Systems Documentation Report

CSE511: Data Processing at Scale

Spring 2021

Group 3

Siddharth Kale: (ASU ID: 1219495147)
Rakhi Agrawal: (ASU ID: 1219782889)
Karan Dabas: (ASU ID: 1219496811)
Rohan Narayana Kanuri: (ASU ID: 1219398531)
Ayushi Shekhar: (ASU ID: 1219510799)
Amit Kumar Sinha: (ASU ID: 1219500399)

1. Introduction

The amount of spatial data is increasing at an alarming pace. Earth science datasets, geotagged social media, car trajectories, and sensor measurements are examples of such information. Furthermore, everything we do on our smartphones and wearable gadgets, such as booking a cab or making a dinner reservation, leaves geospatial digital data. Moreover, several cities are starting to harness the power of sensors to track the urban climate. For example, the city of Chicago has begun to add sensors at road intersections to track the landscape, air quality, and traffic. Making use of such spatial data can be useful for a variety of technologies that have the potential to change science and society.

Existing relational DBMSs support a wide range of spatial data types, operators, and index constructs for processing spatial operations, but most of them struggle when scaling up. Latest experiments, such as SpatialHadoop and HadoopGIS for Hadoop MapReduce, have used distributed data processing frameworks to address this problem and scale out spatial operations. While these methods are highly scalable, they have slow run time efficiency, which the consumer would not tolerate.

Resilient Distributed Datasets (RDDs), on the other hand, are arrays of entities partitioned through a cluster of machines provided by Apache Spark. Each RDD is constructed using parallelized transformations (filter, join, or groupBy), which can be traced back to recover the RDD results. RDDs in memory empower Spark to perform better than existing models (MapReduce).

2. Problem Statement

A big peer-to-peer taxicab company has asked us to assist them in evaluating their spatial data and drawing concrete conclusions from it. Geographic statistics, as well as real-time location data of their customers, are included in the data.

The firm's final deliverables shall have the following features:

a. Given a geographical border in a town or city, quantify the number of customers who used the client's app to order a taxicab service.
b. For a set of customers who demanded a taxicab service in the city and a geographical boundary within the city, calculate [boundary – customer] pairs so that a customer is within the radius controlled by the boundary.
c. Find all the customers who have requested a cab who is within a distance D of the former, given the location of a customer who has requested a cab and a distance D.
d. Determine if two customers are within D miles of each other given their current locations and a distance D.
e. Hot Zone Analysis: Using boundary data and consumer data, obtain customer intensity and profitable geo-location analysis.
f. Hot Cell Analysis: Using spatial data derives statistically important hotspots.

The above project criteria were met with the use of Geo-Spatial Analysis techniques.

3. Geospatial Data Analysis using Spark SQL

With the Spark architecture, the team used the Scala programming language to create spatial queries and research functions. Spark is a "quick and general engine for large-scale data processing," according to Apache. Spark provides a DataFrame abstraction for expressing transformations, filters, and aggregations, as well as effectively executing the computation through a distributed collection of tools.

Apache Spark is an in-memory computing cluster system. Spark introduces a new storage abstraction known as resilient distributed datasets (RDDs), which are arrays of entities distributed through a cluster of computers. Each RDD is constructed using parallelized transformations (filter, join, or groupBy), which can be traced back to recover the RDD results.

SparkSQL is a standalone Spark module that handles hierarchical data processing. It offers a higher-level abstraction over Spark RDD known as DataFrame. A DataFrame is organized in the format of a table, complete with column detail. The SparkSQL optimizer uses structure details to refine queries. SparkSQL accepts two types of APIs: (1) DataFrame API, which manipulates DataFrame using Scala and Java APIs, and (2) SQL API, which manipulates DataFrame explicitly using SQL statements.

Unfortunately, Spark and SparkSQL do not accept spatial data or spatial operations natively. As a result, users must perform the time-consuming process of writing their own spatial data discovery jobs on top of Spark.

