Data Processing at Scale (CSE 511) MEng Portfolio Report

Yu Chang
Arizona State University
Tempe, US
ychan175@asu.edu

I. Introduction

This is a portfolio report for the Data Processing at Scale (CSE 511) course taken in the Fall 2024 semester. In this course, we completed two projects that required the application of topics and concepts taught during regular class sessions.

In Project 1, I developed two Python functions that interact with a NoSQL database, aimed at solving real-world data retrieval problems. These functions were designed to fetch relevant business information based on user-defined input criteria. The first function retrieves all businesses located in a specified city and saves the extracted data to a designated file for further use. The second function searches for businesses of a specific category within a given radius of a specified geographical location and stores the results in a predefined file. These functions were developed with a focus on efficient data retrieval, designed to support practical business applications that rely on timely and accurate information.

In Project 2, I worked on two hotspot analysis tasks that used spatial statistical techniques to process spatiotemporal big data with Apache Spark and Scala. The objective was to identify statistically significant spatial hotspots by analyzing large-scale data. These tasks were derived from the 2016 ACM SIGSPATIAL GISCUP competition but simplified to fit the course context and manage computational complexity. Task 1 involved calculating and ranking the number of taxi pickups within specified rectangular geographic zones based on input datasets. Task 2 was aimed at identifying taxi pickup hotspots using the Getis-Ord Gi* statistic [1], computing and highlighting the top 50 spatial-temporal hotspots.

II. Methods and Solution

A. Project 1

In Project 1, I implemented two essential functions specifically designed to efficiently retrieve business data from a NoSQL database. These functions were written in Python, which was chosen for its simplicity and flexibility in handling data-driven tasks. The purpose of these functions was to facilitate seamless interaction with the NoSQL database and make the overall data retrieval process more efficient: `FindBusinessBasedOnCity` and `FindBusinessBasedOnLocation`.

The first function, `FindBusinessBasedOnCity`, operates by accepting three key parameters: the target city that the user wants to search for, the file save location where the final results will be stored, and the business collection that contains the dataset. The function systematically iterates through the entire collection of business data, extracting crucial details such as the business name, full address, city, and state. If a business is found to be located in the specified city, it is added to a results list. This results list is then carefully formatted and saved to a file, with each entry separated by dollar signs ($) to maintain a standardized structure for the output, ensuring consistency in the format.

The second function, `FindBusinessBasedOnLocation`, is designed to focus on identifying businesses based on their geographical proximity and category. It requires five input parameters: the categories of businesses to search for, the user's current geographic location (specified by latitude and longitude), the maximum search radius in miles, the file save location for storing the results, and the business collection containing the relevant data. The function first checks if each business belongs to any of the specified categories, and then calculates the exact distance between the user's location and the business location using the Haversine formula [2]:

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cdot\cos(\varphi_2)\cdot\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

If the calculated distance falls within the specified search radius, the business is added to the result list, which is subsequently saved to a file for easy access and later use.

B. Project 2

In Task 1, a hotzone analysis was performed to count the number of taxi pickups within predefined rectangular geographic zones. The implementation utilized Apache Spark and Scala to handle large-scale spatial data effectively. The primary goal of this task was to calculate how many taxi pickups fell within each rectangular zone based on the input dataset and then rank these zones by the number of pickups.

To start, two datasets were loaded into Spark's DataFrame API: the taxi pickup data (in CSV format), which provided geographical coordinates for each pickup, and the rectangle data, representing the predefined geographic zones. Both datasets were stored as temporary views in Spark to facilitate efficient SQL queries for the spatial analysis [3].

Before performing the spatial analysis, the taxi pickup data underwent a preprocessing step. This involved cleaning the coordinate data by removing extraneous characters such as parentheses to ensure that the data was in a standardized format suitable for further processing.

Once the data was preprocessed, a user-defined function (UDF) was employed to determine whether each taxi pickup point fell within the boundaries of a given rectangle. This function was applied during a spatial join between the two datasets, filtering the points to retain only those that were located inside the rectangles.

