CSE 511 - Data Processing At Scale
Systems Documentation Report
Group - 20

Nithiya Shree Uppara	-	(1215229834)
Rajashekar Reddy Aluka	-	(1217211645)
Venu Gopinath Nukavarapu	-	(1216106879)
Venkata Bhargav Siddhartha Kondamuri	-	(1217370037)
Rajasree Chennupati	-	(1218519276)
Naga Sai Aishwarya Tallapragada	-	(1218515961)

1. Introduction

Geospatial data is any data that is referring to a geographic location on earth with the help of spatial identifiers i.e. latitude and longitude. This geospatial data could refer to a point location like the location of a house, restaurant, museum, etc. or it could also encompass an area like the boundaries of the state of Arizona. In present-day geospatial data is present all around us be it in location-tagged Instagram and Facebook posts, location-tagged tweets, suggested routes by google maps, location of stores, etc. The analysis made with the help of geospatial data is extremely useful in the present market as many applications are using this information to customize the user experience for the end-users. Especially for taxi services that rely heavily upon the geospatial data, proper analysis done on the customer data collected to date can help the company make strategic business decisions crucial for the company's growth.

The project involves developing and running multiple spatial queries on a large database that contains geographic data as well as real-time location data of customers, for a large peer-to-peer taxi firm to retrieve data to be used by the firm. Since the data is mostly unstructured, the client wanted the popular Big Data software SparkSQL to be used to develop and run the spatial queries. The goal of the project is to retrieve the data from the database which will be further utilized by the client for operational and strategic level decisions.

Phase 1

In phase 1 of the project, there are 4 spatial queries that are used to retrieve the geographic data. There are two user-defined functions that are supposed to be written by the developer. The user-defined functions are ‘ST_Contains’ and ‘ST_Within’.These two user-defined functions are used by the 4 queries to help retrieve the data. The function ‘ST_Contains’ is used to determine if a point lies within a given rectangle. This function takes a point and rectangle coordinates as input and returns a Boolean value as output depending on whether the conditions are satisfied. The function ‘ST_Within’ is used to determine if two points are within a certain fixed distance from each other. This function is given the points and the fixed distance as input and it returns a Boolean value

as output depending on whether the conditions are satisfied. The four spatial queries are Rang query, Range join query, Distance query, Distance join query.

Range query
Given a rectangle and a set of points this query is used to identify how many of the points from the set lie inside the specified rectangle. This data is retrieved with the help of the function ‘ST_Contains’.

Range join query
When given an input of a set of rectangle coordinates and set of points, this query is used to identify all the rectangle and point pairs such that the points lie inside the rectangle. The function ‘ST_Contains’ is used to identify if the points lie inside the rectangle.

Distance query
Given an input of a point, a fixed distance, and a set of points, this query is used to identify and return all the points from the set that are within a fixed distance from the given point. The function ‘ST_Within’ is used to identify whether the points are within a fixed distance from the given point.

Distance join query
Given an input of two sets of points and a distance, this query is used to identify and return all the pairs of points such that the first point is from set 1 and the second point is from set 2 and these points are within a fixed distance from each other. The function ‘ST_Within’ is used to identify all such points.

Phase 2
In this phase, we are implementing spatial hot spot analysis to find the Hot Zones and Hot Cells inside the chosen rectangular dataset. This Phase is further divided into Hot Zone analysis and Hot Cell analysis building on top of the functionality implemented in phase1.

Hot zone analysis: In the Hot zone analysis we are using the rectangular dataset in the spatial data to find a number of points concentrated at different locations inside the rectangle dataset. In this analysis, we are extracting all the points in the rectangular dataset and the concentration is measured based on the density of the points at any given point. The hotness is measured in such a way that the Hotness of a cell is proportional to the concentration of points at any given point in the rectangle dataset.

Hot Cell analysis: In the Hot cell analysis, we are using temporal and spatial data to find a significantly hotter spot. We are applying spatial statistical analysis to the present Hot Cells to find significantly hotter cells.

Task Distribution

This project is implemented by a team of six, sub divided into two teams of three. In order to efficiently complete the project, each phase of the project is further divided into two parts. The teams are divided based on their interests and capabilities.

The first phase is divided among the teams based on the functionality of the project. Team1(Nithiya, Rajashekar, Venu) worked on implementing the ST_Within() function and other tasks related to the function. Team2(Siddhartha, Rajasree, Aishwarya)worked on implementing the ST_Contains() function and other tasks related to the function. We used Github to collaborate among the teams.

