Large-scale Geospatial Data Analysis using SparkSQL

Abhijit Chakraborty
Arizona State University
achakr40@asu.edu

Aaron Stanek
Arizona State University
astanek2@asu.edu

Zachary Gateley
Arizona State University
zgateley@asu.edu

Aaron Nugent
Arizona State University
awnugent@asu.edu

ABSTRACT

With the influx of the ubiquitous collection of spatio-temporal observational data like vehicle tracking data, the identification of unusual patterns of data in a statistically significant manner has become a key problem that many businesses and organizations are trying to solve. Spatial data analysis or spatial statistics includes any of the formal techniques which study entities using their topological, geometric, or geographic properties. Here, spatial statistics is applied to New York City taxi data to identify the fifty most significant hot zones within the city as specified by the ACM SIGSPATIAL Cup 2016. The analysis of the spatio-temporal big data is done using the Spark SQL module of the Apache Spark cluster framework.

KEYWORDS

Distributed and parallel databases, Apache Spark, Apache Hadoop, PostgreSQL, SparkSQL, Amazon Web Services, GeoSpark, largescale, Geospatial data analysis, Spatial query analysis.

1 INTRODUCTION

Geospatial data or spatial data is information that has a geographic aspect to it. In other words, the records in this type of information set have features like coordinates, an address, city, postal code or zip code, included with them. In recent years, there has been a massive explosion in the amount of spatial data generated. This is due to applications like smart phones and Internet of things (IoT) devices that use a variety of sensors. These sensors generate great volumes of data with spatial dimensions.[1] These spatial features play a vital role in many mobile applications. For example, Uber and Lyft use the customer and driver locations as the most important features.

Analysis of geospatial data is most often a complex problem due to high density i.e it contains a very large number of data points. This requires a great deal of computational and processing effort to get valuable insights and better understanding of the data on a high level. Querying and analyzing such voluminous spatial data to achieve high throughput and low latency is of course a challenge. Therefore, this is a very tedious problem that many organizations are trying to solve nowadays.

Most of the traditional and existing databases used in spatial data analytics are disk oriented. Even though they are optimized for I/O efficiency, their performance degrades when scaling up to large voluminous spatial data. To solve this problem, a number of large scale distributed data processing frameworks have emerged and efficiently handle the heavy workloads. These distributed frameworks have characteristics like fault tolerance and replication to ensure the integrity of data. Two such popular open source distributed frameworks are Apache Hadoop and Spark. This project report describes the use of these frameworks in a distributed cloud cluster to analyze the New York City taxi data from the years 2009 to 2012 with an objective of identifying hot zones for pickups in different areas in the city.

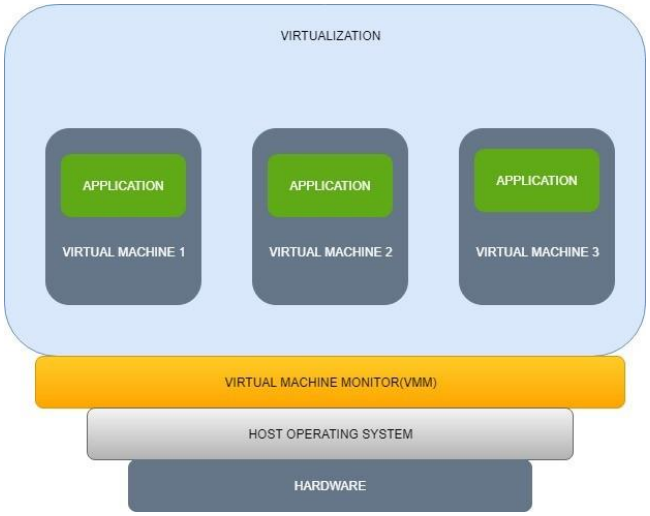


Figure 1: Virtual Machine architecture (Iaas)

The system developed as part of this project uses 3 Elastic Cloud Compute (EC2) instances of the Amazon Web Services with the Apache Spark and Hadoop frameworks installed. The spatial data analysis is carried out by the GeoSpark Application Programming Interface on Apache Spark. The querying of data is done with the SparkSQL module of Apache Spark.[2] Given a dataset of points and rectangles, the objective is to determine which set of points lie within the given set of rectangles.

The ACM SIGPATIAL Cup 2016 website describes a statistic ‘Getis - Ord Gi*‘, which is used to calculate the hot zones based on the number of customer pickups at a location. The taxi trip data set is pre-processed and converted to a space time cube of latitude by longitude by day in the month when a particular pickup took place. A cell in this space time cube consists of a range of locations per day and the ‘Getis - Ord Gi*‘ statistic

determines the spatial proximity of each cell by weighting the values with respect to other cells.

2 SYSTEM ARCHITECTURE

For large-scale geospatial data analysis, it is important to have a reliable and fault-tolerant system capable of scaling itself as per the needs of the input data. Some of the important factors considered to build such systems include performance, efficient computation cost, in-built monitoring of VMs, fast recover from failures and most importantly scalability and load balancing.