To calculate the number of taxi pickups within each rectangle, a SQL query was used to group the points by rectangle and compute the count of pickups within each zone. The results were then ordered by rectangle ID to provide a ranked summary of taxi pickups within the zones:

```
SELECT rectangle, count(*) as point_count
FROM joinResult
GROUP BY rectangle
ORDER BY rectangle
```

This query produced the final output of Task 1, which summarized the pickup density in each rectangular zone.

In Task 2, the main focus was on performing a comprehensive hotspot analysis using the Getis-Ord Gi* statistic to identify statistically significant clusters of taxi pickups within a specific geographic area. The objective was to determine and pinpoint the top 50 hotspots by calculating the G* score for each spatial-temporal cell. The analysis made use of Apache Spark and Scala to efficiently manage and process the large-scale spatiotemporal dataset, ensuring smooth and rapid handling of the data.

The taxi pickup dataset, which included both geographical coordinates and timestamps, was first loaded from a CSV file into Spark. In order to carry out the hotspot analysis, the spatial and temporal information needed to be transformed into a grid structure, where X and Y represented longitude and latitude, and Z represented the time dimension. This transformation process was achieved using user-defined functions (UDFs) that prepared the raw data, making it suitable for further in-depth processing.

Once the data was properly preprocessed, the number of taxi pickups within each cell was calculated using a carefully constructed SQL query. This query aggregated the data based on the X, Y, and Z coordinates and counted the number of pickups occurring within each spatial-temporal cell. The query ensured that only the cells falling within the specified geographic and temporal boundaries were included:

```
SELECT x, y, z, COUNT(*) as count
FROM pickupinfo
WHERE x BETWEEN $minX AND $maxX
  AND y BETWEEN $minY AND $maxY
  AND z BETWEEN $minZ AND $maxZ
GROUP BY x, y, z
```

This step resulted in the generation of a core dataset that contained the pickup counts for each cell, which formed the foundational basis for further analysis.

Next, global statistics were computed using another SQL query. These statistics included the global mean, standard deviation, and total number of cells, which are critical for determining the statistical significance of each cell's pickup count. The SQL query used to compute these essential statistics was as follows:

```
SELECT AVG(count) as mean,
     STDDEV(count) as std,
     COUNT(*) as num_cells
FROM pointcounts
```

These global values acted as the reference points for the subsequent G score calculation, providing a necessary baseline.

To account for spatial dependence between neighboring cells, the next step involved calculating the sum of point counts in neighboring cells for each target cell. Using the `IsNeighbor` UDF, which identifies adjacent cells, the following SQL query computed the total number of points in each cell's neighborhood, as well as the number of neighboring cells:

```
SELECT p1.x as x, p1.y as y, p1.z as z,
     SUM(p2.count) as neighborsum,
     COUNT(*) as neighborcount
FROM pointcounts p1, pointcounts p2
WHERE IsNeighbor(p1.x, p1.y, p1.z,
                 p2.x, p2.y, p2.z)
GROUP BY p1.x, p1.y, p1.z
```

The results from this query provided the local neighborhood sums and counts, which are crucial for assessing the local hotspot significance in relation to the overall dataset.

The G* score for each cell was then calculated using the following query, which utilized the previously computed neighbor sums, counts, and global statistics:

```
SELECT x, y, z,
     GScore(x, y, z, neighborsum,
            neighborcount) as g_score
FROM neighborinfo
```

The `GScore` UDF calculated the G* score by comparing the local neighborhood sum to the global mean and standard deviation. The G* score reflects whether a cell and its surrounding neighborhood represent a statistically significant hotspot or a coldspot.

Finally, the top 50 hotspots were identified by ranking the cells with the highest G* scores using the following SQL query:

```
SELECT x, y, z
FROM gscores
ORDER BY g_score DESC
LIMIT 50
```

This query arranged the cells by their G* score in descending order, ensuring that the top 50 statistically significant hotspots were successfully selected for further analysis.