Similar to phase1, the phase2 is divided among the teams to implement Hot Zone analysis and Hot Cell analysis. Team1(Venu, Rajasree, Aishwarya) worked on implementing Hot Zone analysis. Whereas Team2(Siddhartha, Rajashekar, Nithiya) worked on implementing Hot Cell Analysis. We used Github to collaborate on the project.

Pitfalls

While Implementing Hot Zone analysis, we faced difficulty as the program is generating multiple output files. The requirement states that the generated output should be stored in only one file. We have addressed the issue by changing the return finalResultDf command in hot zone analysis.scala to return finalResultDf.coalesce(1) which helped us storing the output in one file.

2. Requirements

The below frameworks and technologies for required for the implementation of the project :

1. **Scala :** Scala is a general-purpose programming language providing support for functional programming and a strong static type system. Scala is object-oriented and also has many features of functional programming languages like Scheme, Standard ML and Haskell.

2. **Spark : Apache Spark** is an open-source distributed general-purpose cluster-computing framework originally developed at the University of California, Berkeley's AMPLab, which provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Spark can be deployed in a variety of ways to provide native bindings for the Java, Scala, Python, and R programming languages, and supports SQL, streaming data, machine learning, and graph processing. It also supports a rich set of higher-level tools including Spark SQL for SQL and DataFrames, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for stream processing.

3. **Spark SQL : Spark SQL** is a component on top of Spark Core that introduced a data abstraction called DataFrames, which provides support for structured and semi-structured data. Spark SQL provides a domain-specific language (DSL) to manipulate DataFrames in Scala, Java, or Python.

   For our project, we used
   - spark - version 3.0.0
   - hadoop - version 2.7
   - Scala - version 2.13.1

3. Implementation Details

Phase 1

In Phase1 , Spark SQL is used to run four spatial queries namely RangeQuery, RangeJoinQuery , Distance Query and Distance joinQuery. These queries use two user-defined functions 'ST_Contains' and 'ST_Within' to run four spatial queries.

ST_Contains

ST_Contains takes 2 parameters: a point coordinate and rectangle coordinates. It returns a boolean value indicating whether the given point lies within the rectangle. To check whether the given point lies within the rectangle, we check whether the given point's x coordinate lies within the min and max of the rectangle's x coordinates and point's y coordinate lies within the min and max of the rectangle's y coordinates

We implemented 2 spatial queries RangeQuery and RangeJoinQuery with ST_Contains function.

1. **Range Query:**
   This query takes a rectangle R and a set of points P as input and returns all the points among P which lies inside the Rectangle R. It uses the ST_Contains method defined above.
2. **Range Join Query:**
   This query takes a set of rectangles R and a set of Points P as input and returns all rectangle,point pairs which satisfy the condition that point lies within the rectangle.It uses the ST_Contains function defined above.

ST_Within

ST_Within takes 3 parameters: a point coordinate p1, another point coordinate p2 and a double value distance. We first compute the euclidean distance between the points P1 and P2. It returns a boolean value indicating whether the euclidean distance between the points is less than the distance value or not.

We implemented 2 spatial queries DistanceQuery and DistanceJoinQuery with ST_Within function.

1. **Distance Query:**
   This query takes a point P and distance d as input and returns all the points that lie within a distance d from point P. It uses the ST_Within function defined above.
2. **Distance Join Query:**
   This query takes a set of Points P1,another set of Points P2 and distance d as input and return all pairs of points(p1,p2) which satisfy the condition that point p1 lies within a distance d from p2(p1 is a point that belongs to point set P1 and p2 is a point that belongs to point set P2) .It uses the ST_Within function defined above.

CodeFlow:

- SpatialQuery.scala
  - userContains function
    - Returns bool true or false by checking if the points are inside the cube.
  - userWithin function
    - Returns true or false by checking if the euclidean distance between the points is less than the distance value or not.
  - runRangeQuery
    - Returns all the points among P which lies inside the Rectangle R.
  - runRangejoinQuery
    - Returns all rectangle,point pairs which satisfy the condition that point lies within the rectangle.
  - runDistanceQuery
    - Returns all the points that lie within a distance d from point P
  - runDistanceJoinQuery
    - Returns all pairs of points which satisfy the condition that point p1 lies within a distance d from p2

<u>Phase 2</u>

1. **Hot Zone Analysis**
   In the implementation of Hot Zone analysis, there are two Scala files namely HotzoneAnalysis.scala and HotzoneUtils.scala. Code in the former file calls functions that are defined in the latter. HotzoneUtils.scala contains the ST_Contains() function. This is the same function that was written in phase 1 which returns true if the given query point is contained inside the query rectangle and returns false otherwise.

