# Distributed Hot Spot Analysis over Spatio-temporal Data Using Apache Spark

Anurag Natthuram Lengure
CIDSE, ASU
1217340644

Harsh Kavdikar
CIDSE, ASU
1216890779

Kshitij Jayprakash Sutar
CIDSE, ASU
1217539440

## Abstract

The main drawback of traditional databases is that they are not designed to provide the power of geospatial data to perform large scale analytics on top of it. Also, that these databases don't employ geospatial data to partition and store the data efficiently which is essential to speedup analytics queries. Any performance compromise is significant when running on large data. Hence to compute this big Spatio-temporal data we have used Spark SQL module of Apache[7] cluster which provides an interface for programming clusters with fault tolerance and data parallelism.

## Keywords

Apache Hadoop, Apache Spark, SparkSQL, AWS Cloud Watch, Amazon Web Services EC2, Geo-spatial data, Spatial query, Spatial-temporal data

## 1 Introduction

A major peer-to-peer taxicab firm planned to develop and run multiple spatial queries on a large database containing geographic data along with real-time location data of their customers. Geo-spatial data is a special type of data that combines physical space with data attributes. In current years, there has been a huge increase in the amount of spatial data generated due to applications like the Internet of things devices and smartphones. Geo-spatial data could be static or dynamic. In this case we are given a set of dynamic data, which has a third dimension i.e. time, with the physical two-dimensional data.This project aims to help the taxi business employ it's big data capabilities to make data-driven strategic decisions by understanding it's geospatial data.[1]

A spatial query is a special type of query supported by geo-databases and spatial databases. The queries differ from traditional SQL queries in that they allow for the use of points, lines, and polygons. Given geospatial data consist of a large number of points and hence performing analysis on this kind of data needs heavy computation in order to deliver with the least latency.Apache Spark is used to carry out this processing of big data and handle the complex and computationally intensive spatial queries. We have performed spatial queries such as range, distance, range join and distance join query for the first phase of this project. As part of the second phase of the project, we implemented hot zone analysis and hot cell analysis where the Hot zone was finding the number of data points present in a rectangle that is defined by its diagonal points.[2] It was used to create a heat map with varying colors to suggest the hotness of the region, whereas, in Hot cell analysis, we applied spatial statistics to Spatio-temporal data to identify statistically significant spatial hot spots.

## 2 Methodology

### 2.1 System Architecture

This project consists of large geospatial data analysis task, therefore the system should be able to scale itself as well as be fault-tolerant. The factors such as performance, reliability and load balancing should also be considered.

Amazon Web Services(AWS) provides EC2 instances which provides the above mentioned benefits. This project is configured on a master-slave system architecture, which consists of one master node and two worker nodes. These EC2 instances needs to be set up to execute the spatial data analysis queries in Spark. The steps involved are Passwordless SSH Login, Hadoop Configuration and Spark Configuration.

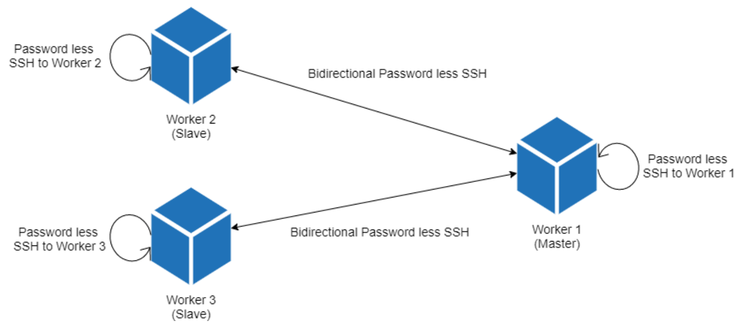


**Figure 1: System Architecture of AWS EC2 instances**

### 2.2 Setup and Configuration

#### 2.2.1 *Passwordless SSH Login*

(1) Generate keys using 'ssh-keygen' on every worker machine.
(2) Copy the public key of worker nodes on the master nodes 'authorized keys' file.
(3) Similarily, copy the public key of master node in the 'authorized keys' file. This will establish a bidirectional SSH login on every master and worker node.
(4) Test by logging into each worker machine from master and vica versa using 'ssh user@ec2-machine-private-ip'

#### 2.2.2 *Hadoop Configuration[3]*

(1) Download and Install Hadoop-2.7.3 on each EC2 machine from the Hadoop download archives.
(2) Make the following changes in the files mentioned below:
    - In core-site.xml, add property 'fs.defaultFS' as value 'ubuntu@master-private-ip'

- In master node's hdfs-site.xml, add property 'dfs.replication' as value '3' and 'dfs.namenode.name.dir' as 'file:///usr/local/hadoop/hdfs/data'
- In master node's mapred-site.xml, add property 'mapreduce.jobtracker.address' as value 'ubuntu@master-private-ip'

(3) Add the 'ubuntu@private-ip-address' of master node in '../hadoop/etc/hadoop/masters' and slave node's in '../hadoop/etc/hadoop/slaves'
(4) Use the command './bin/hdfs namenode -format' to format the HDFS system.
(5) Start the HDFS server by using the command './sbin/start-dfs.sh' on master node and open the web browser and open 'master-public-ip:50070' to check whether installation was successful.

