

GROUP 21 - Data Processing at Scale

CSE 511

## **PROJECT REPORT**

Members:

Arnav Chakravarthy

Nipun Mediratta

Divya Kshatriya

Chirra Reddy Sai Sankeerth Reddy

# Table of Contents

<b>Introduction</b>	<b>3</b>
<b>System Overview</b>	<b>4</b>
<b>Frameworks Used</b>	<b>6</b>
Apache Spark	6
<b>SYSTEM SETUP</b>	<b>7</b>
Setting Up Password-Less SSH	7
Configuring Spark Cluster	8
Starting the Spark Cluster	8
<b>GEOSPATIAL DATA ANALYSIS</b>	<b>8</b>
SparkSQL	8
HotSpot Analysis	11
HotCell Analysis	13
<b>CONCLUSION</b>	<b>16</b>

# **Introduction**

Geo-spatial data is different from the data we normally observe, since it is tied to physical space. They basically are data which have a location on the surface of the earth. This kind of data could be static ( location of a restaurant ) or could be dynamic ( a moving vehicle ). Geospatial data combines spatial information and in some cases temporal information too.

A major peer-peer taxi cab firm requires the development and running of multiple spatial queries on their database which contains both geographic data as well as real-time location data of it's customers. The goal is to help the firm extract data from the database which will help them in taking operational and strategic level decisions by developing and running these queries.

The traditional databases lack the ability to do large scale data analytics since their performance degrades as the amount of data increases. Geospatial data contains a large number of points and hence is a very dense kind of data. Performing analysis on this kind of data requires large computation and throughput while achieving low latency. Hence efficient data processing frameworks are needed to address these issues.

This is where Apache Spark comes into the picture. Apache Spark is a distributed in-memory cluster computing framework which can make use of resources of multiple nodes ( Virtual Machines ), in order to fulfill a complex and computationally intensive query. This is the data processing framework we will be using for our analysis.

The aim of this project is to perform large scale geo spatial analysis of the New York Taxi dataset which spans the years from 2009 to 2012 ( we will be performing our analysis on a subset of this), and identify "hot zones" for pickups and drops.

We have set up an Apache Spark cluster using 3 of Amazon's Elastic Cloud Compute Instances, and utilizing the master-slave architecture for effective data processing and resource sharing. The Geo Spatial Analysis is carried out using the Spark SQL module provided by Apache Spark.

The first phase of the project requires us to perform queries such as range query, range join, distance query and distance join.

The second phase of the project requires us to perform hot zone analysis as well as hot cell analysis. Hot zone refers to finding the number of data points (longitude,latitude) present in a rectangle defined by its diagonal points, hence inferring the "hotness" of the rectangle. Whereas Hot cell analysis refers to applying spatial statistics to spatio-temporal big data in order to identify statistically significant spatial hot spots. The

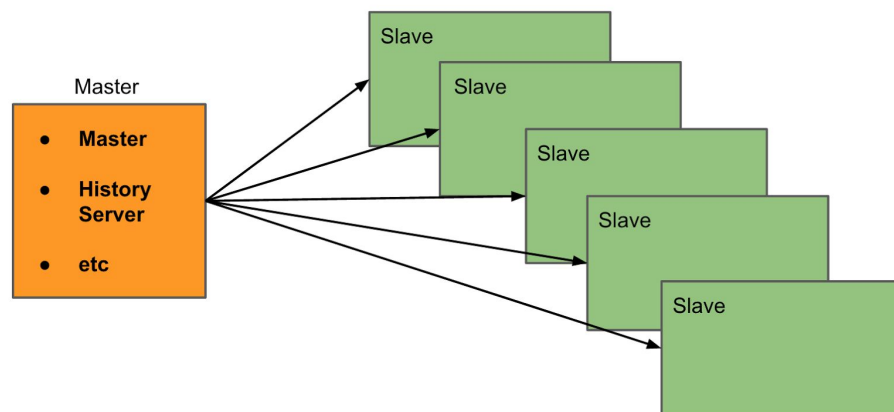
statistic measure used in this analysis is the Gertis-Ord  $G_i^*$  which tells us where features with either high or low values cluster spatially and in our case temporally too. This task is taken from the ACM Sigspatial Cup 2016 website.

