Project Report for CSE 511: Data Processing at Scale, Project #1

Harsh Sanjaykumar Nagoriya
Arizona State University
Email: hnagoriy@asu.edu

Krushali Shah
Arizona State University
Email: kshah51@asu.edu

Savan Dixesh Doshi
Arizona State University
Email: sdoshi7@asu.edu

***Abstract*—The collecting and processing of spatio-temporal data have increased dramatically in recent years. Processing them and producing meaningful information from them remains a major challenge for businesses and tech organizations. The technique of extracting a data subset from a map layer by dealing closely with the map data is known as a spatial query. Data is stored in attribute tables and feature/spatial tables inside a spatial database. The goal of this project is to execute various geographical queries for a big peer-to-peer taxi cab company's operational (day-to-day) and strategic (long-term) decisions.**

I. Introduction

Geospatial data, often known as spatial data, is a resource with a geographic component. In other terms, the information in this type of data set often comprises 2 or more dimensional attributes such as graph coordinates. The traditional Structured Query Language does not support processing such data. A spatial query mechanism shall be implemented in this case. The level of geographical data collected has increased dramatically in recent years. The most significant characteristics of a cab company are the client and driver locations. As a result, they must strive to query and analyze such massive volumes of spatial data in order to achieve high performance and low latency.

II. Development Approach

As previously stated, a spatial query is a form of query that geodatabases and spatial databases support. Points, lines, and polygons are used in the queries, which makes them different from standard SQL queries. Because the database is vast and mostly unstructured, Big Data software programs such as SparkSQL, Hadoop, and others are required.

We proceeded by setting up the environment and installing all of the necessary applications in order to begin developing. There were several concerns with the development environment that needed to be handled. To fulfill the aim, we created a development environment using software and frameworks that include Java 1.8, Scala 2.13 and Spark 3.2. We learned Scala syntax by following several internet lessons because we were unfamiliar with it. The next stage was to brainstorm how logic could be applied to this problem. Following that, the algorithm was written into the scaffolding code. To create the jar file and execute the spark-submit command, we utilized sbt-assemble.

Lastly, we compared our output to a sample output provided to us.

III. Requirements

As stated above, to compile the scaffolding code, we needed to have following resources in our machines.

A. Machine requirements

We used the machines with following specifications in order to compile and test the code.

1) MacBook Air: 8-Core CPU, 7-Core GPU, 8GB Unified Memory, 256GB SSD Storage

2) MacBook Pro: 8-Core CPU, 14-Core GPU, 16GB Unified Memory, 512GB SSD Storage

B. Software requirements

We used the following software configurations in order to compile and test the code.

1) Java: Version 1.8.0_321

2) Scala: Version 2.13.8

3) Spark: Version 3.2.1

4) IntelliJ Idea community edition: Version 2021.3.3

5) Visual Studio Code: Version 1.66

So as to keep all these things working, we set proper environment variables.

IV. Workdone

In SparkSQL, we created two user-defined functions: '`ST_Contains`' and '`ST_Within`'. On the spatio-temporal dataset, we used these to execute the following four spatial queries:-

A. *Range Query*

The given query would use the user defined function 'ST_Contains'. This will find all the points from the given set of points P within the given query rectangle R.

B. *Range join query*

The given query would use the user defined function 'ST_Contains'. A set of Rectangles R and a set points S would e given. The task of this query would be to find all the (Point,Rectangle) pairs such that the point would lie inside the rectangle.

C. *Distance query*

The given query would use the user defined function 'ST_Within'. A point P and distance D would be given as an input the task of this query would be to return the set of points that lie within the distance D from point P.

D. *Distance join query*

The given query would use the user defined function 'ST_Within'. Two set of points S1 and S2 would be given along with distance D in km. This query would return all the pairs (s1,s2) such that s2 would be within distance d from s1.

Figure 1 shows the system components for the solution of given problem statement. The queryloader executes the spatial queries which in turn uses user defined functions 'ST_Contains' and 'ST_Within' to get the required information.