2.1 Connectivity and Infrastructure

Amazon Web Services EC2 instances have been chosen for our project as the cloud infrastructure, as it meets all the requirements required for a large-scale geospatial data analysis which means it has inbuilt disk imaging capabilities and a variety of snapshots, incorporates adaptive control and monitoring for VM instances, and is also cost-effective. With the use of EC2 instances, we achieved high network and computing performance for our complex computational workloads such as the spatial analysis of New York City taxi data.

The system architecture consists of one master node ("dds-master") and two worker/slave nodes ("dds-slave-1" and "dds-slave-2"), all of which are present in the same local network.

Set up of EC2 Instances:

SSH is used to login to remote servers for execution of commands and programs through Command Line tools (CLI). The above EC2 instances communicate with themselves as well as the other instances through password-less SSH. Also, these instances are made accessible on web interface using port-forwarding and addition of certain instance rules. The below figure is a representation our architecture.

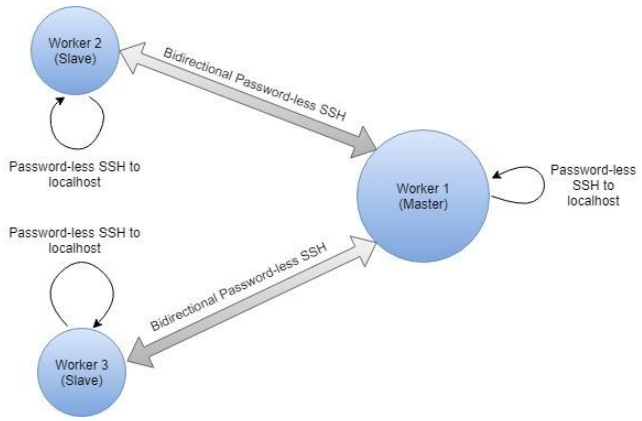


Figure 2: AWS EC2 instances system architecture

Below is the configuration details of our EC2 instances as seen in AWS console on the web interface. It shows the instance name, instance-id, instance-type, availability zone, instance state, and public DNS.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
dds-master	i-0359061e398761df4	t2.medium	us-west-2b	running	2/2 checks ...	None	ec2-54-214-230-104.us...	54.214.230.104
dds-slave-1	i-0aa392846b4e7a4...	t2.medium	us-west-2b	running	2/2 checks ...	None	ec2-54-214-63-235.us-...	54.214.63.235
dds-slave-2	i-0b04c77c7787e7b71	t2.medium	us-west-2b	running	2/2 checks ...	None	ec2-34-217-146-206.us...	34.217.146.206

Figure 3: System configurations

2.2 Frameworks and Modules

For geospatial data analysis, Apache Hadoop and Apache Spark are installed and integrated.

2.2.1 Apache Hadoop. Apache Hadoop[4] is a set of software utilities that facilitate using a network of master-slaves to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using Hadoop Distributed File System (HDFS) and the MapReduce programming model. HDFS incorporates replication of data and hence supports data integrity.

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ip-172-31-20-135.us-west-2.compute.internal:50010 (172.31.20.135:50010)	0	In Service	29.02 GB	24 KB	6.32 GB	22.68 GB	0	24 KB (0%)	0	2.7.7
ip-172-31-25-209.us-west-2.compute.internal:50010 (172.31.25.209:50010)	1	In Service	29.02 GB	24 KB	3.51 GB	25.49 GB	0	24 KB (0%)	0	2.7.7
ip-172-31-25-16.us-west-2.compute.internal:50010 (172.31.25.16:50010)	1	In Service	29.02 GB	24 KB	3.62 GB	25.39 GB	0	24 KB (0%)	0	2.7.7

Figure 4: Hadoop Live Nodes

Hadoop is installed on all the three instances. All the necessary configurations are made to make the master node identify the worker nodes. The status of the Hadoop cluster can be verified by entering "master_ public_ip:50070" in the browser. On successful installation, the below page with all relevant details is obtained. The three nodes indicate the 3 live nodes of the hadoop cluster.

Data in the form of CSV files are loaded into HDFS for further computation. This is done using the command line interface, and can be verified using the web interface of Hadoop server.

2.2.2 Apache Spark. Apache Spark[5] is an open-source distributed general-purpose and in-memory cluster-computing framework. It provides an interface performing operations on entire clusters of Resilient Distributed Dataset (RDD) with implicit data parallelism and fault tolerance.

To run Spark tasks, the spark server is installed on all the instances, and started on the master node. The status of the Spark master can be verified by entering "master_public_ip:8080" in the browser, which provides the below page on success. The three nodes present indicate the three instances that were set-up.

The spark cluster reads the data present in HDFS, and performs operations and computations using APIs, and GeoSpark APIs in

Worker Id	Address	State	Cores	Memory
worker-20181207061658-172.31.20.135-43367	172.31.20.135:43367	ALIVE	2 (0 Used)	2.9 GB (0.0 B Used)
worker-20181207061659-172.31.25.16-43891	172.31.25.16:43891	ALIVE	2 (0 Used)	2.9 GB (0.0 B Used)
worker-20181207061659-172.31.25.209-35189	172.31.25.209:35189	ALIVE	2 (0 Used)	2.9 GB (0.0 B Used)

Figure 5: Spark workers overview

our scenario. No extra configurations are required for this step.