4. Project Tasks: Description and Distribution

The first step in our project was to determine the objectives and devise a strategy for completing them. Some tasks were assigned to individual members while others were completed by everyone on the team.

Besides, we designed a timetable for our project and made sure that each step of implementation was completed before moving on to the next. Each team member was assigned a specific set of roles and responsibilities.

Task	Description	Members
Setup Apache Spark and Spark SQL	Each member had to install Apache Spark and Spark SQL on their machine and being acquainted with the software needed to complete the assigned tasks.	Everyone
Understand basics of Apache Spark SQL and Spatial	Regardless of the tasks, everyone on the team was expected to become acquainted with Spark SQL, which was required for phase 1, and spatial queries, which were required for phase 2. Everyone was responsible	Everyone

Queries	for reading and comprehending the architecture and functionalities of Apache Spark before disseminating information among team members via daily meetings.	
Phase 1	The objective of this phase of the project was to write two Spark SQL User Defined Functions, ST_ CONTAINS and ST_ WITHIN, and use them to perform four spatial queries. We implemented these two functions using the provided template. All members of our group oversaw writing the functions and a part of the group was also in charge of writing the different queries and then debugging and running the code.	ST_Contains: Karan, Rakhi & Amit ST_Within: Siddharth, Rohan & Ayushi Debug and Run: Rohan, Ayushi, Amit
Phase 2	In this phase, we had to perform spatial hot spot analysis on the provided dataset. Hot Zone analysis and Hot Cell analysis were the two hotspot analysis activities that have been defined. We used a similar strategy to distribute the tasks among different members. We made two pairs of team members, one pair working on Hot Zone analysis and the other pair working on Hot cell analysis. The remaining members were responsible for analyzing and evaluating the results.	Hot Zone Analysis: Rakhi, Karan, and Siddharth Hot Cell Analysis: Ayushi, Rohan, Amit Debug and Run: Rakhi, Karan, and Siddharth
Debugging and system testing	Each team member oversaw debugging and system testing. This was done to ensure that the code was stable and error-free. During this phase of the project, regular meetings were scheduled so that each team member could express his or her thoughts and test cases, allowing everyone on the team to understand and develop the consistency of the test cases and the outcome.	Everyone
System Documentation Report	This report discusses the work performed on the project and contains enough information that any programmer can easily port this approach to another programming environment. Every team member was given the task of writing individual parts of the	Everyone

	document so that everyone could work in parallel to successfully compose the document with the least amount of time and effort. All oversaw each other's work.	

5. Software Architecture

In approaching the project and its objectives, the team used the SDLC waterfall model. The waterfall model describes the software development process as a continuous, sequential process. This ensures that the next stage of the project will begin only after the previous stage has been completed. The stages of the waterfall model do not overlap.

The software was developed in two stages. The first phase package includes functions for performing simple spatial queries, while the second phase package includes spatial hot spot analysis activities. Both packages are intended to be deployed on a Spark cluster and to store test results in.csv format locally.