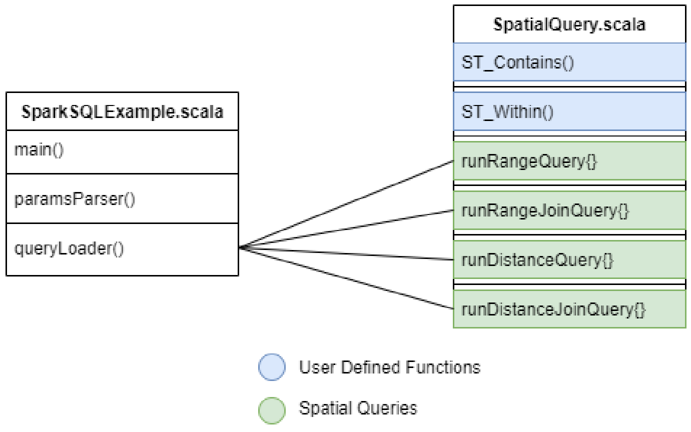


Fig. 1. System Components

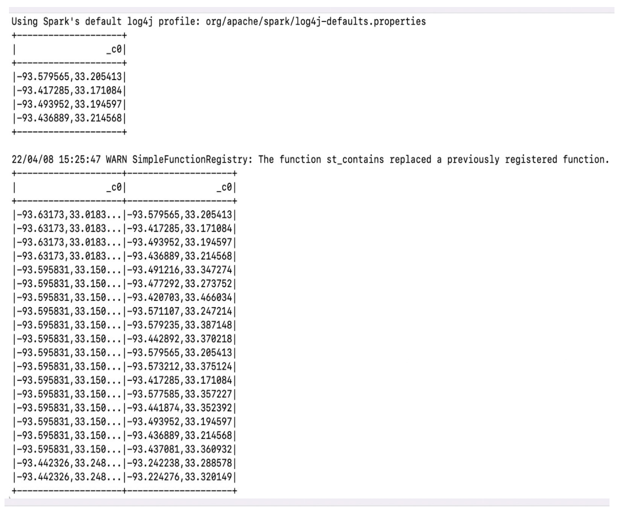
- 'ST_Contains' function takes point P and rectangle R as parameter and returns a boolean result. It's task is to check whether or not p lies inside the rectangle R
- 'ST_Within' takes two points p1, p2 and distance D as input. It calculates the Euclidean distance between p1 and p2 and returns true if the distance calculated is less than d. The following is the equation for calculating the Euclidean distance.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

These two functions form the basis for running all the queries namely, Range Query, Range Join Query, Distance Query and Distance Join Query as mentioned in the Requirements section.

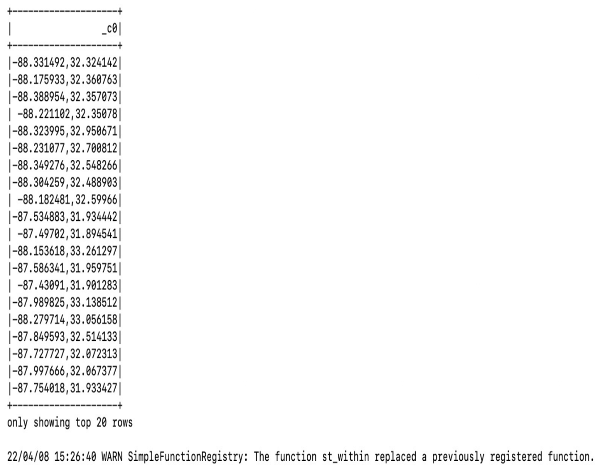
V. Solution Verification

Our team had compared our results with the sample output provided in the Project Template. We did run the provided command with the jar file after writing our code. Hence, all the modifications in the code would be incorporated in the Jar file. The following **Figure 2** shows the obtained output from our solution.