III. Result

A. Project 1

The first function successfully retrieved businesses located in the specified city from the database. Below is an example of how the function was executed:

```
FindBusinessBasedOnCity('Scottsdale',
    'output_city.txt', data)
```

Function output:

```
["3 Palms$
 7707 E McDowell Rd, Scottsdale, AZ 85257$
 Scottsdale$AZ",
 "Bob's Bike Shop$
 1608 N Miller Rd, Scottsdale, AZ 85257
 $Scottsdale$AZ",
 "Ronan & Tagart, PLC$
 8980 E Raintree Dr, Ste 120,
 Scottsdale, AZ 85260$
 Scottsdale$AZ",
 "Sangria's$
 7700 E McCormick Pkwy,
 Scottsdale, AZ 85258$
 Scottsdale$AZ",
 "Turf Direct$
 8350 E Evans Rd, Scottsdale, AZ 85260$
 Scottsdale$AZ"]
```

The second function successfully identified businesses of a specific type within a given radius of the specified geographical coordinates. Here's an example of how the function was called:

```
FindBusinessBasedOnLocation(['Gardeners'],
    [33.3482589, -111.9088346], 20,
    'output_loc.txt', data)
```

Function output:

```
['Turf Direct']
```

B. Project 2

The Task 1 was successfully completed, calculating the number of taxi pickups within 165 distinct rectangular regions based on the input data. Below are some sample outputs:

```
"-73.862040,40.706406,
-73.829798,40.739016",16
"-73.864482,40.833000,
-73.851686,40.842478",1
"-73.867911,40.734291,
-73.847510,40.748860",1
"-73.869236,40.765468,
-73.825139,40.798517",136
"-73.873659,40.744942,
-73.853128,40.758349",1
```

The second task was also successfully executed. It calculated the Getis-Ord Gi* statistic for all cells within the designated area and identified the top 50 cells with the highest scores. Here are some examples of the output:

```
-7399,4075,15
-7399,4075,29
-7399,4075,22
-7399,4075,28
-7399,4075,14
```

IV. Contribution

For this project, I was responsible for all the work. I wrote the Python functions, used a NoSQL database to get business data, and added a distance calculation using the Haversine formula. I also worked on improving the code so it could process large datasets. Finally, I tested the functions to make sure they were working correctly and gave the right results.

V. New Skill Acquire

Throughout the development of both Project 1 and Project 2, I gained a variety of new technical skills that significantly improved my proficiency in various areas of data processing and software development.

In Project 1, I enhanced my Python programming skills by developing custom functions that interact with a NoSQL database to efficiently retrieve and process business data based on specific user inputs. During this process, I also learned how to implement complex geospatial calculations using the Haversine formula, which allowed me to accurately determine distances between various geographic points. Additionally, I gained valuable experience in optimizing code to ensure it could handle large datasets effectively while also focusing on efficient file management techniques for storing and organizing query results.

In Project 2, I further expanded my knowledge of data processing and system setup by working with a range of different environments. I experimented with setting up Apache Spark in various configurations, including using VirtualBox with multiple Linux distributions, running Spark on Windows, and working with WSL (Windows Subsystem for Linux) through an Ubuntu installation. Through this process, I developed skills in utilizing Apache Spark for large-scale data processing tasks, writing efficient SQL queries to manipulate and analyze datasets, and creating custom user-defined functions (UDFs) in Scala to extend the functionality of the system. This project also provided me with deeper insights into distributed computing and the efficient handling of big data in real-world applications.

Overall, through the experience gained in both projects, I developed a strong foundation in managing and processing large datasets, optimizing system performance, and utilizing modern data processing tools to achieve efficient and scalable solutions.

References

[1] A. Getis and J. K. Ord, ”The analysis of spatial association by use of distance statistics,” Geographical Analysis, vol. 24, no. 3, pp. 189-206, 1992. Accessed: Oct. 21, 2024. doi: 10.1111/j.1538-4632.1992.tb00261.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.1992.tb00261.x

[2] ”Haversine formula,” Wikipedia. https://en.wikipedia.org /wiki/Haversine_formula. (Accessed: Oct. 20, 2024).

[3] ”Spark SQL, DataFrames and Datasets Guide,” Apache Software Foundation. https://spark.apache.org/docs/latest/sql-programming-guide.html (Accessed: Oct. 20, 2024).