2.3 GeoSpark

GeoSpark is a computing system that processes large-scale spatial data. It extends Apache Spark and SparkSQL with a set of spatial Resilient Distributed Datasets (RDD) that efficiently loads, processes, and analyzes large-scale spatial data across machines. It is a complete spatial analytics system that provides the best performance.

Geospark is mainly used for computation of complex geometris and spatialSQL. It consists of three parts - GeoSpark-core which comprises of SpatialRDDs and query operators, GeoSpark-SQL which is an SQL interfaces for GeoSpark core, and GeoSpark-Viz which is the visualization extension of GeoSpark core. Here, we use GeoSpark for analysis of large-scale Geospatial data.

2.4 Setup Considerations

2.4.1 Virtual Machine, Hadoop and Spark setup. Virtual machines were setup during this phase and haddop and spark were respectively installed. As no prebuilt VM images containing Hadoop and Spark were used, this particular phase was a bit challenging mainly due to setting up the network configurations, environment variables, XML setting files and also editing the Bash script to install and run Hadoop and Spark successfully. As these virtual machines were present on AWS, SSH (in Linux) or Putty (In windows) was used to access these VMs.

2.4.2 Java ARchive (JAR) testing. During this phase, an initial test was performed using the Spark Interactive Shell to load an example API as a single JAR file. This is done to verify whether the setup in the previous phase was successful and also if the JAR files are able to execute on the machines. The GeoSpark API is loaded along with importing relevant libraries that allow the data to be loaded and apply RDD spatial operations on it.

2.4.3 Cost Effectiveness and System Performance. As AWS has a plethora of EC2 instances, balancing cost effectiveness and system performance was a key task. Therefore, t2.micro instances were used as workers to give a balance between cost and performance.

2.4.4 Password-less SSH. In this phase, the worker nodes were configured to have the ability to SSH into each other without using passwords to do so. The following steps were performed to achieve this:

(1) Generate the public and private keys for each worker machines using "ssh-keygen" command.
(2) Copy all the public keys of the worker nodes onto the master nodes authorized_keys file.
(3) Copy the public keys of the each machines onto itself in the authorized_keys file. This will give the ability for every machine to communicate bidirectionally with every other machine as well.
(4) Test on every machine if you can ssh without using the password by typing the command "ssh ubuntu@target_machine".

2.4.5 Hadoop Cluster.

(1) Make the following changes in the file as part of initial setup:
- File: hadoop-env.sh
  Property: JAVA_HOME
  Value: /usr/lib/jvm/JAVA-FOLDER-LOCATION
- File: core-site.xml Property: fs.default.name
  Value: hdfs://master:54310
- File: mapred-site.xml
  Property: mapred.job.tracker
  Value: master:54311
- File: hdfs-site.xml
  Property: dfs.replication
  Value: 3

(2) Add the following lines in the HadoopFolder/etc/Hadoop/slaves file on the master machine:
- master
- slave-1
- slave-2

(3) Use the command "hadoop namenode-format" to format the HDFS system.
(4) Start the process on the master machine using the command "sbin/start-all.sh".
(5) Test if the installation was successfully and the process is active by opening "localhost:50070" in a web browser.

2.4.6 Spark Cluster. Spark cluster are setup on 3 machines using the steps mentioned below:

(1) In file "SparkFolder/conf/spark-env.sh", add the line "SPARK _MASTER_IP=master".
(2) Start the spark process by executing the command "sbin/startmaster.sh".
(3) Open "localhost:8080" on a web browser to check the status of the spark master.
(4) On the 2 worker machines, execute the command below command in SparkFolder/bin folder to register and run the worker in Spark master:
Command: ./spark-class org.apache.spark.deploy.worker.Worker spark//master:7077

3 GEOSPATIAL DATA ANALYSIS

3.1 SparkSQL

SparkSQL[6] is used to run four spatial queries namely Range Query, Range Join Query, Distance Query, and Distance Join Query. These queries use two user defined functions in order to successfully complete the task. The two user defined functions

are named ST_Contains and ST_Within. These two functions had to be implemented.

ST_Contains takes two parameters, a point and a rectangle. It returns a boolean value indicating whether the given point is within the rectangle or not. The algorithm for the method is as follows:

Function: ST_Contains
Input: String point, String rectangle
Output: Boolean Steps:

(1) "point" string contains x and y coordinate of the point separated by ",". Hence it is split into coordinates and assigned to variable x and y respectively.
(2) "rectangle" string contains two points which are at the end of one of the diagonal of the rectangle. Hence the rectangle string is split into 2 points which in turn have 2 coordinates and have been assigned to x1, y1, x2, y2 respectively.
(3) If the value of x falls between x1 and x2 and the value of y falls between y1 and y2, a boolean value of True is returned indicating the rectangle contains the point.
(4) Else a boolean value of False is returned indicating the rectangle does not contain the point.

