Portfolio Report for CSE 511: Data Processing at Scale

Harsh Sanjaykumar Nagoriya
Ira A. Fulton Schools Of Engineering — School Of Computing And Augmented Intelligence
Arizona State University
Email: hnagoriy@asu.edu

***Abstract*—In recent years, the collection and processing of spatiotemporal data has risen rapidly. For businesses and IT companies, processing them and extracting useful information from them remains a huge hurdle. A spatial query is a method of retrieving a data subset from a map layer by working directly with the map data. Within a spatial database, data is stored in attribute tables and feature/spatial tables. The purpose of this project is to run multiple geographical queries for functional and economic choices for a large peer-to-peer taxi cab firm.**

I. Introduction

A resource with a geographic component is geospatial data, also known as spatial data. In other words, this sort of data collection frequently contains two or more dimensional properties, such as graph coordinates. Such data cannot be processed using the standard Structured Query Language. In this instance, a spatial query method must be built. In recent years, the amount of geographical data acquired has surged. The customer and driver locations are the most important elements of a cab company. As a result, they must aim for high performance and low latency while querying and analyzing such large amounts of geographical data. As stated above, this sort of data collection frequently contains two or more dimensional properties, such as graph coordinates. In this situation, two spatial query techniques will be implemented:

1) Hot zone analysis
2) Hot cell / Hot-spot analysis.

II. Description of solution

A. Project Phase 1

1) Description: ‘‘`ST_Contains`’’ and ‘`ST_Within`‘ are two SparkSQL user-defined functions. The following four spatial queries were run on the spatio-temporal dataset using these.

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.

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.

Distance query: The provided query would make use of the user-defined function ‘`ST_Within`‘. The purpose of this query would be to return the set of points that fall within the distance D from point P, given a point P and a distance D.

Distance join query: The user written function ‘`ST_Within`‘ would be used in the distance join query. There would be two sets of points S1 and S2, as well as a distance D in kilometers. This query will return all pairs (s1, s2) in which s2 is within d distance of s1.

The system components for solving the given issue statement are shown in **Figure 1**. The queryloader runs the spatial queries, which then use the user-defined functions ‘‘`ST_Contains`’’ and ‘`ST_Within`‘ to get the data.

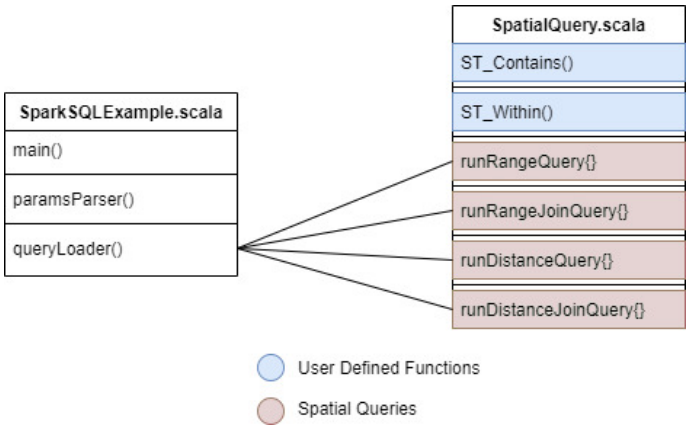


Fig. 1. System Components: Reference: Project report submitted by our team

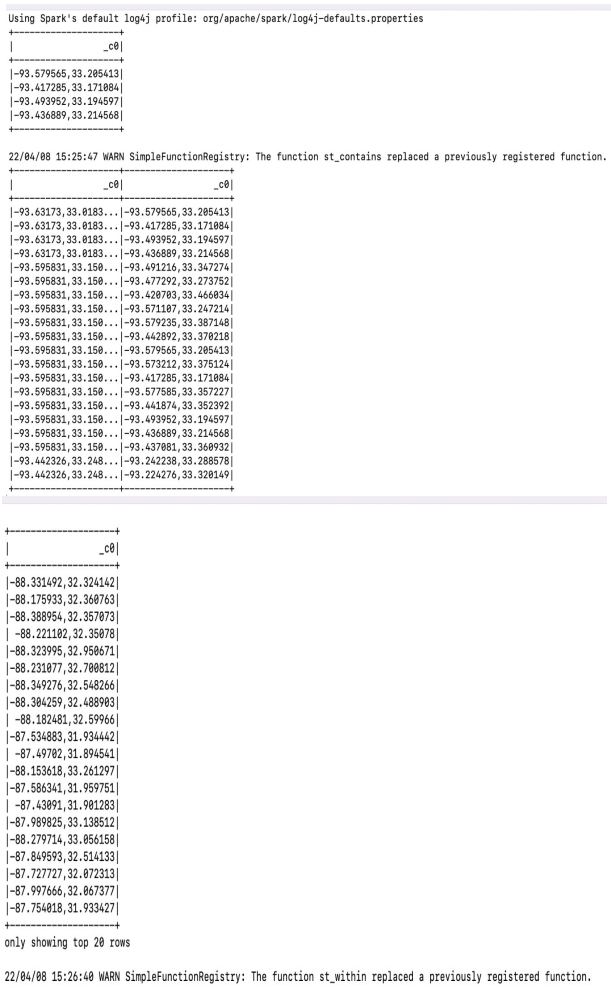
The ‘‘`ST_Contains`’’ function accepts two parameters: point P and rectangle R, and returns a boolean result. Its function is to see if p is contained within the rectangle R.