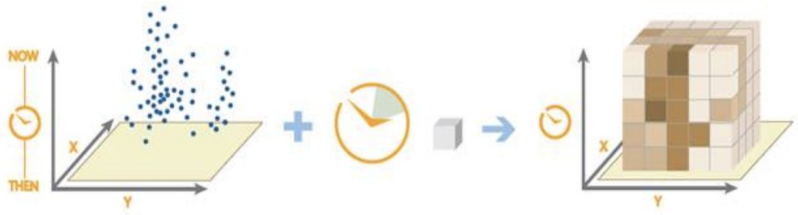
   In HotzoneAnalysis.scala, the ST_Contains() function is called in the step where the data frame is generated as the result of joining the two datasets passed and the result is saved in a view called joinResult. The query we write works on this dataframe to generate the final result and return it to the caller. This query returns

the rectangle along with the number of points contained in it sorted by the ascending order.

2. **Hot cell Analysis**

In hot cell analysis, we calculate the z-scores i.e. getis-ord, in order to identify statistically significant spatial hot spots.

To calculate the z-scores, we are provided with spatio-temporal points. These points are formed by converting the existing 2D hot zones as below into spatio-temporal cubes by adding time dimension to it.



We use the following formulae to calculate z-score. The z-score calculation requires the calculation of mean ($\bar{X}$) and standard deviation (S) of the spatio temporal points.

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j} x_j - \bar{X} \sum_{j=1}^{n} w_{i,j}}{S \sqrt{\frac{\left[n \sum_{j=1}^{n} w_{i,j}^2 - \left(\sum_{j=1}^{n} w_{i,j} \right)^2 \right]}{n-1}}}$$

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n}$$

$$S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\bar{X})^2}$$

The execution of hot cell analysis code starts from the driver function (runHotcellAnalysis) mentioned in HotCellAnalysis.scala. We have helper functions in the HotcellUtils.scala to calculate whether the point is present inside the cube, the number of neighbours around a cell and to calculate z-score.

Code Flow:

HotCellAnalysis.scala

- runHotcellAnalysis
  - register isCube() function from HotcellUtils
  - Using spark-sql we calculate whether each point is inside the cube or not by using the isCube().
  - To calculate mean, we call calMean function from HotcellUtils.
  - Calculate stand deviation.
  - Register neighbour cell function from HotcellUtils.
  - Calculate neighbour cells for each cell using spark-sql.
  - Register Gscore function HotcellUtils.
  - Calculate Gscore for each cell by passing the neighbour count, mean, standard deviation and sum of neighbour weights.
  - Finally returns the top 50 rows in the descending of z-score using spark-sql.

HotcellUtils.scala

- isCube function
  - Returns bool true or false by checking if the points are inside the cube.
- calMean
  - Returns double value i.e. mean.
- NeighbourCount
  - Returns integer i.e. the no neighbour cells for a cell.
  - No of neighbours "Inside"(26) , "face"(17),"edge"(11),"corner"(7)
- GScore
  - Returns the z-score.

4. Testing Process and Results

The jar file is created by running the command “sbt assembly” or “sbt clean assembly” depending on the phase working upon inside the top level folder inside the code base. If all syntax is correct, the jar file is created inside the scala subfolder. Else, errors are reported and must be worked upon. The “src” folder from our code base along with the jar file is placed in the “bin” folder inside the “spark-3.0.0-preview2-bin-hadoop2.7” folder.

“Spark submit” command is run inside the bin folder and the output files are compared with the sample output files given in the project.

Results for each task is as follows :

ST_Contains() & ST_Within() : 4 output folders are generated numbered from 0 through 3. Each folder has a success file and the actual value of the computation. Each of the four results correspond to the RangeQuery, RangeJoinQuery, DistanceQuery and DistanceJoinQuery and all the values - 4, 7612, 302, 123362 were consistent with the values in the example output.

Hot Zone Analysis & Hot Cell Analysis: 2 output folders are generated numbered from 0 through 1. Each folder has a success file and the actual value of the computation csv file. The folder with name output0 has the result of hot zone analysis which is generated in a csv file which contains all zones with their count, sorted by "rectangle" string in an ascending order. Similarly the folder output1 has the result of hot cell analysis which is generated in a csv file which contains the top 50 hot cells sorted by their G score in a descending order.