ST_Within takes three parameters, a point p1, another point p2 and a double value distance. It returns a boolean value indicating whether the distance between point p1 and point p2 is less than the distance value provided or not. The algorithm for the method is as follows:

Function: ST_Within
Input: String p1, String p2, Double distance
Output: Boolean Steps:

(1) "p1" string contains x and y coordinate of the point separated by ",". Hence it is split into coordinates and assigned to variable x1 and y1 respectively.
(2) "p2" string contains x and y coordinate of another point separated by âĂIJ,âĂİ. Hence it is split into coordinates and assigned to variable x2 and y2 respectively.
(3) Find Euclidean distance between p1 and p2 using the below formula and assign the result to variable d.

$$d = \sqrt{((x1 - x2)2 + (y1 - y2)2)}$$

(4) If the value of d is less than or equal to given distance value, then a boolean value of True is returned indicating that point p1 is within the given distance from point p2.
(5) Else a boolean value of False is returned indicating the point p1 is not within the given distance from point p2.

As mentioned previously, the above functions are used to run 4 spatial queries which are explained below.

(1) Range Query: This query takes a rectangle R and a set of points P as input. The query will return all the points among P which lies inside rectangle R. This query uses ST_Contains user defined function which was described previously.

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

Figure 6: Range query output

(2) Range Join Query: This query takes a set of rectangles and a set of points as input. The query will return all rectangle, point pairs which satisfy the condition that rectangle contains the point. The query uses ST_Contains user defined function which was described previously.

```
+--------------------+--------------------+
|                 _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
```

Figure 7: Range join query output

(3) Distance Query: This query takes a point P and distance D as input. The query will return all points that lie within distance D from point P. The query uses ST_Within user defined function which was described previously.

(4) Distance Join Query: This query takes two set of points P_set1 and P_set2, and a distance D as input. The query will return all pairs of points p1, p2 which satisfy the condition

```
+--------------------+
|                 _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
```

Figure 8: Distance query output

that p1 belongs to P_set1, p2 belongs to P_set2 and p1 is within distance D from p2. The query uses ST_Within user defined function which was described previously.

The above functions are implemented and packaged in a jar which can be provided to spark submit application as shown in the below diagram. Then the queries are run using Spark submit to get the results.

3.2 New York Taxi Cab Data Geospatial Analysis

The dataset provided contains all records of New York City Yellow Cab taxi trips from 2009 to 2012. This dataset is filtered to contain records of only five New York city boroughs to minimize the noise. This is done by selecting locations which have latitude ranging from 40.5N to 40.9N and longitude ranging from 73.7W to 74.25W. The below figure represent the dataset mentioned with blue dots representing the pickup locations.

It was required to do spatial hot spot analysis on the provided dataset. Two hotspot analysis tasks were defined namely Hot Zone analysis and Hot Cell analysis.

3.2.1 Hot Zone Analysis. This is analyzed on a small subset of the dataset mentioned. The analysis task takes a rectangle dataset and a point dataset and runs Range Join query on it. The result is then grouped on rectangle so that count of all the points within each rectangle can be obtained. The rectangle with more points is considered hotter than the ones with less number of points in it.

```
+--------------------+--------------------+
|                 _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
```

Figure 9: Distance join query output

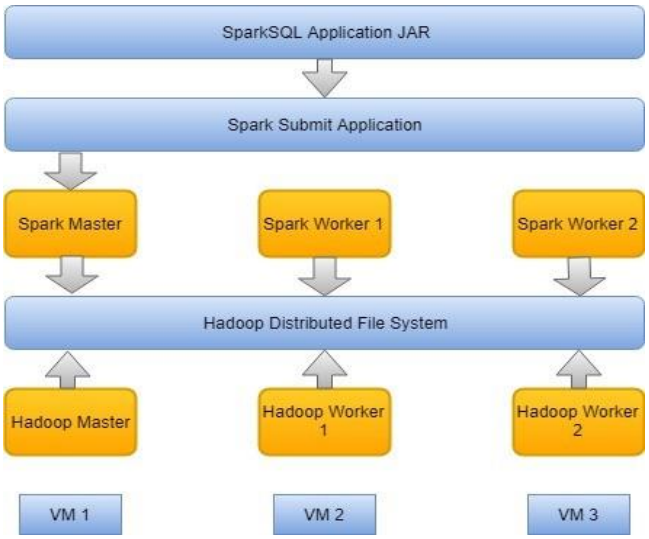


Figure 10: Integrated SparkSQL and Hadoop architecture

Name: runHotZoneAnalysis
Input: SparkSession spark, String pointPath, String rectanglePath Output: Sorted dataframe representing rectangle and the number of points in it (hotness). Steps:

Figure 11: New York Taxi Cab Pickup Locations

(1) Get all the points from pointPath.
(2) Get all the rectangles from rectanglePath.
(3) Run a spark join query on the above two sets with join condition obtained from ST_Contains method to get the join result dataframe.
(4) Run a spark group by query on the join result dataframe grouping on rectangle and count(points) as aggregation function sorted by ascending order of rectangle to obtain sorted dataframe of rectangles and their hotness.

```
+--------------------+-----------+
|           rectangle|numOfPoints|
+--------------------+-----------+
|-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|
+--------------------+-----------+
only showing top 20 rows
```

Figure 12: Hotzone Analysis output

3.2.2 Hot Cell Analysis. This is analyzed on the given New York trip dataset. For this task, a list of cells and their hotness (Getis Ord satistic value) in descending order has to be returned. A cell is a space-time cube with x-axis representing the latitude of the pickup location, y-axis representing the longitude of the pickup location and z-axis representing the date of the pickup location.

