Data Processing at Scale

Spatiotemporal Analysis with Apache Spark

James William Dunn
jwdunn2@asu.edu

I. Introduction

The following project addresses big data. Given a large dataset of Yellow Cab taxi trips throughout the greater New York City area, the aim is to provide decision support for daily operations. The multiple project phases utilize a functional programming language and the Apache Spark computing framework to determine the number of pickups in a given region and then, the statistically significant geospatial cells with the highest number of pickup points. With the completed implementation, one could adjust parameters and explore the dataset further for both tactical and strategic purposes.

This was a team programming challenge with a predetermined starting code template. In the first phase, we installed Spark and learned the Scala programming language. In the next phase, we developed a pair of user-defined functions to be used within queries. In the final phase, we designed and implemented a pair of analysis routines to determine both hotzones (regions of higher fares) and hotcells[1] (applying spatial statistics). We were also given a target result set for each phase. The goal was to fully comprehend the process from end to end (see Fig. 1) and produce an exact matching output list.

II. Solution

As a team, we determined a plan of action by first understanding the theory behind the primary computational elements in the project. The initial task involved determining whether a given rectangle *R* *contains* a given set of points *P*. This represents the real-world scenario of locating customers relative to geographic areas. We explored two options and decided on a more optimal fail-fast approach of checking for coordinates beyond the bounds of *R*, due to the higher probability of occurrence. Next, we examined the second task of determining whether two given points *p1* and *p2* are *within* a given distance *d* of one another. This particular solution is a simple Cartesian distance calculation as follows:

$$pointDistance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (1)$$

where *p1* is defined by the coordinates (x_1,y_1) and *p2* is defined by (x_2,y_2). Next, we turned our attention to the hotzone and hotcell analysis.

[1]http://sigspatial2016.sigspatial.org/giscup2016/problem

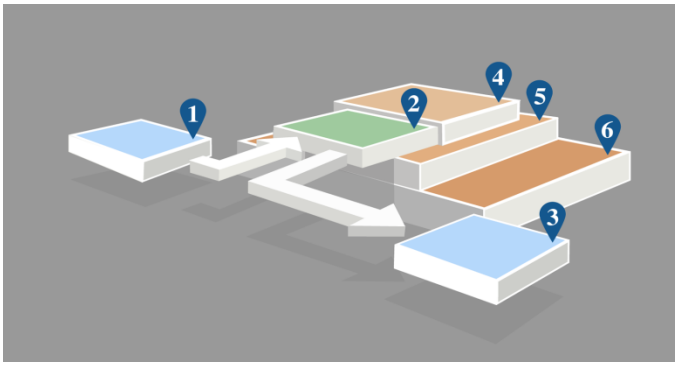

Figure 1: Architectural overview of the system. 1-input, 2-project solution code, 3-output, 4-Spark & SparkSQL, 5-JavaRE, and 6-OS

For the hotzone analysis solution, we used our implemented *contains* function and developed the missing steps in the given procedure, adding the requisite DataFrame grouping, ordering, counting, and coalescing. Then, for the hotcell analysis task, we needed to first understand the data model as a multi-dimensional structure of a geographic region containing sparse taxicab pickup points with associated time stamps. Further, we needed to devise a processing scheme in order to compute the target results.

To locate clustering patterns of spatiotemporal related data, we utilized the Getis-Ord G_i^* statistic [1]:

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

where x_j is the pickup point within cell j, and mean $\bar{X}$ and standard deviation S are calculated as:

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n} \qquad S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\bar{X})^2} \qquad (3)$$

It is important to note that n is the total number of cells in the population, not just those with associated data points. Built-in library functions such as those in Scala or Java may not take this into account and therefore can produce significantly incorrect results. Weighting of one cell to another is indicated by $w_{i,j}$ with the simplification given by [2] that w=1 for cell j and its 26 immediately neighboring cells, and w=0 for all

others. We used a 2.4 gigabyte dataset consisting of a single month of taxicab pickup coordinates spanning 31 days in January 2009. This dataset is read by the given code template, and an *in-memory* DataFrame is produced. Our solution then follows these next steps to process the data:

Algorithm: Determine hot cell ranking
1: Filter only cells within a given envelope.
2: Group and count the cells.
3: Determine mean and standard deviation.
4: Self-join and count neighboring cells within 1 unit distance.
5: Group by coordinates and sum.
6: Calculate Getis-Ord score for each cell.
7: Sort in descending order of score.
8: Return resulting tuples.

