Project 3: Big Data Management – Fall 2024

"Working with Big Data Infrastructure Spark"

Total Points:	**100**
Given Out:	**Tuesday, Oct 22nd, 2024**
Due Date:	**Tuesday, Dec 3rd, 2024**
	Submit the project via CANVAS.
Teams:	**Project is to be done in assigned teams (posted on CANVAS).**

Project Overview

In this project, you will be designing and executing scalable data processing jobs on the Spark infrastructure. You can store data in either HDFS (files are read/written over HDFS) or *local disk* (files are read/written over the local file system). In either case, your code should be designed with a high degree of parallelization so to be scalable.

Project Submission and Deliverables

1. **One** member of your team will submit a **zip file** containing the programs for creating data files, Scala/python code for your Spark analytics via CANVAS for your team. You are welcome to submit the zip file of your IDEA project directory. However, **make sure you don't include any large dataset** (a test dataset small is fine). Please don't submit a .jar file. Use the following submission format: project3_team-number.zip (for example team 5's file name will be project3_5.zip).
2. You will also submit a document (pdf) containing documentation that describes how you accomplished each task. Keep in mind that it is not just important that it "runs", it also should be a scalable solution.
3. In your project report, please indicate **the relative contributions of each team member explicitly.**[1] This requires you to discuss with each other your expectations and how best you can work together and help each other succeed at this first project. By submitting, all team members confirm the division of labor as indicated in your report.

Using Generative-AI (e.g., ChatGPT):

[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. Or, if one team member helped and taught the second team member how to do it, and then the 2nd team member succeeded to do some of the queries (even if with some guidance), please report this. If you have closely collaborated and done the same amount of effort working side by side helping each other, also please state this.

I encourage responsible and sensible use in the assignment. Please report if, how, and what you used to complete the assignment. Explain how you validated the trustworthiness of the solution, which prompt/s you used, and how you used the output of the model (basic code/documentation/etc.).

Project Demonstration

Once completed, one or at most two teams may be asked to provide a brief demonstration of their results to your classmates to review how you solved this project. **If necessary for grading, the instructor/TA will communicate with your team about your project, and also may request a demonstration of your solution.**

Some Recommended Resources

Apache Spark Documentation: https://spark.apache.org/docs/latest/quick-start.html
Spark By Examples: https://sparkbyexamples.com/

Project Description

Please complete the following tasks.

1- Spark-RDDs: Scalable Virtual Handshake Applications (40 points)

Overview: This problem explores distance-based interactions for activating virtual handshakes. In this scenario, a "virtual handshake" represents the exchange of contact information between people who come within a specified range of each other in a physical space. All attendees at the event have the HANDSHAKE app installed on their devices, but only a portion of them have the app initially activated. When two individuals are within the designated proximity (e.g., 6 units), and at least one of them has the app actively running, the HANDSHAKE app on the inactive device will automatically activate. This enables the previously inactive user to participate in virtual handshakes with others who come into their range.

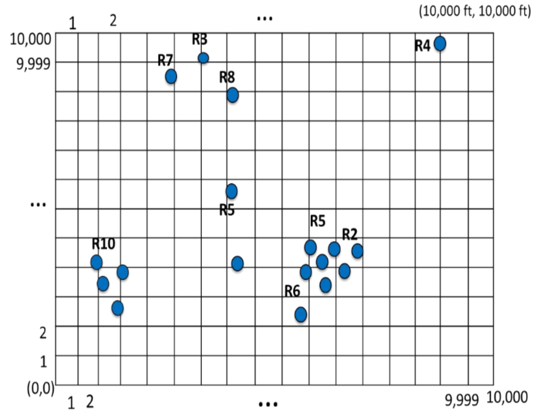

Figure 1: PEOPLE at an event standing closed to each other (2D Points).

We assume three data files, each containing two-dimensional points stored on HDFS (or locally). Each file represents a dataset of locations (x, y) where both coordinates range from 1 to 10,000 units. These locations correspond to people's positions at an event.

Each point is associated with a person's current location, along with a unique identifier and other attributes referring to their contact information such as name, age, email, etc. These points indicate the location of people at an event as in Figure 1.

Step 1 – Data Creation (4 points)**:**

Create three datasets called **PEOPLE, ACTIVATED** and **PEOPLE_WITH_HANDSHAKE_INFO**.