Name: runHotcellAnalysis
Input: SparkSession spark, String pointPath Output: sorted dataframe representing cells and their hotness Steps:

(1) Get all the points from pointPath.
(2) Run spark sql query on all the points which will output a dataframe with x, y and z as columns. X gets the latitude extracted from the point, y gets the longitude extracted from the point and z gets day of the pickup. Result is a cell dataframe.
(3) Find Getis - Ord Statistic value for each of the cell in the dataframe and return the top 50 cells in the descending order of the Getis - Ord Statistic value (hotness).

```
+-----+----+---+
|    x|   y|  z|
+-----+----+---+
|-7399|4075| 15|
|-7399|4075| 22|
|-7399|4075| 14|
|-7399|4075| 29|
|-7398|4075| 15|
|-7399|4075| 16|
|-7399|4075| 21|
|-7399|4075| 28|
|-7399|4075| 23|
|-7399|4075| 30|
|-7398|4075| 14|
|-7399|4074| 15|
|-7398|4075| 22|
|-7399|4075| 27|
|-7398|4075| 29|
|-7398|4075| 28|
|-7399|4074| 23|
|-7399|4074| 22|
|-7399|4074| 16|
|-7398|4076| 15|
+-----+----+---+
only showing top 20 rows
```

Figure 13: Hotcell Analysis output

Space Time Cube:

The cell mentioned above is the space time three dimensional cube where x-axis represents the latitude of the pickup location, y-axis represents the longitude of the pickup location and z-axis represents the day of pickup. As mentioned in the specification, two cells are separated on the x-axis and y-axis by a distance of 0.01 units, i.e, if the difference between the x value or y value of 2 cells is 0.01 units. The cell are separated on z-axis by 1 day. The cell value is nothing but the number of trips that happened

within the latitude and longitude range and on that particular day.

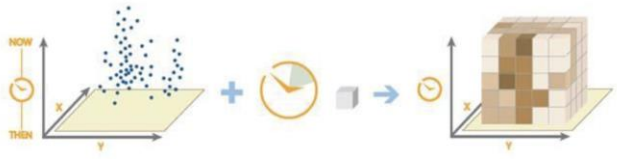


Figure 14: Space time cube

Spatial weight for adjacent cells:

For a given cell, adjacent cells are the cells which either share an edge or vertex with the given cell. From the perspective of the given cell, all of its adjacent cells are given a spatial weight of 1 and all the non adjacent cells are given a spatial weight of 0. Spatial weights are used for the calculation of Getis - Ord Statistic value which represents the hotness of the cell.

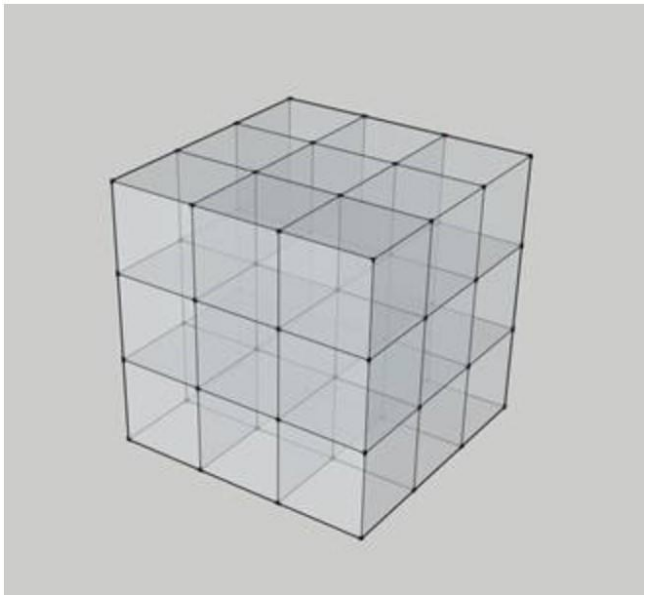


Figure 15: Space weight cube with 16 cells

Getis - Ord Statistic calculation:

Every cell has to be assigned a hotness value in order to find the topmost hottest (significant) cells. This hotness measure is obtained from the Getis - Ord Statistic.[7] The formula for the Getis - Ord Statistic is as follows:

where, x(j) : attribute value of
cell j n : total number of cells
w(i,j) : spatial weight between cell i and j (as defined in the previous section) and