Observe that we perform a self-join at step 4. The predicate we used is similar to the *within* function developed earlier, but optimized for finding cells within one unit of distance by avoiding the square root. Step 6 computes the Getis-Ord G_i* statistic. The completed Scala code for this is shown in Fig. 3.

III. Results

Query results of phase one were acceptable. Likewise, all initial testing for the hotzone analysis section passed.

For the hotcell analysis section, the resulting top 50 cells with the highest statistical significance overlapped in only 5 geographic regions (see Fig. 2). Below is a listing of those regions (see Table I). Of particular interest is the recurrence of the 15th day of the month in each of the regions. What does this imply…payday?

Table I
Statistically Significant Cells

Lng	Lat	Day	Getis-Ord G_i*
-74.00	40.73	30	67.24455856
-73.99	40.74	15	72.79502084
-73.99	40.75	15	78.47816293
-73.98	40.75	15	75.01413643
-73.98	40.76	15	71.06202269

In contrast to the smaller cell size in the GISCup in 2016, this project improves processing time by using a larger cell size of 0.01 degrees. This translates to cell dimensions of approximately 843m east/west by 1110m north/south (see Fig. 2). While this level of resolution is sufficient for a test case, reducing the size of the cells would yield more useful results for decision making, at the cost of increased processing time.

I see a direct correlation of this project to the research I am involved with at kynamatrix Research Network. We are investigating an algorithmic approach to building a temporary urban environment in the aftermath of disaster. In this work there is a big data scenario, requiring geo-spatial analysis to determine survivor location hotspots and aggregate incoming sensor information from a streaming city-management dataset.

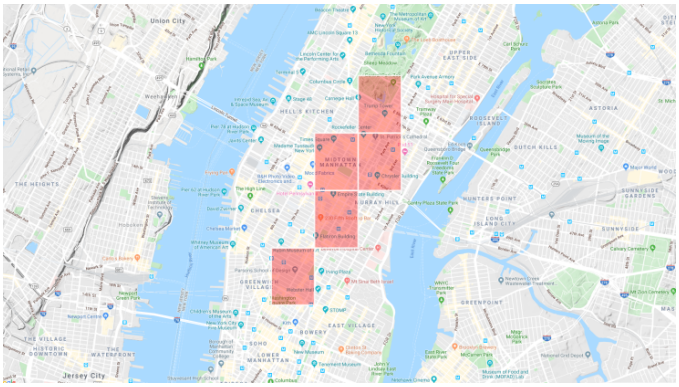

Figure 2: The top 'hot' cells found in downtown Manhattan.

IV. Contributions

In this project, I acted in several roles including project manager, developer, tester, writer, and code librarian. To 'get acquainted', our team took on a subtask to discuss whether relational databases will remain the default choice in the emerging era of big data, cloud computing, and NoSQL databases. After the discussions concluded in Slack, I collected the body of thoughts, cross-correlated the common ideas, and integrated a cohesive report of our observations.

Next, I prompted for phase one status and shepherded questions surrounding the installations of Spark, Java Development Kit, SparkSQL, and Scala. A couple of members on our team encountered difficulty with configuring their respective local system, however, I was able to provide guidance to make progress in achieving total participation. With installations out of the way, we then discussed a task breakdown strategy. I noticed that it would be in the team's best interest to utilize a Kanban-style project board to track tasks. This came up at a time when we were beginning to share some code within the private Slack channel. With the need to control code versioning, I set up a private repository on GitHub, which includes a project-tracking tool.

I asked the team to visit the repository and update project task items. For those teammates that had no prior experience with GitHub or the use of agile software techniques, this would afford them more experience for the future. Further, I compiled some common usage instructions in a document named GitHubGuidance.md. In one case, I provided exact steps to follow. To create a progress report at milestone 3, I gathered status paragraphs from each team member by having them update a common document in the repository.

For the design and coding phase of this project, I collaborated with team members on two of the primary components. In the first phase, I assisted with the *contains* function, while in the second phase, I completed the missing design steps of the hotzone and hotcell algorithm. Further, I implemented the final steps of hotcell, optimized code, and performed tests. In the hotzone implementation, I suggested the use of fast-failing logic to test for points outside of the rectangle. In the hotcell implementation, I contributed the calculation of the mean and

standard deviation, plus the Getis-Ord G_i* scoring code with integrated edge condition checking (see Fig. 3).

```
def gScore(sum:Int, x:Int,y:Int,z:Int,
  mean:Double,stddev:Double,numCells:Double,
  minX:Int,maxX:Int,
  minY:Int,maxY:Int,
  minZ:Int,maxZ:Int): Double = {
  var edge = 0
  if (x == minX || x == maxX) edge += 1
  if (y == minY || y == maxY) edge += 1
  if (z == minZ || z == maxZ) edge += 1
  var neighbor = 0
  edge match {
    case 1 => neighbor = 18
    case 2 => neighbor = 12
    case 3 => neighbor = 8
    case _ => neighbor = 27
  }
  (sum - mean * neighbor) / (stddev *
    math.sqrt((numCells * neighbor -
    neighbor * neighbor) / (numCells - 1)))
}
```

Figure 3: Scala code I contributed for computing the Getis-Ord score (see equation 2) with edge cases.