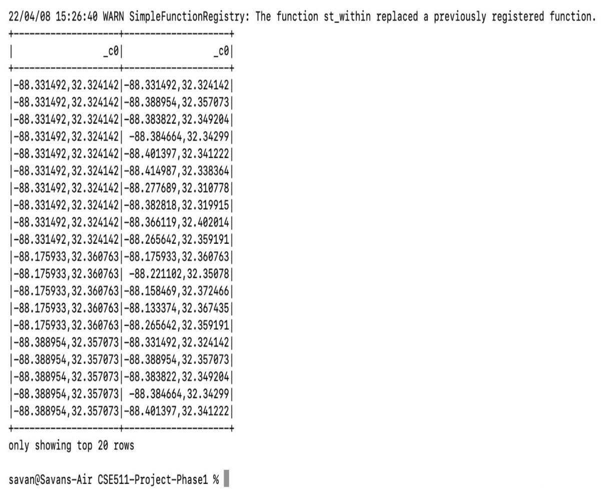
```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
+--------------------+
|                 _c0|
+--------------------+
|-93.579565,33.205413|
|-93.417285,33.171084|
|-93.493952,33.194597|
|-93.436889,33.214568|
+--------------------+

22/04/08 15:25:47 WARN SimpleFunctionRegistry: The function st_contains replaced a previously registered function.
+--------------------+--------------------+
|                 _c0|                 _c0|
+--------------------+--------------------+
|-93.63173,33.0183...|-93.579565,33.205413|
|-93.63173,33.0183...|-93.417285,33.171084|
|-93.63173,33.0183...|-93.493952,33.194597|
|-93.63173,33.0183...|-93.436889,33.214568|
|-93.595831,33.150...|-93.491216,33.347274|
|-93.595831,33.150...|-93.477292,33.273752|
|-93.595831,33.150...|-93.420703,33.466034|
|-93.595831,33.150...|-93.571107,33.247214|
|-93.595831,33.150...|-93.579235,33.387148|
|-93.595831,33.150...|-93.442892,33.370218|
|-93.595831,33.150...|-93.579565,33.205413|
|-93.595831,33.150...|-93.573212,33.375124|
|-93.595831,33.150...|-93.417285,33.171084|
|-93.595831,33.150...|-93.577585,33.357227|
|-93.595831,33.150...|-93.441874,33.352392|
|-93.595831,33.150...|-93.493952,33.194597|
|-93.595831,33.150...|-93.436889,33.214568|
|-93.595831,33.150...|-93.437081,33.360932|
|-93.442326,33.248...|-93.242238,33.288578|
|-93.442326,33.248...|-93.224276,33.320149|
+--------------------+--------------------+
```



```
+--------------------+
|                 _c0|
+--------------------+
|-88.331492,32.324142|
|-88.175933,32.360763|
|-88.388954,32.357073|
| -88.221102,32.35078|
|-88.323995,32.950671|
|-88.231077,32.700812|
|-88.349276,32.548266|
|-88.304259,32.488903|
| -88.182481,32.59966|
|-87.534883,31.934442|
| -87.49702,31.894541|
|-88.153618,33.261297|
|-87.586341,31.959751|
| -87.43091,31.901283|
|-87.989825,33.138512|
|-88.279714,33.056158|
|-87.849593,32.514133|
|-87.727727,32.072313|
|-87.997666,32.067377|
|-87.754018,31.933427|
+--------------------+
only showing top 20 rows

22/04/08 15:26:40 WARN SimpleFunctionRegistry: The function st_within replaced a previously registered function.
```



```
22/04/08 15:26:40 WARN SimpleFunctionRegistry: The function st_within replaced a previously registered function.
+--------------------+--------------------+
|                 _c0|                 _c0|
+--------------------+--------------------+
|-88.331492,32.324142|-88.331492,32.324142|
|-88.331492,32.324142|-88.388954,32.357073|
|-88.331492,32.324142|-88.383822,32.349204|
|-88.331492,32.324142| -88.384664,32.34299|
|-88.331492,32.324142|-88.401397,32.341222|
|-88.331492,32.324142|-88.414987,32.338364|
|-88.331492,32.324142|-88.277689,32.310778|
|-88.331492,32.324142|-88.382818,32.319915|
|-88.331492,32.324142|-88.366119,32.402014|
|-88.331492,32.324142|-88.265642,32.359191|
|-88.175933,32.360763|-88.175933,32.360763|
|-88.175933,32.360763| -88.221102,32.35078|
|-88.175933,32.360763|-88.158469,32.372466|
|-88.175933,32.360763|-88.133374,32.367435|
|-88.175933,32.360763|-88.265642,32.359191|
|-88.388954,32.357073|-88.331492,32.324142|
|-88.388954,32.357073|-88.388954,32.357073|
|-88.388954,32.357073|-88.383822,32.349204|
|-88.388954,32.357073| -88.384664,32.34299|
|-88.388954,32.357073|-88.401397,32.341222|
+--------------------+--------------------+
only showing top 20 rows

savan@Savans-Air CSE511-Project-Phase1 %
```

Fig. 2. Received Output

VI. Conclusion

The data of a town or city's geographical boundaries, as well as a set of points P indicating clients who want taxi cab service, were processed in this project. The Apache Spark framework was used to overcome the problem of large-scale geographic data computation. As a result, Apache Spark proved to be an effective choice for achieving high performance. We were able to properly analyze geospatial data primarily with the use of SparkSQL APIs on Scala.
We designed an algorithm that extracts essential information from a dataset using Apache Spark and Scala, which can then be used to make operational and strategic decisions. There were also a few specific spatial queries built. Our developed system gives the client statistically significant geographic areas to help them plan their business and better serve their customers.

Contributions

We, all three students, have worked equally and all together with no conflicts of interest.

Acknowledgment

We, authors, would like to thank Dr. Zhichao Cao (Instructor), Akkamahadevi Hanni (Teaching Assistant), Rithvik Chokkam (Grad Service Assistant) Arizona State University for their constant encouragement towards the realization of this work.

References

[1] Spark SQL - Quick Guide, Available at: https://www.tutorialspoint.com/spark_sql/spark_sql_quick_guide.htm
[2] Spark Overview, Available at: https://spark.apache.org/docs/latest/
[3] CSE512 Project Phase2 Template, Available at: https://github.com/jiayuasu/CSE512-Project-Phase2-Template
[4] Scala Tutorial Full Course, Available at: https://www.youtube.com/watch?v=i9o70PMqMGY
[5] GeoSpark SQL: An Effective Framework Enabling Spatial Queries on Spark, Available at: https://doi.org/10.3390/ijgi6090285
[6] Download & install apache-spark on MacOS (Big Sur, Monterey, Catalina, Mojave) via Homebrew / brew, Available at: https://www.youtube.com/watch?v=6c1uP_UbuBg