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

Figure 16: Getis - Ord Statistic

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

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

4 EXPERIMENTAL SETUP

4.1 Cluster Settings

The system is implemented using Amazon Web Services EC2 instances with the following configurations:[8]

Ubuntu Base image - ubuntu-18.04-amd64-server (ami-0bbe6b35405ecebdb) Instance type: t2.medium
Availability zone: us-west-2b
Apache Hadoop version: 2.7.7
Apache Spark version: 2.3.2

All commands are entered and run in an Ubuntu Terminal. In order to execute the jobs, we first start the Hadoop and Spark servers and the jobs are run. In the Hotzone and Hotcell analysis, we use the following command:

./bin/spark-submit /GitHub/CSE512-Project-Hotspot-AnalysisTemplate/target/scala-2.11/CSE512-Project-Hotspot-Analysis-Temp late-assembly-0.1.0.jar test/output hotzoneanalysis src/resources/po int-hotzone.csv src/resources/zone-hotzone.csv hotcellanalysis src/res ources/yellow_tripdata_2009-01_point.csv

The I/O cost and communication costs can be determined by this command.

4.2 Data and Query Workload

In order to execute our jobs, the input data files are uploaded to Hadoop Distributed File System (HDFS), and then passed as a parameter to the spark-submit command. Below is the screenshot of our HDFS used for spatial query analysis.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	ubuntu	supergroup	206.56 KB	12/7/2018, 9:33:19 PM	3	128 MB	arealm10000.csv
-rw-r--r--	ubuntu	supergroup	1.76 MB	12/8/2018, 8:18:51 AM	3	128 MB	point-hotzone.csv
-rw-r--r--	ubuntu	supergroup	17.56 MB	12/8/2018, 8:19:05 AM	3	128 MB	yellow_trip_sample_100000.csv
-rw-r--r--	ubuntu	supergroup	409.97 KB	12/7/2018, 9:33:33 PM	3	128 MB	zcta10000.csv
-rw-r--r--	ubuntu	supergroup	11.73 KB	12/8/2018, 8:19:18 AM	3	128 MB	zone-hotzone.csv

Figure 17: HDFS directory overview

Above is a screenshot of the details of the input data files present in our HDFS used for Hotzone and Hotcell analysis. The files "pointhotzone.csv" and "zone-hotzone.csv" are used for Hotzone analysis. They are relatively smaller in size, with their computation time also being less. On the other hand, the file "yellow_trip_sample_100000.csv" is larger. And we see that the computation of Hotcell analysis using Getis - Ord is relatively higher because of the use of adjacency lists.

4.3 Monitoring Setup

Execution of tasks can be monitored in several ways:

(1) AWS EC2 instances -
(2) Spark - Spark provides a web interface on port 8080 through which the execution time and performance can be monitored. The health of the nodes can be monitored as well.

5 EVALUATION OF RESULTS

5.1 Hadoop and Spark Analysis

Given a query point and a query rectangle, the function ST_Contains checks if the rectangle fully contains the point. On-boundary points are also considered. SparkSQL uses this function to perform spatial queries such as range query and range join query. Given a query rectangle R and a set of points P, the range query finds all the points within R.And, given a set of rectangles R and a set of points S, the range join query finds all the point-rectangle pairs such that the point is within the rectangle.

Given two query points and a distance, another function ST_Within checks whether the two points are within the given distance. SparkSQL uses this function to perform the distance query and distance join query spatial queries. Given a point location P and distance D, the distance query finds all points that lie within a distance D from P. And, given a set of points S1 and a set of points S2 and a distance D, the distance join query finds all (s1, s2) pairs such that s1 is within a distance D from s2. The execution flow of SparkSQL can be seen below:

In hot zone analysis, we perform a range join operation on a rectangle dataset and a point dataset. For each rectangle, the number of points located within the rectangle is obtained. A hotter rectangle means that it includes more points. The output of this job is the dataframe with all zones with their count, sorted by the "rectangle" string in ascending order.

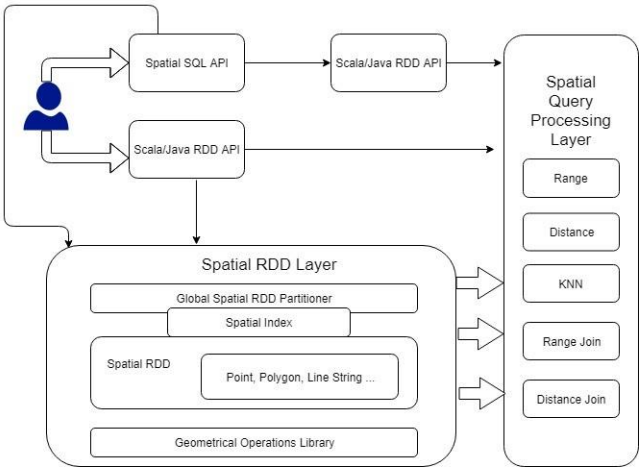


Figure 18: SparkSQL execution workflow

The hot cell analysis focuses on applying the spatial statistics to spatio-temporal big data in order to identify statistically significant spatial hot spots using Apache Spark. Here, we implement a Spark job that calculates the Getis - Ord statistic of the New York City Taxi Trip datasets. The output of this job is the dataframe with coordinates of top 50 hottest cells sorted by their G score in descending order.