## System Overview

We have utilized Amazon's Elastic Cloud Compute Instances as our cloud infrastructure since it provides capabilities like VM monitoring, auto scaling (if needed), ability to specify Amazon Machine Images to launch new instances with the same configuration, hence making the scaling process much easier.

Apache Spark uses the master slave architecture for its cluster setup. There are many types of cluster setups but we will be using the Spark Standalone setup. This uses the Spark standalone cluster manager which is included as part of the Spark Distribution.

Here we have 3 instances with 1 master and 2 worker nodes. The Spark driver runs on the master node and requests for resources from the Standalone cluster manager which also resides on the same node, and then starts the execution of the application on the worker nodes. The driver is responsible for the coordination of all worker nodes and overall execution of tasks. It also hosts a Web Client on the port 8080 where one can view all clusters and job statistics.



**Fig 1:** Apache Spark Standalone Master Slave Setup

All the 3 instances reside within the same Virtual Private Cloud (VPC) and hence can communicate with each other using private IP addresses. However for us to log in to an instance we can make use of SSH by using the IPv4 IP address. These instances are password

protected by a .pem file. Hence we must provide this .pem file when trying to access any instance.

Launch Instance

Connect

Actions

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
<input type="checkbox"/>	Master	i-041dcf882f8ab992d	t2.micro	us-east-2a	<div>running</div>	<div>2/2 checks ...</div>	None	<div>ec2-18-218-79-235.us-...</div>	18.218.79.235
<input type="checkbox"/>	Slave1	i-0512e17103df8f29a	t2.micro	us-east-2a	<div>running</div>	<div>2/2 checks ...</div>	None	<div>ec2-3-133-114-250.us-...</div>	3.133.114.250
<input type="checkbox"/>	Slave2	i-0c5456beb3d5e13f7	t2.micro	us-east-2a	<div>running</div>	<div>2/2 checks ...</div>	None	<div>ec2-3-15-148-130.us-e...</div>	3.15.148.130

Fig 2: EC2 dashboard

The security groups of these instances were configured using port binding to allow all external traffic to access it. This was done to allow us to access the web interface from any client machine outside the VPC.

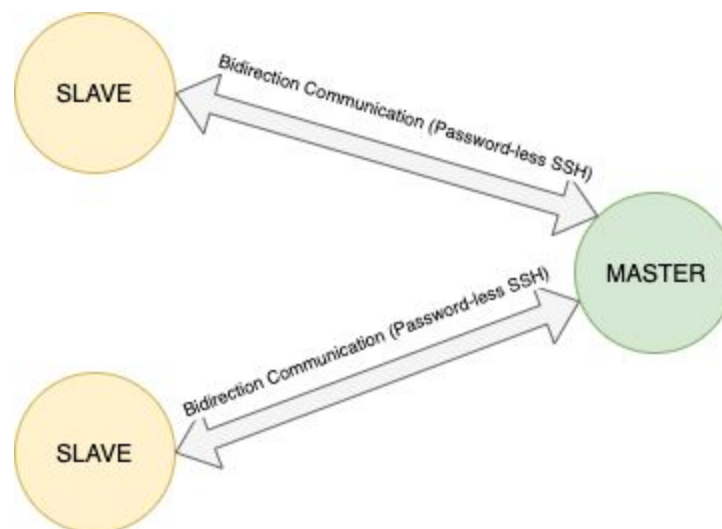


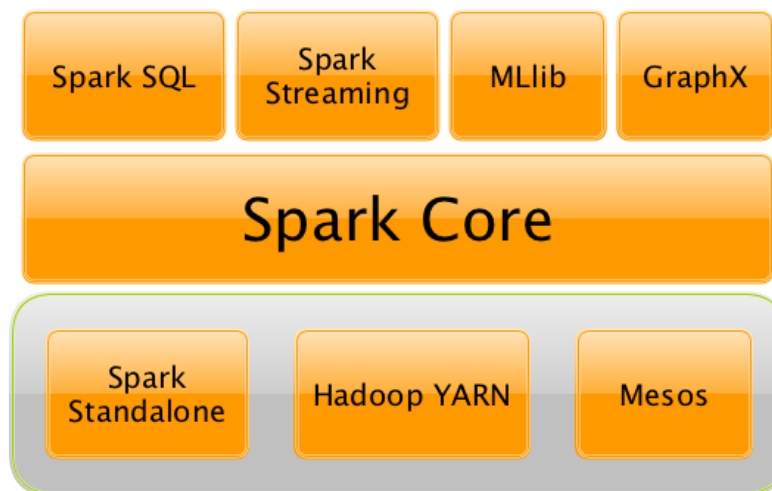
Fig 3: EC2 Spark cluster setup

As seen in the above diagram, we had to configure the machines for password-less SSH in order to allow for bidirectional communication when running a Spark Job.