‘`ST_Within`‘ accepts two points p1 and p2 as input, as well as a distance D. It determines the Euclidean distance between p1 and p2 and returns true if the estimated distance is smaller than d. The equation for determining the Euclidean distance is as follows. Reference: Project report submitted by our team.

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

These two functions provide the base for all of the queries as mentioned, including Range Query, Range Join Query, Distance Query, and Distance Join Query.

2) Solution Verification: Our team compared our results to the Project Template's example output. After finishing our code, we ran the given command using the jar file. As a result, all changes to the code will be reflected in the Jar file. The outcome from our solution is shown in the **Figure 2** below.



```
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|
|-87.997666,32.867377|
|-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 Reference: Project report submitted by our team

B. Project Phase 2

1) Description: The Hot Zone analysis and the Range Join Query from Phase 1 have the same criteria. There would be a set of Rectangles R and a set of points S. The first step in implementing this query would be to discover all the (Point, Rectangle) pairings where the point is inside the rectangle, and then use a GROUP BY clause to aggregate the result for each rectangle to determine its hotness. We went over the offered template to understand the provided inputs, pre-processing, and desired output in order to grasp the assignment for Hot Cell Analysis. The template had cells with two geographical and one temporal feature, such as latitude, longitude, and month day. To get the count of visits for each cell, the initial stage in the strategy was to apply aggregate operations on the pickup information data frames, such as GROUP BY and COUNT. The mean and standard deviation would be calculated for the Getis-ord statistic. Following that, we estimated the number of trips made by nearby cells for each cell. to determine

$$\sum_{j=1}^{n} W_{i,j}$$

(which is simply the number of neighbors for a cell) We calculated the number of neighbors for a cell based on its geometric placement. If the number of dimensions on boundary values is zero, the number of neighbors would be 27. Similarly, the neighbor count for cells 1, 2, and 3 would be 18,12, and 8, correspondingly. Is then the

$$\sum_{j=1}^{n} W_{i,j}$$

As each neighbor's weight is equal, the number of trips in surrounding cells is counted. Now that we have all of the relevant tables, we can create Getis-ord statistics and look at the numerator and denominator. The top 50 cells are then returned as a scala dataframe after sorting the result in decreasing order based on the G—score value.

2) Solution Verification: As an outcome, a Data-Frame with 50 cells and three locations for the cells with the greatest Getis-Ord statistics is constructed. We debugged the code, revealing intermediate data frames from the previous part's calculations as well as statistics like Mean and Standard Deviation. To make the JAR file, we used the textbfsbt clean assembly command. To launch the JAR file that we used to validate the findings, type textbfSpark-submit. As a result, a CSV file is created in the desired output directory. The input test data for both Hot Cell Analysis and Hot Zone Analysis was given in CSV format. We compared our output to the actual CSV-formatted result file. **Fig 3** shows the obtained output

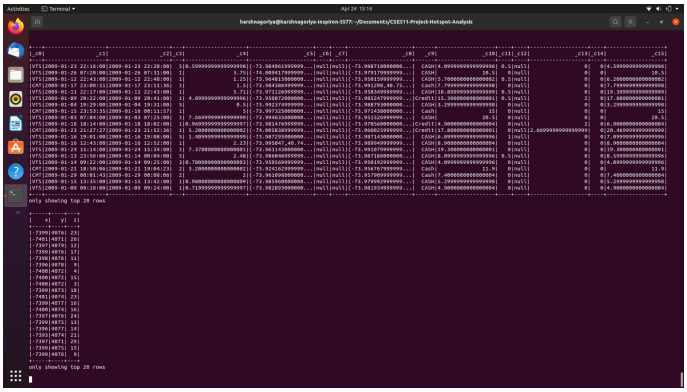


Fig. 3. Output

III. Results and Conclusion

In this project, the physical borders of a town or city, as well as a collection of points P denoting clients who demand taxi cab service, were processed. To solve the challenge of large-scale geographic data computing, the Apache Spark framework was deployed. As a result, Apache Spark has shown to be a viable option for high-performance computing. With the help of SparkSQL APIs on Scala, we were able to appropriately analyze geographical data. Using Apache Spark and Scala, we created an algorithm that pulls vital information from a dataset, which can subsequently be utilized to make operational and strategic choices.

A few particular spatial searches were also implemented. Our solution provides statistically significant geographic areas to the client, allowing them to better manage their company and service their consumers.

IV. My Contribution

Now, in the parts that follow, I'll go over the specific tasks that I completed during the project.