5.2 VM System Workload

Monitoring system workload is an important part of maintaining the performance, reliability, and availability of the Amazon Elastic Compute Cloud (Amazon EC2) instances.[9] Here, certain VM usage metrics such as CPU Utilization, Network Utilization and Memory Utilization on AWS are collected and analysed.

Because of the usage of free-tire computing services, each data point in the below graphs covers the previous 5 minutes of activity for the instance.

CPU Utilization: It is the percentage of allocated EC2 compute resources that are currently used by the instance. It identifies the processing power required to run an application upon a selected instance. The instance-type used is t2-medium. Hence, the operating system shows a considerably lower percentage when the instances are not allocated a full processor core.

Network-In: It is the number of bytes received on all network interfaces by the instance. It identifies the volume of incoming network traffic to a single instance. The number denotes number of bytes received during the period.

Network-Out: It is the number of bytes sent out on all network interfaces by the instance. It identifies the volume of outgoing

network traffic from a single instance. The number denotes the number of bytes sent during the period.

Below are the metrics for the master, slave-1 and slave-2 nodes:

From the above graphs, we see that there is a spike in the CPU utilization when the job is submitted to the spark master. We also see a rise in the CPU usage on all the slave nodes due to the execution of the job.

We also see a rise in the Network utilization of the master once the job is submitted to the spark master. It is also observed in the other two slave nodes. These metrics are because of the number of bytes received on all network interfaces by the instance.

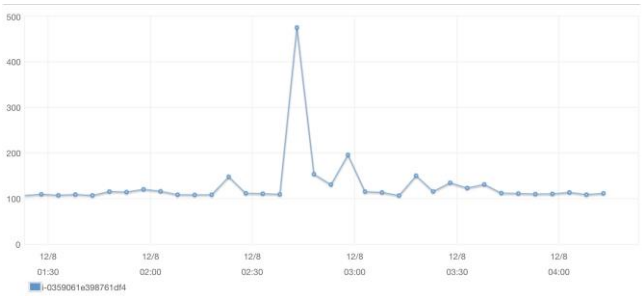


Figure 20: Network Packets - In on Master node

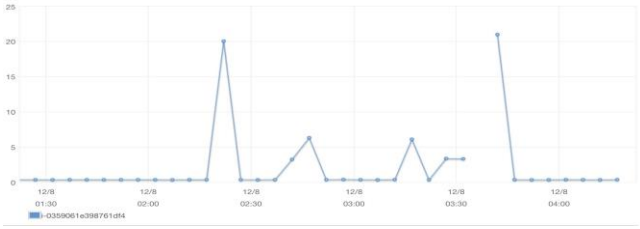
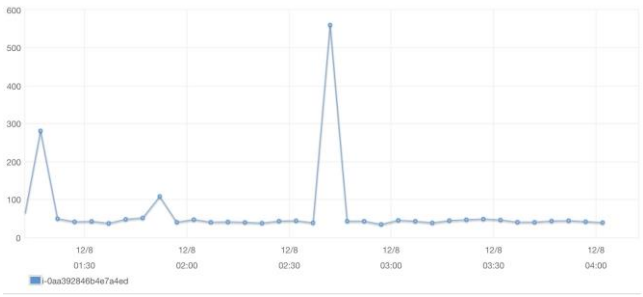