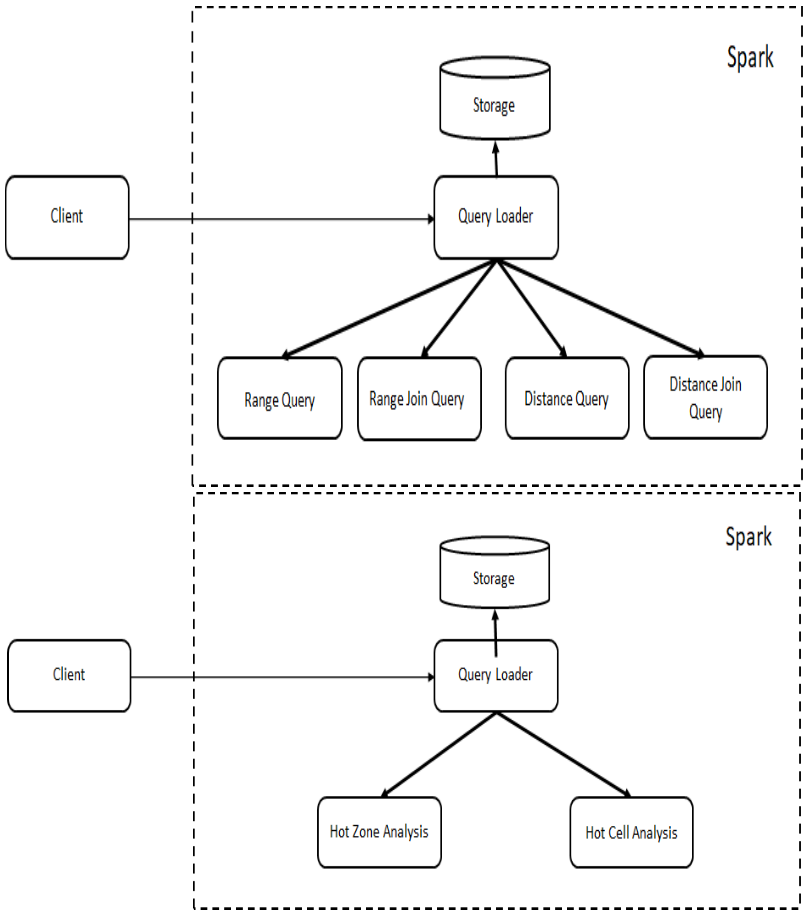


Figure: Main components of Phase 1 and Phase 2

The ‘Query Loader' component, which is visible to the client, parses parameters in the format '<Output Location> <Query Name> <Query Parameters>'. The Query Loader element then executes the corresponding query and saves the results to the output location specified.

6. Code Documentation

Below is the API documentation for all the functions that we have implemented in the project. Each entry describes the function in brief, the input parameters, and the output it is generating.

Queries for Phase 1, that are defined in the SpatialQuery.scala file:

a. **RangeQuery:** Find all the points inside R given a query rectangle R and a set of points P.

Arguments:

- i. PointData: String filepath to a CSV containing points in the format ‘x, y’.
- ii. RectData: String representation of points that make up a rectangle in the format ‘x_1, y_1, x_2, y_2’.

Output: A csv containing the total count of points in PointData that are within the specified rectangle.

b. **RangeJoinQuery:** Find all (point, rectangle) pairs where the point is inside the rectangle given a set of rectangles R and a set of points P.

Arguments:

- i. PointData: String filepath to a csv containing points in the format ‘x, y’.
- ii. RectData: String filepath to a csv containing points that make up a rectangle in the format ‘x_1, y_1, x_2, y_2’.

Output: A csv containing the total count of point, rectangle pairs of points in PointData that are within the specified rectangles in RectData.

c. **Distance Query:** Find all points within a distance D from P, given a fixed-point position P and a distance D (in kilometers).

Arguments:

- i. PointData: String filepath to a CSV containing points in the format ‘x, y’.
- ii. PointString: String representation of a point ‘x, y’.
- iii. Distance: Double distance in kilometers.

Output: A CSV containing the total count of points in PointData that are within the specified distance of PointString.

d. **DistanceJoin Query:** Find all (p1, p2) pairs such that p1 is within a distance D from p2 given two sets of points P1 and P2 and a distance D (in kilometers) (i.e., p1 belongs to P1 and p2 belongs to P2).

Arguments:

- i. PointData1: String file path to a CSV containing points in the format ‘x, y’.
- ii. PointData2: String file path to a CSV containing points in the format ‘x, y’.
- iii. Distance: Double distance in km

Output: a CSV containing the total count of points in PointData1 that are within the specified distance of points in PointData2.

Next is the documentation for Phase 2. All the code is implemented in the file HotzoneAnalysis.scala, HotzoneUtils.scala, HotCellAnalysis.scala and HotcellUtills.scala.

a. **HotZoneAnalysis:** A range join procedure on a rectangle dataset and a point dataset will be needed for this task. The number of points inside a rectangle would be determined for each rectangle. The hotter the rectangle, the more points it contains. So, the aim of this challenge is to figure out how hot all the rectangles are.