To explore the edge conditions where a cell has fewer neighbors than the interior case, I created an interactive sketch using the p5js library (see Fig. 4). This allowed me to freely move around the space and position the reference cell into one of three special conditions: touching one plane; the intersection of two planes; or the corner of three planes. Through Slack, I shared my visualization with the team for feedback and with the general student body of our class.

For the source code submission, I analyzed all dependencies and jettisoned anything extraneous. I was able to create a minimal tree, which compiled for Spark and compressed to a 4K zip file. I shared this with the team for review and testing.

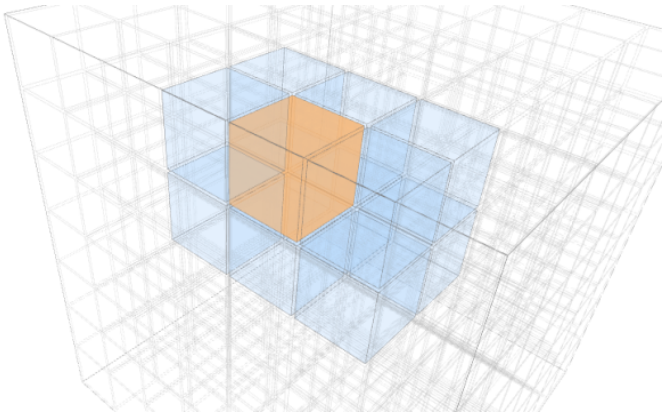

Figure 4: Interactive sketch I created to visualize edge conditions. See openprocessing.org/sketch/799890

Finally, for the systems documentation, I authored complete sections and edited the language throughout for readability. The section titles I authored include "Validation and Testing" and "Process, Roles, and Responsibilities." Further, I added the code examples. To improve the appeal of the documentation, I collaborated with a professional graphic designer to define an overall layout approach and to construct a logo representing the four-member team.

V. Lessons Learned

Perhaps the greatest lesson extracted from this project is an enhanced ability to understand and apply statistical analysis to large-scale spatiotemporal datasets such as the multi-gigabyte NYC taxicab trip data. In addition, a working knowledge of Apache Spark is invaluable for managing big data in-memory. Another key takeaway was the learning of a new programming language. While I enjoy understanding the many dialects of Java and C, the Scala language in particular has specific application to my current work in building responsive and high-performance web applications. The website *Scala.js* provided an opportunity to see an alternative to the strongly typed option of TypeScript over JavaScript.

Beyond the provided course and project materials, I sought to find alternative sources to enhance my comprehension of the complex substance of data processing at scale. I discovered and investigated related textbooks, video courses, articles, projects, and pertinent examples of software documentation. For instance, I visited the ASU library online to gain a better understanding of transaction processing. I found books at my local library to heighten my grasp of statistics. Finally, I learned to listen more closely for meaning hidden between the words in written, video, and audio discussions.

While I have experience in large-scale relational database design and development, the project heightened my awareness of spatial analysis, computational geometry, geographic information systems, and geospatial intelligence. The intense focus on Spark and spatiotemporal analytics has elevated my interest to learn more about big data processing and mining.

I enjoyed the opportunity to collaborate with colleagues on this project; my CSE511 teammates were Jonathan Kunze, Niloufar Roozegar, and Ryan Berg. As the project moved forward, each member provided insights and added to the collective value through their range of experience and unique perspectives. Lastly, this project increased my awareness that a distributed team can launch immediately, communicate effectively, meet the assigned goals, and deliver on time.

VI. References

[1] J.K. Ord and A. Getis. 1995. 'Local Spatial Autocorrelation Statistics: Distributional Issues and an Application' in Geographical Analysis 27(4).

[2] S. Peng *et al*., “Simplification and Refinement for Speedy Spatio-temporal Hot Spot Detection Using Spark (GIS Cup 2016).” Available: cs.umd.edu/~hyw/papers/gis16sr.pdf