- Data File Structure: Each file should have the same schema, which includes:
    - 2D coordinates (x, y)
    - A unique identifier (id)
    - A set of additional attributes (e.g., name, age, email)
    - The data format for each line should be (id, x, y, attributes), where id is the person's unique identifier, x and y are their coordinates, and attributes contain additional information.

Dataset Descriptions:

PEOPLE should be a large dataset representing everyone present at the event. While all individuals have the HANDSHAKE app installed, not everyone has it actively running at the start. The dataset includes information about each person, such as their location (x, y), a unique identifier, and other attributes. The dataset does not specify which individuals have their app active initially.

ACTIVATED should be a smaller dataset that consists of people with the HANDSHAKE app initially active. It is a subset of the **PEOPLE** dataset and follows the same structure, containing the same fields for location, identifier, and attributes. These individuals are capable of activating the HANDSHAKE app for nearby users within a specified range.

PEOPLE_WITH_HANDSHAKE_INFO should be a dataset including the same information as PEOPLE but has an additional "HANDSHAKE" column indicating whether the app is currently active for each person ("yes" or "no"). At the start, the "HANDSHAKE" values are determined based on the **ACTIVATED** dataset, with "yes" for individuals in **ACTIVATED** and "no" for all others.

Random Data Generation:

You may randomly generate the points and attributes, scaling the dataset (large one) size to be fairly large (e.g., 100MB), while ensuring it remains manageable on your system.

Notes:

- **PEOPLE** should be a **large** dataset representing everyone at the event, all of whom have the HANDSHAKE app installed, but only some have it initially active.

- **ACTIVATED** is a smaller subset of **PEOPLE**, containing only those with the HANDSHAKE app active at the start.

- **PEOPLE_WITH_HANDSHAKE_INFO** provides information on the activation status for each person in **PEOPLE**. Initially, it reflects the activation states from the **ACTIVATED** dataset.

PEOPLE_WITH_HANDSHAKE_INFO is **large.** For this one you can go with one of the following options:

Step 2 – Queries:
For each of the queries below, please design a solution on the Spark infrastructure. We encourage you to work with Scala, as that is the more popular and scalable solution for using Spark natively. However, alternatively, you and your team could also opt to use instead PySpark, the Python API over Spark. But in either case, your code should be scalable and utilize **RDDs. (You are not expected to use SparkSQL to work on this problem.)** Provide a brief description and discussion of your solution.

<u>Query 1</u>: Given the datasets **PEOPLE** (large) and **ACTIVATED** (small), for each person connect-i in the **ACTIVATED**, find all individuals p-j in **PEOPLE** who are within a 6-unit range. If a person p-j is within this range and did not have the HANDSHAKE app initially active, assume that their app gets activated. Return a list of join pairs (pi, connect-i) indicating that pi from **PEOPLE** has been activated by being near connect-i. (12 points)

Note: You cannot use the dataset of **PEOPLE_WITH_HANDSHAKE_INFO** to help you get individuals who are from **PEOPLE** but not **ACTIVATED**. For this query, you can only use **PEOPLE** or **ACTIVATED** datasets.

<u>Query 2</u>: Given the same setup in Query 1, however, return the unique identifiers pi.id of all individuals in **PEOPLE** who were within 6 units of any person in **ACTIVATED**, ensuring each person appears **only once** in the result **with duplicates removed**. That is, even if a person pi was close to two different "activated" people, return that person pi only once. (12 points)

Note: Same as query 1, you cannot use the dataset of **PEOPLE_WITH_HANDSHAKE_INFO** to help you get individuals who are from **PEOPLE** but not **ACTIVATED**. For this query, you can only use **PEOPLE** or **ACTIVATED** datasets.

Query 3: Given the **PEOPLE_WITH_HANDSHAKE_INFO** dataset, count how many individuals are within a 6-unit range of each "activated" person (those with "HANDSHAKE" = "yes"). Return pairs in the format (connect-i, count-of-close-contacts) for each initially "activated" individual, where count-of-close-contacts indicates the number of nearby people. (12 points)

2- SparkSQL and MLlib: Processing Purchase Transactions (60 points)

Overview: This problem studies Purchase Transactions from users. After generating the data, you will be required to analyze the data using SparkSQL. After which you will utilize Spark MLlib to predict the predict Purchase Prices and analyze the results.