Arguments:

i. PointData: String file path to a CSV containing points in the format 'x, y'.
ii. RectData: String representation of points that make up a rectangle in the format 'x_1, y_1, x_2, y_2'.

Output: A csv containing the joined results of PointData and RectData grouped by the count of points in the rectangles sorted by the points that make up the rectangle.

b. **HotCell Analysis:** This task would use Apache Spark to apply spatial statistics to spatio-temporal big data to find statistically meaningful spatial hot spots.

Arguments:

i. tripData: String file path to a CSV (delimited by '$;$') containing data representing pickup locations and time, where the 'x, y' coordinate is at position 5 and the timestamp at position 1.

Output: A CSV containing the 'x, y, z' coordinates of pickup instances ordered by their getis-ord statistic value where the 'x, y', coordinates are positions on a two-dimensional plane, and the z-coordinate is the day of the month the pickup occurred.

7. Setup Guide

We setup both Spark and IntelliJ, for the debugging we used IntelliJ as they have a very user-friendly interface to debug the code with many options such as evaluating expression, etc. to aid with the debugging. Spark we mainly used to check if the code is working well with spark submit, as this was the requirement for the project and to run the code quickly to test different inputs and corresponding outputs.

a. Setting up Spark and Scala environment

We used Spark and Scala along with the Java dependencies to run the geospatial code. Below are the steps that we have followed to set up the Spark cluster.

i. Java Installation

- Check the version of Java installed on the system.
- If the Java version is not 1.8.x, install Java.

ii. **Scala Installation**

- Check the version of Scala installed on the system.
- If Scala is not installed, the following steps must be followed:
    - Download Scala tar file.
    - Extract Scala file by using the command -$ tar xvf scala-2.11.6.tgz
    - Move the file to usr/local/scala
    - $ su –
    - Password:
    - # cd /home/Hadoop/Downloads/
    - # mv scala-2.11.6 /usr/local/scala
    - # exit
- Setup Path
    - $ export PATH = $PATH:/usr/local/scala/bin

iii. **Spark Installation**

- Download Spark. We have used spark-3.1.1-bin-hadoop2.7
- Extract the files using the command $ tar xvf spark-3.1.1-bin-hadoop2.7.tgz
- Move the files to the appropriate folder.
    - $ su –
    - Password:
    - # cd /home/Hadoop/Downloads/
    - # mv spark-3.1.1-bin-hadoop2.7 /usr/local/spark
    - # exit
- Export the path of Spark.
    - export PATH = $PATH:/usr/local/spark/bin
- Use the following command for sourcing the ~/.bashrc file
    - $ source ~/. bashrc
- Verify that the Spark is installed correctly.
    - $spark-shell

iv. **Sbt Installation**

- echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
- curl -sL "https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x2EE0EA64E40A89B84B2DF73499E82A75642AC823" | sudo apt-key add.
- sudo apt-get update.
- sudo apt-get install sbt.

b. Running the code on the Spark Cluster

The following steps are executed in the manner they are presented to ensure that our code is working properly, and we are generating all the files that are required for successful submission.

- Move the entire code to the production server cluster.
- CD into the project directory
- Here we must create a JAR file that will be submitted to the Spark cluster as a job.
- For creating the jar file, we are using sbt.
- Run $sbt clean-assembly command to generate the JAR file.
- The jar file would be created inside the target/scala2.11 folder.
- The name of the file would be exactly like your root folder.
- Then we are using the spark-submit method to submit this file to the spark cluster.
- We must also provide some command-line arguments to run the file.
- Below is a sample spark-submit command:
    - spark-submit CSE512-Project-Phase2-Template-assembly-0.1.0.jar result/output rangequery src/resources/arealm10000.csv -93.63173,33.0183, -93.359203,33.219456 rangejoinquery src/resources/arealm10000.csv src/resources/zcta10000.csv distancequery src/resources/arealm10000.csv -88.331492,32.324142 1 distancejoinquery src/resources/arealm10000.csv src/resources/arealm10000.csv 0.1.
- The outputs would be in a CSV file, stored in the directory depending on the command line argument, in the above case the file is generated in result/output.