## Frameworks Used

### Apache Spark

Apache Spark is an open source distributed general purpose cluster computing framework with an in memory data processing engine that can perform large scale analytics on data in the form of batches as well as streaming.



**Fig 4:** The Spark Platform

It is generally faster than Apache Hadoop since in contrast to Hadoop's two stage disk based Map Reduce computation engine, Apache Spark consists of a multi stage in-memory computing engine allowing most of the computations to run in memory. The abstraction it provides for in memory cluster computing is called Resilient Distributed Databases (RDD). It also makes use of a Directed Acyclic Graph (DAG) of computation stages. Another feature is that it postpones any processing until really required for actions (also known as lazy evaluation). Fault tolerance is achieved by maintaining an RDD lineage graph, which it can use to reconstruct any RDD lost due to any system failure.

# **SYSTEM SETUP**

## **Setting Up Password-Less SSH**

To enable machine 1 to access machine 2 via password-less ssh, we need to do the following steps:

1. Do `ssh-keygen -t rsa` on machine 1. This would generate SSH keys in your `$HOME/.ssh/id_rsa.pub` file.
2. We then copy this key and append to the `$HOME/.ssh/authorized_keys` of machine 2.
3. We can now ssh into machine 2 without providing a password.

Hence this has to be done between each slave and master machine and vice versa.

## **Configuring Spark Cluster**

1. We need to configure Spark on all three machines, and ensure they are all installed on the same path as well as on the same user.
2. We also install Scala as well as Java.
3. Sbt is also needed on the master machine in order to build the Scala project
4. On the master machine we must configure the `spark/conf/slaves` and add the following:
  - Slave1 IP address
  - Slave2 IP address

We could either add the IP address directly or give an alias to it in `/etc/hosts` and add the alias names

## **Starting the Spark Cluster**

1. Execute `$PATH/spark/sbin/start-master.sh` to start the standalone master server
2. Execute `$PATH/spark/sbin/start-slaves.sh` to start the slave servers
3. We can now access the master Web Client on `Master-IP-Address:8080`

4. We can replace steps 1 and 2 by executing `$PATH/spark/sbin/start-all.sh`
5. To stop the cluster we can execute `$PATH/spark/sbin/stop-all.sh`

After performing these steps, if we run a `spark-submit` command whilst specifying the master IP address, the code containing the transformations and actions are converted into a logical Directed Acyclic Graph, which is then converted into a physical execution plan with a set of stages. It creates small physical execution units referred to as tasks which are then sent to the Spark Cluster. Executors are then launched on the worker nodes to execute the tasks assigned to them.

## GEOSPATIAL DATA ANALYSIS

### SparkSQL

As discussed earlier, in the first phase of the project, we use Apache Spark to write two user defined functions namely - `ST_Contains()` and `ST_Within()`. Essentially, 4 queries are executed using these 2 functions which are helper functions for the 4 queries that are actually run on the dataset. Hence the two helper functions were implemented

`ST_Contains()`

This is a boolean function which is used to check and return true if a given point is within the given rectangle or not. Otherwise, it returns false.

Input : String point, String rectangle

Output: Boolean

Implementation

- A point is represented by the coordinates x and y. For instance, consider a point P1 with coordinates (1,2) where x = 1 and y = 2
- Likewise, a rectangle can be represented by two points which represent the coordinates of its diagonal. For instance, consider a rectangle with coordinates (3,4) and (6,7) which are the coordinates of the diagonal of the rectangle
- The inputs are in string format. For instance, point P1 is passed as an input which is of the form "x,y". In addition, rectangle can be passed as an input to the function in the following form "x1,y1,x2,y2".
- Since the inputs are in the string format, we use `split()` to split by "," in order to get the individual coordinate points for calculation purposes
- We check if the x coordinate of the point lies in between the x coordinates of rectangle. Likewise, we check if the y coordinate of the point lies in between the y coordinates of the rectangle. If they do, then we return true otherwise we return false.



## ST\_Within()

This is another boolean function which is used to check and return true if the two given points lie within a given distance. Otherwise, it returns false.

