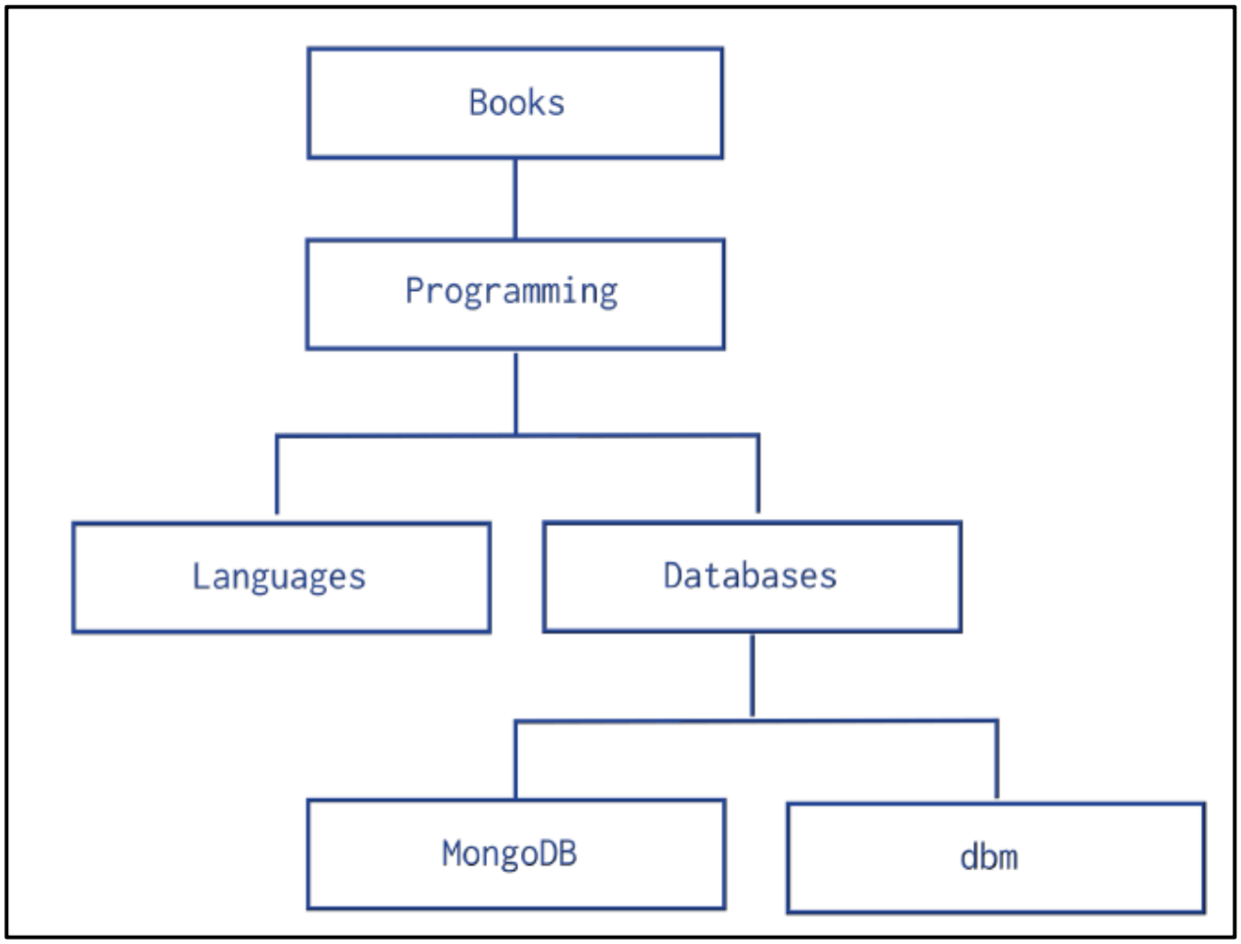
Question 2: [15 Points, 5 Points for each sub-question]

As a continuation of the dataset used in Question 1, answer the following aggregation queries:

1. Write an aggregation query that group by the award name, i.e., the “award” field inside the “awards” array, and reports the count of each award.
2. Write an aggregation query that groups by the birth year, i.e., the year within the “birth” field, and report an array of _ids for each birth year.

3. Report the document with the smallest and largest _ids. You first need to find the values of the smallest and largest and then report their documents.

Question 3 [20 Points, 5 Points for each sub-question]



1. Assume we model the records and relationships in Figure 1 using the Parent-Referencing model (Slide 4 in MongoDB-3). Write a query to report the ancestors of “MongoDB”. The output should be an array containing values [{Name: “Databases”, Level: 1}, {Name: “Programming”, Level: 2}, {Name: “Books”, Level: 3}]
   * Note: “Level” is the distance from the “MongoDB” node to the other node. It should be computed in your code

2. Assume we model the records and relationships in Figure 1 using the Parent-Referencing model (Slide 4 in MongoDB-3). You are given only the root node, i.e., _id = “Books”, and write a query that reports the height of the tree. (It should be 4 in our case).
3. Assume we model the records and relationships in Figure 1 using the Child-Referencing model (Slide 9 in MongoDB-3). Write a query to report the parent of “dbm”.
4. Assume we model the records and relationships in Figure 1 using the Child-Referencing model (Slide 9 in MongoDB-3). Write a query to report the descendants of “Books”. The output should be an array containing values [“Programming”, “Languages”, “Databases”, “MongoDB”, “dbm”]