c. Installing IntelliJ and dependencies

- Download and install the latest IntelliJ Idea Community Edition from the Jetbrains website. The version details are Version: 2021.1, Build: 211.6693.111.
- Download and install JDK for java 8 from the Oracle website. The version details are jdk1.8.0_23 (Note: As this is an old version, might need to create an Oracle account before downloading)
- Download Hadoop winutils from https://github.com/steveloughran/winutils.
- Set an environment variable as the directory containing winutils from the previous step.
- Open Intellij and install the plugin for Scala, use Ctrl+Alt+S and open the plugins page, search for Scala, and install, restart IntelliJ after this to use the plugin.

d. Compiling and Execution

- The main classes for Phase1 and Phase2 are SparkSQLExample.scala and Entrance.scala respectively, first uncomment ". master ("local [*]")" in these files
- Make sure there are no errors in the build.sbt and dependencies have been installed.
- Go to Run -> Edit Configurations and check if the main classes are accurate. Add the arguments as mentioned in the project description or based on testing strategy, also check if "Include dependencies with Provided scope" is checked.
- Now Apply and run the code to check, alternatively you can put breakpoints and debug the code as well.
- On successful completion the output will now be generated in the folder mentioned in the parameters.

8. Testing and Evaluation

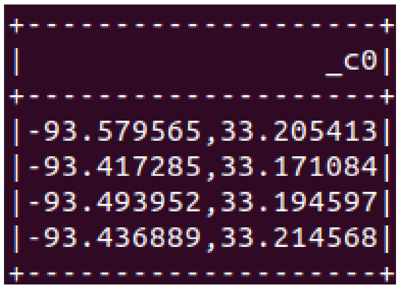
For both phases, initially we completed the development on IntelliJ as this provides easy debugging tools and upon completion different test cases were formed, and we checked and verified the results for each of them. Arguments were passed through the command line to execute the different evaluations, as discussed in the previous section.

For both the phases, we were provided the sample command-line argument(input) and its corresponding outputs on the GitHub page of the project. We tested our code against these sample inputs and matched the outputs to these corresponding given outputs. We successfully completed both phases with each input argument matching the required output. In the next section, we have shown the output we have generated after each phase.

9. Results

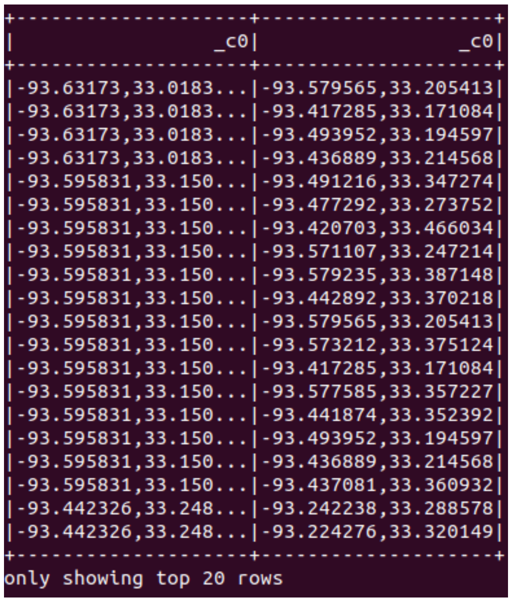
Following are the results of the queries we performed during phase 1 and phase 2.