Step 1 – Data Creation:
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 18 to 100
CountryCode: random number (integer) between 1 and 500
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 2000
TransNumItems: Number of items as random number (integer) between 1 and 15
TransDesc: Text of characters of length between 20 and 50 (careful: no commas)

Task 2.0) Load your data into your storage. (5 points)

Use SparkSQL to express the following workflows (20 points)

Task 2.1) Filter out (drop) the Purchases from P with a total purchase amount above $600. Store the result as T1. (5 points)

Task 2.2) Group the Purchases in T1 by the Number of Items purchased. For each group calculate the median, min and max of total amount spent for purchases in that group. Report the result back to the client side. (5 points)

Task 2.3) Group the Purchases in T1 by customer ID only for young customers between 18 and 25 years of age. For each group report the customer ID, their age, and total number of items that this person has purchased, and total amount spent by the customer. Store the result as T3. (5 points)

Task 2.4) Return all customer pairs IDs (C1 and C2) from T3 such that

a. C1 is younger in age than customer C2 and
b. C1 spent in total more money than C2 but bought less items.

Store the result as T4 and report it back in the form (C1 ID, C2 ID, Age1, Age2, TotalAmount1, TotalAmount2, TotalItemCount1, TotalItemCount2) to the client side. (5 points)

Use MLlib to express the following workflows (35 points):

Task 2.5) Data Preparation 1: Download a related dataset from Kaggle:

- **ecommerce_customer_data_large.csv** form https://www.kaggle.com/datasets/shriyashjagtap/e-commerce-customer-for-behavior-analysis?select=ecommerce_customer_data_large.csv (1 point)
- Load the dataset into the (2 points)

Task 2.6) Data Preparation 2: (Randomly) split *Dataset* into two subsets, namely, *Trainset* and *Testset*, such that *Trainset* contains 80% of *Dataset* and *Testset* the remaining 20%. (2 points)

Task 2.7) **Prediction: Choose a target attribute to predict (Product Price/ Gender/…)**

- Prediction type: The chosen target attribute will dictate the type of prediction you will be focusing on. Largely speaking, we will focus on classification/regression (https://spark.apache.org/docs/latest/mllib-classification-regression.html) but you are welcome to explore others (e.g., Clustering). If the value you aim to predict is numeric (e.g., Product Price), you will work with regression (https://spark.apache.org/docs/latest/ml-classification-regression.html#regression). If choose to predict a categorical value, e.g., Product

Category or Gender, then you will focus on a classification task (https://spark.apache.org/docs/latest/mllib-linear-methods.html#classification)

- Preparation: If
- **Task 2.7.1)** Identify and train at least 3 machine learning algorithms to predict your target attribute. For the first version of the model, focus on the numeric features of the dataset (i.e., Product Price, Quantity, Total Purchase Amount, Customer Age, Returns and Churn). (8 points)
    - The algorithms should be trained over *Trainset* (Task 2.6) and applied (inference) over *Testset* (Task 2.6).
    - **Note:** Avoid using your target attribute as a feature.

- **Task 2.7.2)** Identify and train at least 3 machine learning algorithms to predict your target attribute. For the advanced version of the model, try to use all the features of the dataset. For example, use one-hot-encoding to encode categorical features (https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.OneHotEncoder.html). (8 points)
    - Explain the additional features and how you processed them.
    - The algorithms should be trained over *Trainset* (Task 2.6) and applied (inference) over *Testset* (Task 2.6).

Task 2.8) **Evaluate your prediction:**

- Identify and use at least 3 metrics to evaluate the algorithms from Task 2.7. (8 points)
- **Note**: use the appropriate evaluation metric based on the task you aim to solve.
    - If you use Regression: https://spark.apache.org/docs/1.6.1/mllib-evaluation-metrics.html#regression-model-evaluation.
    - If you use Classification: https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html

Task 2.9) **Report and discuss the results:** Analyze the experimental results and report your conclusion. The analysis should compare results (Task 2.8) of the different models (Task 2.7) in a table/graph. Report your observations discuss their implications (e.g., Model X was the most/least effective/efficient because ABC…). (6 points)