Hot Spot Analysis Report

Shachi Shah
Arizona State University
CSE 511 - Summer 2025
Azusa, California
spshah22@asu.edu

I. Introduction

The Hot Spot Analysis project in CSE511 focused on uncovering geographic and temporal patterns in large-scale taxi ride datasets using distributed processing frameworks. By leveraging Apache Spark and Scala, I worked with real-world spatial and temporal data to uncover patterns that indicate “hot spots” of activity across New York City. The project was designed to simulate how large-scale data analytics systems can process location-based data to support applications such as urban planning, public safety, and traffic analysis.

The project involved two key components: Hot Zone Analysis, which focuses on identifying whether a set of spatial points fall within given rectangular zones, and Hot Cell Analysis, which employs statistical modeling (specifically the Getis-Ord G* score) to detect areas of significantly high activity, or "hot spots," in a discretized 3D space consisting of x, y, and z coordinates.

Working with such a rich dataset in a distributed computing setting gave me hands-on experience in transforming, filtering, and statistically analyzing large-scale geospatial datasets. The challenge lay not only in the correctness of the algorithms but also in their efficient and scalable implementation using Spark SQL and user-defined functions (UDFs). This report outlines the approach, lessons, and implementation logic I followed to complete the Hot Spot Analysis project.

II. Reflection

This project involved implementing two core spatial queries using Apache Spark in Scala: Hot Zone Analysis and Hot Cell Analysis. I began the project by carefully reviewing the starter files and understanding the data formats and query requirements. I approached the problem by dividing the tasks into distinct milestones: getting Hot Zone Analysis to pass all test cases first and then debugging and implementing Hot Cell Analysis correctly.

For the Hot Zone Analysis, I implemented the function ST_Contains that checks whether a spatial point lies within a given rectangle. Once that was functional, I focused on ensuring that spatial joins and filtering worked correctly to identify all the points within rectangles.

The Hot Cell Analysis required significantly more work. Initially, I built the pipeline to calculate spatial cell coordinates (x, y, z) from latitude, longitude, and timestamp data. I filtered valid points, counted occurrences per cell, computed statistical aggregates (mean, standard deviation), and finally calculated the Getis-Ord G* score to identify spatial hotspots. Multiple rounds of debugging were needed, especially around z-coordinate computation and proper score calculation.

Eventually, by validating type casting, removing incorrect column selections, and adhering to correct ordering, I passed all test cases.

III. Lessons Learned

This project reinforced several key concepts and skills:

- Spatial data processing: I gained hands-on experience in interpreting and handling geospatial coordinates and timestamps in a big data setting.
- UDP design in Spark: Implementing coordinate mapping and point-in-rectangle functions as user-defined functions deepened my understanding of Spark SQL integration.
- Distributed computation with DataFrames: Optimizing transformations, joins, and aggregates within Spark's lazy evaluation model was a core part of the debugging process.
- Debugging Spark Applications: Diagnosing null pointer exceptions and incorrect outputs in a distributed setting required careful design, input validation, and clear intermediate outputs.

IV. Implementation

A. *What Hot Zone Analysis is Doing*

Hot Zone Analysis identified which pints from a dataset fall inside predefined rectangular zones. Each rectangle represents a zone of interest, and the objective is to find all pickup points within those rectangles.

The key function used:

```
def ST_Contains(queryRectangle: String,
pointString: String): Boolean = {
  val rect = queryRectangle.split(",")
  val point = pointString.split(",")

  val x1 = rect(0).toDouble
  val y1 = rect(1).toDouble
  val x2 = rect(2).toDouble
  val y2 = rect(3).toDouble

  val px = point(0).toDouble
  val py = point(1).toDouble

  val minX = Math.min(x1, x2)
  val maxX = Math.max(x1, x2)
  val minY = Math.min(y1, y2)
  val maxY = Math.max(y1, y2)

  px >= minX && px <= maxX && py >= minY && py
<= maxY
}
```

This function determines if a given point is spatially enclosed by the rectangle’s boundaries.

In the Spark SQL pipeline, rectangles and points are cross-joined, and the function ST_Contains is applied to filter matches, producing a list of all hits.