a. Range Query



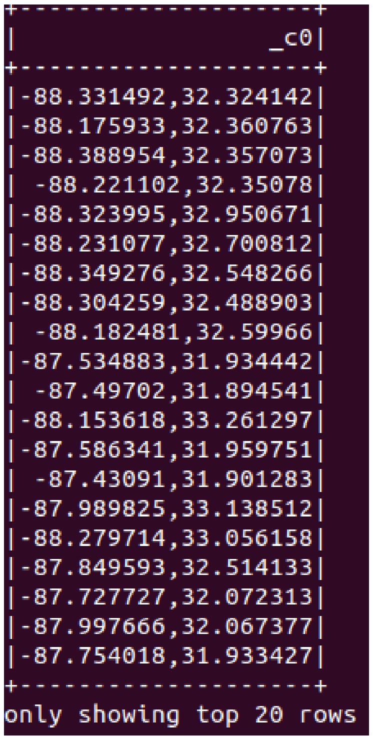
```
+--------------------+
|                 _c0|
+--------------------+
|-93.579565,33.205413|
|-93.417285,33.171084|
|-93.493952,33.194597|
|-93.436889,33.214568|
+--------------------+
```

b. RangeJoin Query



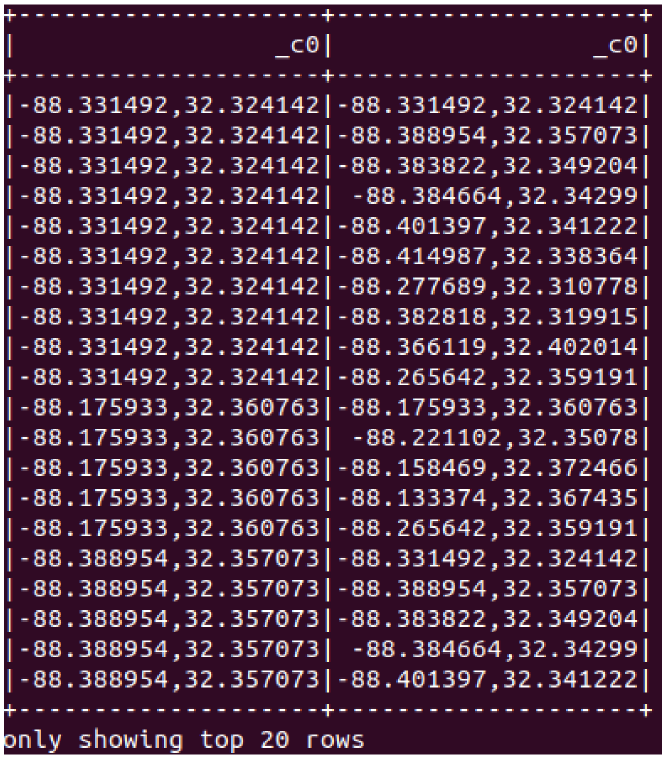
```
+--------------------+--------------------+
|                 _c0|                 _c0|
+--------------------+--------------------+
|-93.63173,33.0183...|-93.579565,33.205413|
|-93.63173,33.0183...|-93.417285,33.171084|
|-93.63173,33.0183...|-93.493952,33.194597|
|-93.63173,33.0183...|-93.436889,33.214568|
|-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|
+--------------------+--------------------+
only showing top 20 rows
```