Input: String point1, String point2, Distance d

Output: Boolean

### Implementation

- As discussed earlier, points are represented by two coordinate points which is x and y. Hence, we have two points, let's say P1 with coordinates x1,y1 and P2 with coordinates x2,y2.
- As mentioned earlier, we use split function to split the strings and get the individual coordinate points for calculation purposes.
- We employ the usage of euclidean distance between the two given points and check if the resulting euclidean distance is less than or equal to the given distance. If it is within the given distance, we return true. Otherwise, we return false.

The queries that use the above helper functions are of the following format

### Range Query:

```
select *  
from point  
where ST_Contains(point._c0, '-155.940114,19.081331,-155.618917,19.5307')
```

From the query above, it can be inferred that Range Query uses ST\_Contains() as a helper function in order to execute the query. In addition, we are querying point table in order to retrieve individual points for checking if each of them lie within the given rectangle or not. Basically, Range Query is used to check if each of the points is within the rectangle boundary or not. The query displays those points that satisfy the condition imposed by ST\_Contains function

### Range Join Query:

```
select *  
from rectangle,point  
where ST_Contains(rectangle._c0,point._c0)
```

From the query above, it can be understood that the query uses two tables - rectangle and point in order to retrieve coordinates from each of the respective tables and pass them to ST\_Contains which is a helper function to check whether the points from the point table lie within the points from rectangle table. Essentially, the query does a join

on rectangle and table based on the condition that the points from the point table lie within the points from the rectangle table. The query displays those points that satisfy the condition imposed by the ST\_Contains function.

#### Distance query:

```
select *  
from point  
where ST_Within(point._c0, '-88.331492,32.324142',10)
```

Distance query is used to check if two given points lie within a distance or not. From the query above, it can be inferred that point table is being queried in such a way that each of the respective points from the point table is compared with the given point by passing those points along with the given distance to ST\_Within function where it is checked if the given input points are within the given input distance or not. The query displays those points that satisfy the condition imposed by ST\_Within function

#### Distance join query:

```
select *  
from point p1, point p2  
where ST_Within(p1._c0, p2._c0, 10)
```

As the name suggests, Distance Join Query is used to join point table with point table on the condition that the two given points from point table are within the given distance. Essentially, the query uses ST\_Within where the points from point table is passed to ST\_Within function which checks if the given points satisfy the constraint of ST\_Within function and if they do, the query displays those points

## HotSpot Analysis

This phase employs the usage of New York City Yellow Cab Taxi TripsDataset. This dataset contains records of all trips from 2009 to 2012. The figure below represents the dataset where the blue dots represent the pickup locations.



**Fig 5:** New York Yellow Cab City Taxi

In this task, we work on spatial hot spot analysis. The two tasks that we focus on are HotSpot Analysis and Hot Cell Analysis.

**Hot Zone Analysis:** This task requires us to find hotter rectangles. Hotter rectangles is defined by having numerous points within the boundary of the rectangle. Basically, the more the points present within the rectangle, the hotter the rectangle. It can be observed that this task is similar to range join query as discussed earlier. This task uses two tables - rectangle and point. The query in order to calculate the HotZone requires to do a join on rectangle and point on the condition that the points are contained within the rectangle. In order to achieve this, we use `ST_Contains()` to return those rectangles that satisfy the condition imposed by the helper function.

Name: `runHotZoneAnalysis`

Input: `sparkSession spark`, `String pointPath`, `String rectanglePath`

Output: Sorted dataframe containing the rectangles and the number of points in it

Steps:

1. A join query between rectangle and point table is done on the condition satisfied by ST\_Contains() which returns the set of rectangles and points which satisfy the condition.
2. The result of step 1 is stored in a temporary view for further processing
3. A query is run on the temporary view which selects all rectangles and count of points present within the rectangle followed by grouping of the rectangle so that it displays a rectangle and count of points it has. In addition, it is sorted in decreasing order so that hotter rectangles are displayed first.