B. *What Hot Cell Analysis is Doing*

Hot Cell Analysis identifies spatial and temporal hotspots using a statistical model based on the Getis-Ord G* statistics. The steps involved are;

Step 1: Coordinate Extraction - each pick up point is transformed into a 3D grid cell using:

- `x`: from longitude
- `y`: From latitude.
- `z`: From timestamp (specifically the day of the month). These are calculated using a uniform step size (0.01) for spatial dimensions.

Step 2: Spatial Filtering - only those grid cells within the project-specified bounds are retained:

- `x: in [-74.50, -73.70]`
- `y: in [40.50, 40.90]`
- `z: in [1, 31]` (days in a month)

```
val minX = math.floor(-74.50 /
HotcellUtils.coordinateStep).toInt
val maxX = math.floor(-73.70 /
HotcellUtils.coordinateStep).toInt
val minY = math.floor(40.50 /
HotcellUtils.coordinateStep).toInt
val maxY = math.floor(40.90 /
HotcellUtils.coordinateStep).toInt
val minZ = 1
val maxZ = 31
```

Step 3: Aggregation - we count how many pickup points fall into each (x, y, z) cell. This gives us the count value per cell.

```
val cellCounts = spark.sql("""
  SELECT x, y, z, COUNT(*) AS count
  FROM filtered
  GROUP BY x, y, z
""")
```

Step 4: Mean and Standard Deviation - using the counts, we compute:

- $mean = sum/n$
- $stddev - \sqrt{sumsq/n - mean^2}$

Step 5: Neighbor Join - each cell is joined with all 26 spatial-temporal neighbors ($\pm$ 1 $in\ x, y, z$):

```
val neighbors = spark.sql("""
  SELECT a.x, a.y, a.z, b.count AS
neighborCount
  FROM cellCounts a JOIN cellCounts b
  ON ABS(a.x - b.x) <= 1 AND ABS(a.y - b.y) <=
1 AND ABS(a.z - b.z) <= 1
""")
```

Then aggregate:

```
val neighborSums = spark.sql("""
  SELECT x, y, z, SUM(neighborCount) AS
sumWijXj, COUNT(*) AS wij
  FROM neighbors
  GROUP BY x, y, z
""")
```

Step 6: Score Calculation - for each cell, the G* score is calculated using:

```
gscore = (sumWijXj - mean * wij) / (stddev *
sqrt((n * wij - wij * wij) / (n - 1)))
```

This captures how significantly a cell's neighbor counts deviate from the expected average.

Step 7: Output - we order all cells by G* score descending and return the top 50 hot cells.

V. Conclusion

This project provided a comprehensive, hands-on experience in spatial data analysis using Apache Spark. By implementing both Hot Zone and Hot Cell analysis, I was able to develop a deeper understanding of how large-scale spatial queries and statistical analyses can be orchestrated in a distributed computing environment. Through the Hot Zone analysis, I practiced parsing and joining geospatial data to identify all points within specified rectangles, a fundamental concept in spatial range queries. In contrast, the Hot Cell analysis introduced me to more complex spatial-temporal analytics, requiring the computation of Z-scores across a 3D grid of spatial and temporal dimensions.

The process of converting real-world pickup data into a structured format, filtering valid spatial-temporal cells, computing statistical measures like mean and standard deviation, and finally identifying spatial hotspots gave me insight into both algorithmic thinking and practical implementation challenges. I also gained valuable debugging experience, as I had to validate assumptions, fix type mismatches, and ensure consistency between raw data formats and computed results.

Ultimately, this project reinforced the importance of rigorous validation and thoughtful code modularity when working with big data. It also demonstrated the power of Spark SQL and UDFs in extending SQL-like query capabilities to large and complex datasets. This experience will be invaluable as I continue exploring data-intensive applications in both academic and professional settings.

References

[1] CSE 511 Course Staff, *CSE 511: Data Processing at Scale. Hot Spot Analysis Project*, Arizona State University, 2025. [Online]. Available: https://canvas.asu.edu/.

[2] CSE 511: Course Staff, *CSE 511: Data Processing at Scale. CSE511-Project-Hotspot-Analysis ZIP file*, Arizona State University, 2025. [Online]. Available: https://canvas.asu.edu/.