c. Distance Query



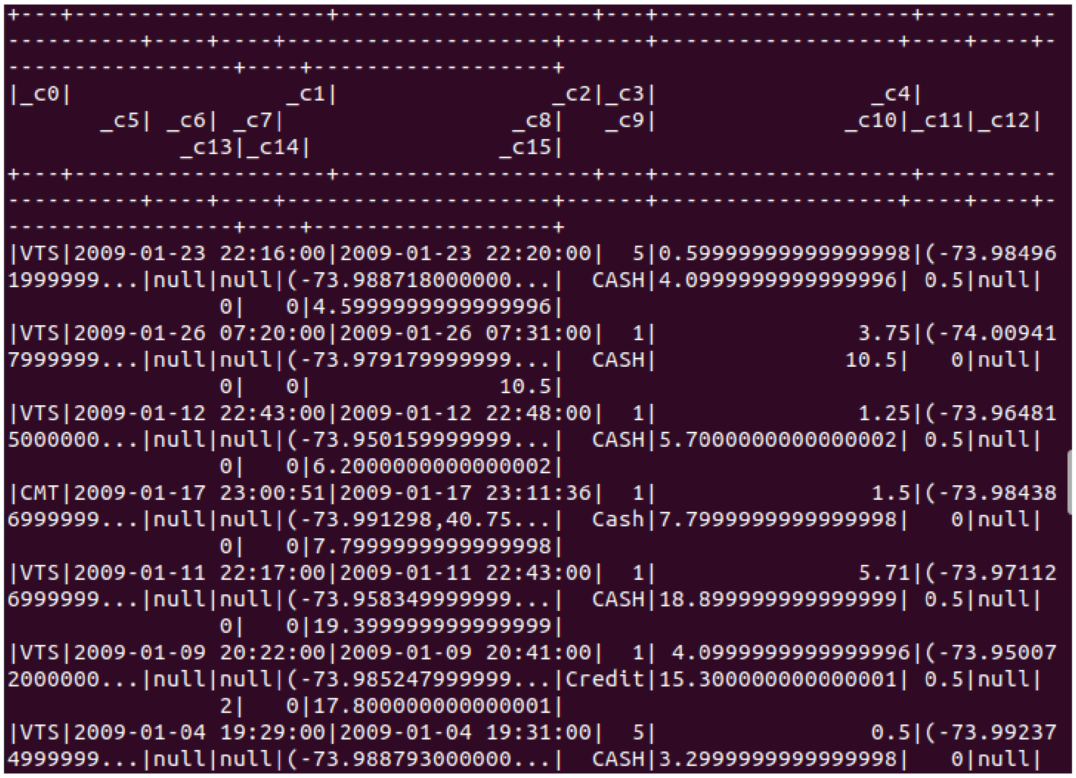
```
+--------------------+
|                 _c0|
+--------------------+
|-88.331492,32.324142|
|-88.175933,32.360763|
|-88.388954,32.357073|
| -88.221102,32.35078|
|-88.323995,32.950671|
|-88.231077,32.700812|
|-88.349276,32.548266|
|-88.304259,32.488903|
| -88.182481,32.59966|
|-87.534883,31.934442|
| -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
```

d. DistanceJoin Query



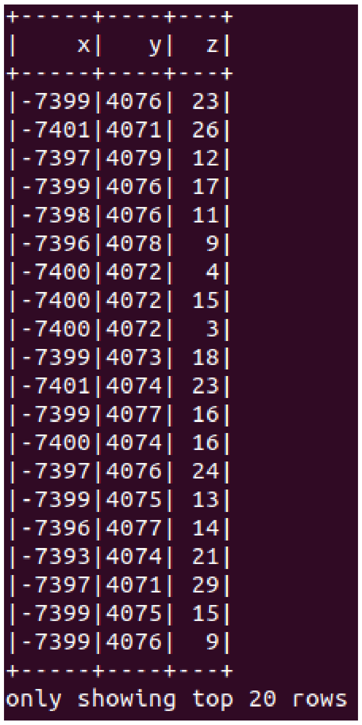
```
+--------------------+--------------------+
|                 _c0|                 _c0|
+--------------------+--------------------+
|-88.331492,32.324142|-88.331492,32.324142|
|-88.331492,32.324142|-88.388954,32.357073|
|-88.331492,32.324142|-88.383822,32.349204|
|-88.331492,32.324142| -88.384664,32.34299|
|-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
```