rectangle	count(point)
[-73.789411,40.666...	1
[-73.793638,40.710...	1
[-73.795658,40.743...	1
[-73.796512,40.722...	1
[-73.797297,40.738...	1
[-73.802033,40.652...	8
[-73.805770,40.666...	3
[-73.815233,40.715...	2
[-73.816380,40.690...	1
[-73.819131,40.582...	1
[-73.825921,40.702...	2
[-73.826577,40.757...	1
[-73.832707,40.620...	200
[-73.839460,40.746...	3
[-73.840130,40.662...	4
[-73.840817,40.775...	1
[-73.842332,40.804...	2
[-73.843148,40.701...	2
[-73.849479,40.681...	2
[-73.861099,40.714...	21

Fig 6: HotZone Output

## HotCell Analysis

In Hot Cell analysis, a list of cells is analyzed where the hotness of the cells is measured. Hotness of Cells is analyzed using G statistic score. Based on the G score, the cells are returned in order of decreasing order of their G scores. A cell is nothing but a space time cube where the x axis represents the latitude of the pickup location, y axis represents the longitude of the pickup location and z axis represents the pickup date.

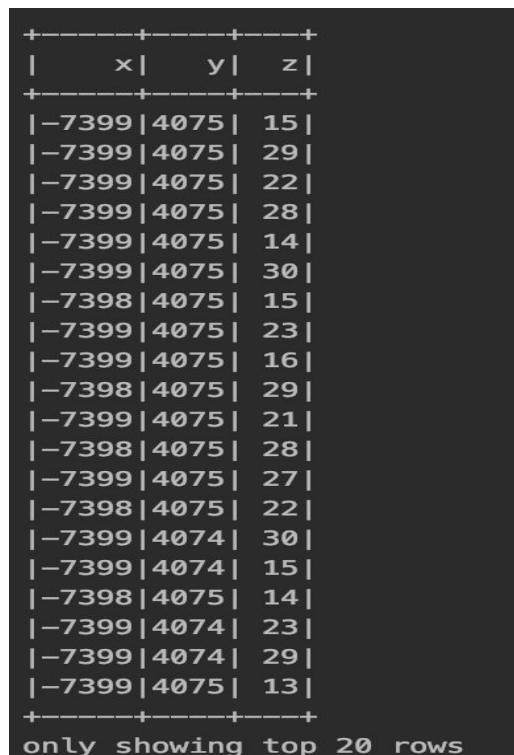
Name: runHotcellAnalysis

Input: spark SparkSession, String pointPath

Output: sorted dataframe representing cells and their hotness

Steps:

1. Find the points from the pointpath
2. Spark SQL is run which will extract all points with x, y and z columns in a dataframe.  
Essentially, X gets the latitude of pickup point, Y gets the longitude of pickup point and Z gets the pickup date. All of this data is in a dataframe.
3. On the extracted data, Getis-Ord statistic value is calculated for each of the cells from the dataframe. The cells with x,y and z values are returned in the decreasing order of the Getis - Ord statistic values.



x	y	z
-7399	4075	15
-7399	4075	29
-7399	4075	22
-7399	4075	28
-7399	4075	14
-7399	4075	30
-7398	4075	15
-7399	4075	23
-7399	4075	16
-7398	4075	29
-7399	4075	21
-7398	4075	28
-7399	4075	27
-7398	4075	22
-7399	4074	30
-7399	4074	15
-7398	4075	14
-7399	4074	23
-7399	4074	29
-7399	4075	13

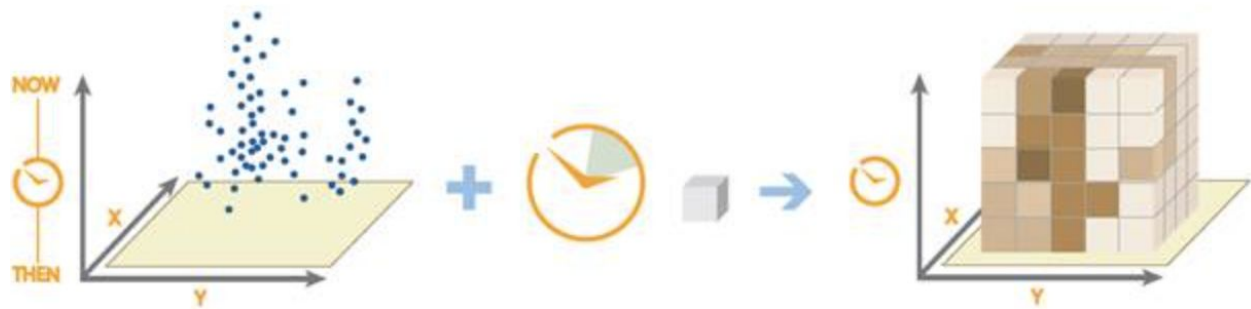
only showing top 20 rows

Fig 7: HotCell Output

### **Space Time Cube**

The cell mentioned above is Hot Cell which is a 3 dimensional space with x values representing the latitude of pickup location, y values representing the longitude of the pickup location and z values representing the pickup date. In addition, the difference between two cells in terms of x and y is 0.01. Also, the two cells are separated in z axis by 1 day.