The first major step I had to do in phase 1 was to install the necessary tools and environment to execute even the scaffolding code. This required the installation of Java 1.8, Apache Spark 2.4.7, and Hadoop 2.7. After successfully installing these apps, I had to set up the path environment variables. I was able to complete things in a timely way since I followed multiple tutorials and TA sessions. After that, my project environment was ready to use.

The following stage was to learn how to program in Scala. To accomplish so, I used several online resources as well as the official documentation for the Scala programming language. After that, I started working on the code aspect of phase 1.

In phase 1, we primarily had to build two functions: `ST_Within` and `ST_Contains`. We split these tasks into two sub-teams, one with myself and Savan and the other with Krushali. I was a member of the subteam that was creating the ST Conatins function. It was a straightforward function that checked if a given point was inside or outside of a rectangle. True was returned if the point was inside the rectangle; otherwise, False was returned. As a result, I put the pseudocode for this method to the .scala file after writing it.

After the `ST_Contains` function was implemented, the range query and range join query were performed. The other participants completed the second half of phase 1, after which we merged the code and double-checked it for accuracy.

In phase 2 of the project, we had to accomplish two tasks: hotzone analysis and hotcell analysis. Like phase 1, we divided this step into two sub-teams. The Hotzone analysis function was designed by Krushali and myself. In Hotzone Analysis, the hotness of a geographic region was determined by calculating the number of cab rides that began there. The supplied data sets of rectangles and taxi journeys were used to produce a heatmap. In increasing order, the final results were given.

After the Hotzone Analysis was done, Savan did the Hotcell Analysis. Following that, the two codes were merged. In all phases, we double-checked our codes against the given testcases to verify that there were no problems. We also compared our results to the test results provided. At the end of each phase, we had to deliver a project report, so I worked on the conceptual half of the project, which related to my coding side.

V. Lessons Learned

I was able to explore many new tools and technologies, such as Spark, Scala, Hadoop, Geospatial queries, and others, since I was fascinated by contemporary advancements in data technology. These tools and technology aided me in mastering cutting-edge data processing technologies, which would make a better path towards my career.

VI. Other Team Members

1) Savan Dixesh Doshi — Email: sdoshi7@asu.edu
2) Krushali Shah — Email: kshah51@asu.edu

VII. Acknowledgement

I'd like to express my gratitude to Savan Doshi and Krushali Shah for their unwavering support, cooperation, and encouragement throughout the project's teamwork journey, which proved crucial to the project's success. Without their support and co-operation this project would not have materialized. I would also 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] Project Report for CSE 511: Data Processing at Scale, Project #1, Submitted to Arizona State University canvas by our team Group 22
[2] Project Report for CSE 511: Data Processing at Scale, Project #2, Submitted to Arizona State University canvas by our team Group 22
[3] Spark SQL - Quick Guide, Available at: https://www.tutorialspoint.com/spark_sql/spark_sql_quick_guide.htm
[4] Jia Yu, "CSE512-Project-Hotspot-Analysis-Template", https://github.com/jiayuasu/CSE512-Project-Hotspot-Analysis-Template
[5] Jia Yu, "CSE512-Project-Phase2-Requirement", https://github.com/jiayuasu/CSE512-Project-Phase2-Template

[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
[7] ACM SIGSPATIAL Cup 2016, Problem Definition , Available at: http://sigspatial2016.sigspatial.org/giscup2016/problem
[8] ACM SIGSPATIAL Cup 2016, Submission and Evaluation , Available at: http://sigspatial2016.sigspatial.org/giscup2016/submit
[9] ACM SIGSPATIAL Cup 2016, Submission and Evaluation , Available at: http://sigspatial2016.sigspatial.org/giscup2016/submit
[10] Spark Overview, Available at: https://spark.apache.org/docs/latest/
[11] Scala Tutorial Full Course, Available at: https://www.youtube.com/watch?v=i9o70PMqMGY
[12] GeoSpark SQL: An Effective Framework Enabling Spatial Queries on Spark, Available at: https://doi.org/10.3390/ijgi6090285