Figure 19: CPU Utilization on Master node



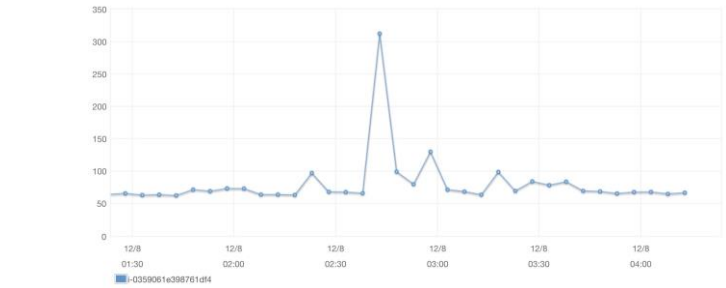


Figure 21: Network Packets - Out on Master node

Figure 22: CPU Utilization on Slave-1

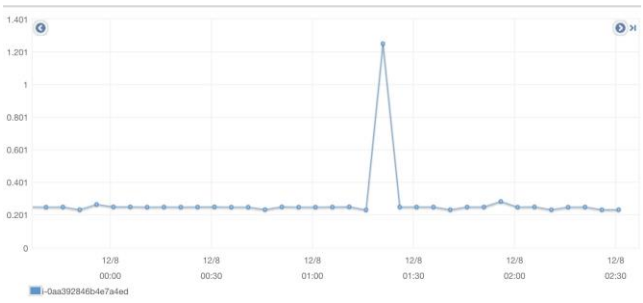


Figure 23: Network Packets - In on Slave-1

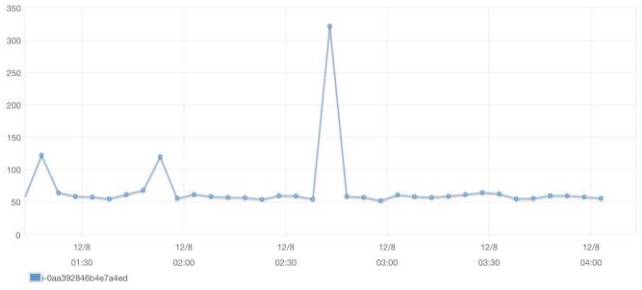


Figure 24: Network Packets - Out on Slave-1

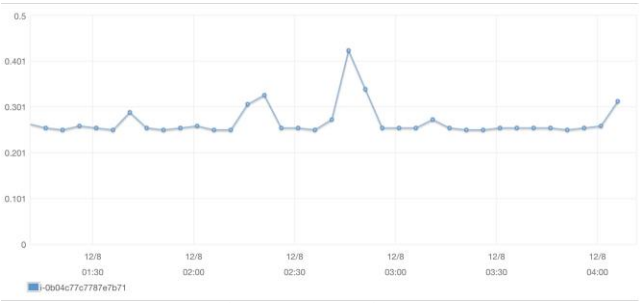
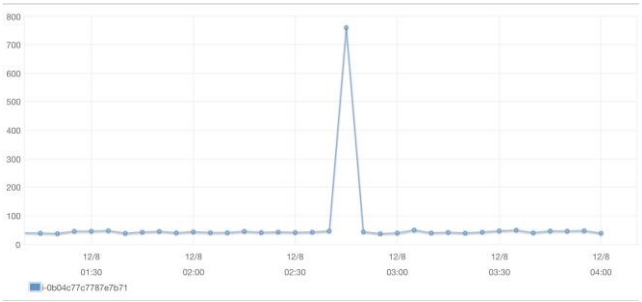


Figure 25: CPU Utilization on Slave-2



[2] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, Minyi Guo: *Simba: Efficient In-Memory Spatial Analytics*. SIGMOD 2016 1071-1085
[3] Zhou Huang, Yiran Chen, Lin Wan, Xia Peng: *GeoSpark SQL: An Effective Framework Enabling Spatial Queries on Spark*. ISPRS International Journal of GeoInformation
[4] https://hadoop.apache.org/
[5] https://spark.apache.org/
[6] http://datasystemslab.github.io/GeoSpark/
[7] http://sigspatial2016.sigspatial.org/giscup2016/home
[8] https://www.youtube.com/watch?v=f-HSJ0eqtpY
[9] https://aws.amazon.com/
[10] http://resources.esri.com/help/9.3/arcgisengine/java/gp_toolref/ spatial_statistics_tools/how_hot_spot_analysis_colon_getis_ord_gi _star_spatial_statistics_works.htm

Figure 26: Network Packets - In on Slave-2

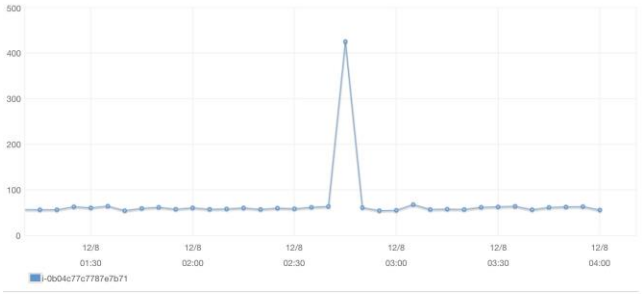


Figure 27: Network Packets - Out on Slave-2

6 CONCLUSIONS

In this project, the fifty most significant hot zones were identified using the New York CIty taxi cab dataset as specified by ACM SIGSPATIAL Cup 2016. The problem of large scale geospatial data processing was solved using the Apache Spark framework. Apache Spark
was thus a good choice for achieving low latency and high throughput. Important features like in-memory computations, caching and fault tolerance helped in reducing the run-time of our system operations which would have taken a considerable amount of time otherwise considering the amount of data.

Spark along with Hadoop distributed file system is a flexible storage platform for storing data consisting of large files. Understanding Apache Spark coupled with Hadoop helped us understand the need for a Distributed and Parallel Database System and of its numerous advantages. The use of SparkSQL APIs on Scala helped us to analyze geospatial data effectively.

7 ACKNOWLEDGEMENTS

Firstly, we would like to thank our professor Prof.Samira Ghayekhloo for giving us the opportunity to implement this project that enabled us to gain in-depth knowledge of distributed and parallel database systems. Next, we would like to thank the teaching assistants for guiding us throughout the course. We would also like to thank our peers for their constant support during the course of the project.

REFERENCES

[1] J ia Yu, Zongsi Zhang, Mohamed Sarwat: *Spatial Data Management in Apache Spark: The GeoSpark Perspective and Beyond*. GeoInformatica Journal, 2018