e. Hot Zone Analysis



```
+---+-------------------+-------------------+---+-------------------+----------
----------+----+----+--------------------+------+-------------------+----+----+-
-----------------+----+------------------+
|_c0|                _c1|                _c2|_c3|                _c4|
      _c5| _c6| _c7|                 _c8|   _c9|                _c10|_c11|_c12|
            _c13|_c14|               _c15|
+---+-------------------+-------------------+---+-------------------+----------
----------+----+----+--------------------+------+-------------------+----+----+-
-----------------+----+------------------+
|VTS|2009-01-23 22:16:00|2009-01-23 22:20:00|  5|0.5999999999999998|(-73.98496
1999999...|null|null|(-73.988718000000...|  CASH|4.0999999999999996| 0.5|null|
                0|   0|4.5999999999999996|
|VTS|2009-01-26 07:20:00|2009-01-26 07:31:00|  1|               3.75|(-74.00941
7999999...|null|null|(-73.979179999999...|  CASH|               10.5|   0|null|
                0|   0|              10.5|
|VTS|2009-01-12 22:43:00|2009-01-12 22:48:00|  1|               1.25|(-73.96481
5000000...|null|null|(-73.950159999999...|  CASH|5.7000000000000002| 0.5|null|
                0|   0|6.2000000000000002|
|CMT|2009-01-17 23:00:51|2009-01-17 23:11:36|  1|                1.5|(-73.98438
6999999...|null|null|(-73.991298,40.75...|  Cash|7.7999999999999998|   0|null|
                0|   0|7.7999999999999998|
|VTS|2009-01-11 22:17:00|2009-01-11 22:43:00|  1|               5.71|(-73.97112
6999999...|null|null|(-73.958349999999...|  CASH|18.899999999999999| 0.5|null|
                0|   0|19.399999999999999|
|VTS|2009-01-09 20:22:00|2009-01-09 20:41:00|  1| 4.0999999999999996|(-73.95007
2000000...|null|null|(-73.985247999999...|Credit|15.300000000000001| 0.5|null|
                2|   0|17.800000000000001|
|VTS|2009-01-04 19:29:00|2009-01-04 19:31:00|  5|                0.5|(-73.99237
4999999...|null|null|(-73.988793000000...|  CASH|3.2999999999999998|   0|null|
```

f. Hot Cell Analysis



```
+-----+----+---+
|    x|   y|  z|
+-----+----+---+
|-7399|4076| 23|
|-7401|4071| 26|
|-7397|4079| 12|
|-7399|4076| 17|
|-7398|4076| 11|
|-7396|4078|  9|
|-7400|4072|  4|
|-7400|4072| 15|
|-7400|4072|  3|
|-7399|4073| 18|
|-7401|4074| 23|
|-7399|4077| 16|
|-7400|4074| 16|
|-7397|4076| 24|
|-7399|4075| 13|
|-7396|4077| 14|
|-7393|4074| 21|
|-7397|4071| 29|
|-7399|4075| 15|
|-7399|4076|  9|
+-----+----+---+
only showing top 20 rows
```

10. Pitfalls Encountered

Both the Phase 1 and Phase 2 phases were developed without any significant stumbling blocks. However, owing to a lack of previous experience of the process, there were several slight setbacks.

a. To begin, it took a few tries for all of us to get Scala, Apache Spark, and all the other project dependencies up and running. It took some time for us to get comfortable with all the technologies we used.

b. Second, we had a problem with .map, which is a Scala 2.12 construct, but we were using in 2.11, and it took some time to figure out.

c. Finally, we defined the wrong path for the.JAR package while running spark-submit. We were able to figure out and successfully complete everything with a little research and coordination.

11. Conclusion

The Apache Spark architecture was used to address the issue of large-scale geospatial data processing. As a result, Spark was an excellent option for achieving low latency and high throughput. Characteristics such as in-memory computations, caching, and fault tolerance aided in lowering the total time taken by our server operations, which would have otherwise been huge considering the data size.

Spark, in conjunction with the Hadoop distributed file system, is a versatile storage platform for storing data in massive formats. Understanding Apache Spark in parallel with Hadoop helped us realize the importance of a Distributed and Parallel Database System as well as its many benefits. The use of Spark SQL APIs in Scala assisted us in effectively analyzing geospatial data.