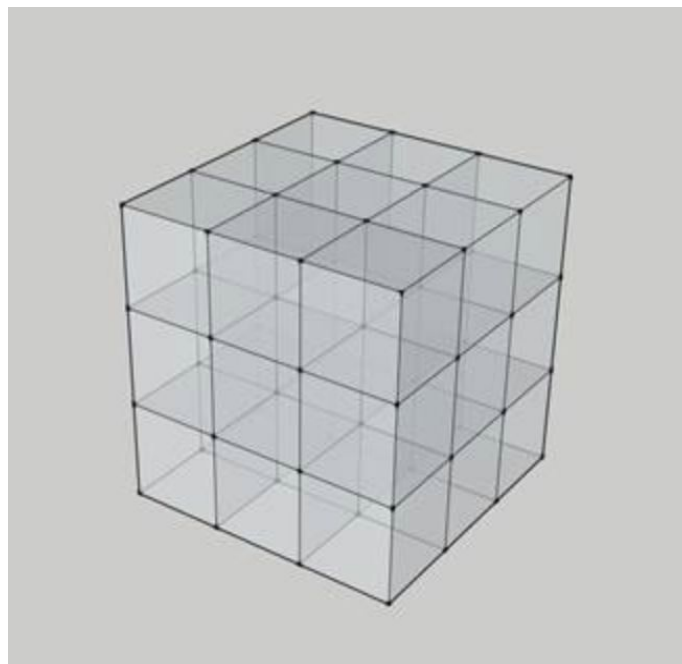
The value of the cell is the number of trips that happened on a particular day.



**Fig 8: Space Time Cube**

### **Spatial Weight of Adjacent Cells**

For a given cell, adjacent cells are the cells that share an edge or vertex of the given cell. For adjacent cells, spatial weight of 1 is given for the cells. For non adjacent cells, spatial weight of 0 is given. The spatial weight of adjacent cells are used for calculating Getis Ord Statistic value which in turn represents the hotness of the cells.



**Fig 9: Space Time Cube**

### **Getis-Ord Statistic calculation**

Every cell is assigned a hotness value which is measured by the Getis-Ord statistic value. The formula for calculating the Getis-Ord Statistic value is given below:

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

**Fig 10:** Formula for Calculating G score

where

$x(j)$  is the attribute value of cell  $j$

$n$  is the total number of cells

$w(i,j)$  is the spatial weight between cell  $i$  and cell  $j$  and

Mean is calculated in the following way

$$\bar{X} = \frac{\sum_{j=1}^n x_j}{n}$$

**Fig 11:** Mean Formula

Standard deviation is calculated in the following way

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

**Fig 12:** Standard Deviation Formula

## **CONCLUSION**

Initially, when the project was introduced Apache Spark was something most of the team members never worked on. When the project details were introduced, since most of us were very new to Apache Spark we spent a lot of time learning how Spark works and setting up Spark in our machines was a huge task in itself. The learning curve during the start of the project was very high. Furthermore, Scala which was the programming language to be used was new to most of us again. As a result, this was the next challenge. After this the work was divided to implement the User defined functions ST\_Contains and ST\_Within and to work on the four spatial queries in phase 1. Forming the cluster and running them took a while.

Furthermore, for phase 2 which has two tasks, Hot Zone Analysis task was done as it was quite similar to what we had to do in phase 1 but then Hot Cell Analysis was something new and it took time to understand what actually had to be done and what calculations were needed to be done to identify a zone as statistically significant. Moreover, testing and debugging this task so that it runs successfully and is able to provide an output within 30 minutes as per the requirements was another challenging task. Even though we came across so many challenges, while doing the project we learnt how useful and powerful Apache Spark is. Even though it is somewhat new compared to Hadoop it is on the rise. It is a very powerful tool and helps in heavy Data analytics including live streaming as that of the Taxi data which is used in the implementation of this project.