#### 2.2.3 *Spark Configuration[4]*

(1) Download and Install Spark-2.4.5 from the Spark download archives.
(2) Add the slave's private address in the file '/conf/slaves' in the master node.
(3) Execute the command './sbin/start-all.sh' to start the spark process.
(4) Open the web browser, and open 'master-public-ip:8080' to check the status of spark master.

## 2.3 Scala SparkSQL

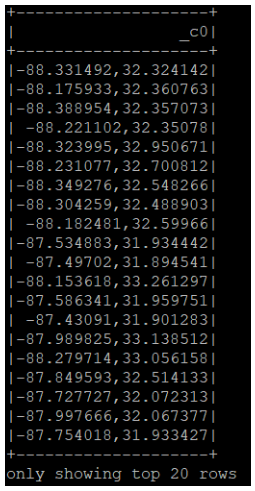
The above setup of passwordless SSH and configuration of Hadoop and Spark is done to execute the geospatial data analysis queries using Scala and SparkSQL.[5] There are two functions and two tasks to be performed in this project. The two user defined SparkSQL function makes use of the spacial queries. The detailed information about the function is given below:

(1) ST-Within
This function is used to check whether the two points are within the boundary point. It takes two points(pt1 and pt2) and distance as input and gives true or false as output. The method used to implement this function is:

(a) As pt1 and pt2 are string and contains the x and y coordinates, it is split into coordinates as (px1,py1) and (px2,py2) respectively. ‘
(b) Find Euclidean distance of pt1 and pt2 by substituting the values of the coordinates in the formula
distance = $\sqrt{(px1 - px2)^2 + (py1 - py2)^2}$
(c) If the value of distance is less than or equal to given distance then the boolean value is returned as True else it is False.

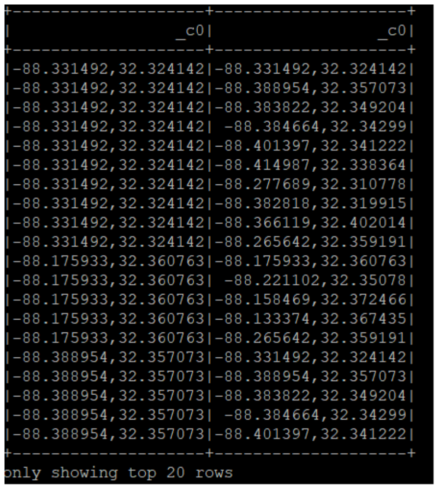
(2) ST-Contains
This fucntion is used to check whether a point is inside the given rectangle. It takes a point(pt) and two rectangle coordinates as input and returns a Boolean value as output. The method used to implement the function is given below:

(a) The point string is split into coordinates as (px, py).
(b) The rectangle string contains two points which indicate the two opposite endpoints of rectangle which are also split into (rx1, ry1) and (rx2, ry2).
(c) If px is between rx1 and rx2 and py is between ry1 and ry2 then the function returns True otherwise False.



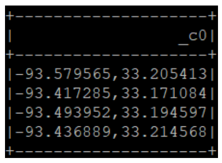
```
+--------------------+
|                 _c0|
+--------------------+
|-88.331492,32.324142|
|-88.175933,32.360763|
|-88.388954,32.357073|
| -88.221102,32.35078|
|-88.323995,32.950671|
|-88.231077,32.700812|
|-88.349276,32.548266|
|-88.304259,32.488903|
| -88.182481,32.59966|
|-87.534883,31.934442|
| -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 2: Distance query output**



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

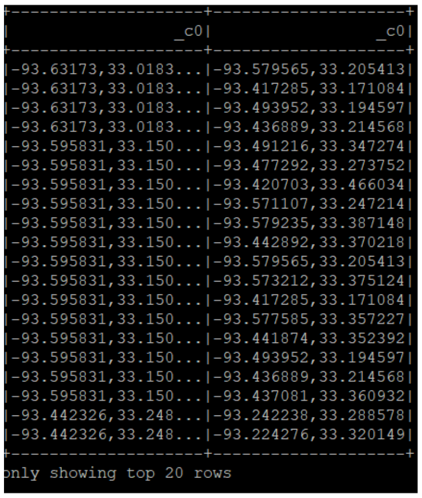
**Figure 3: Distance Join Query output**



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

**Figure 4: Distance query output**

The second phase of the project consisted of two tasks, which required to do spatial hot spot analysis using SparkSQL. The detailed information of the tasks is given below:

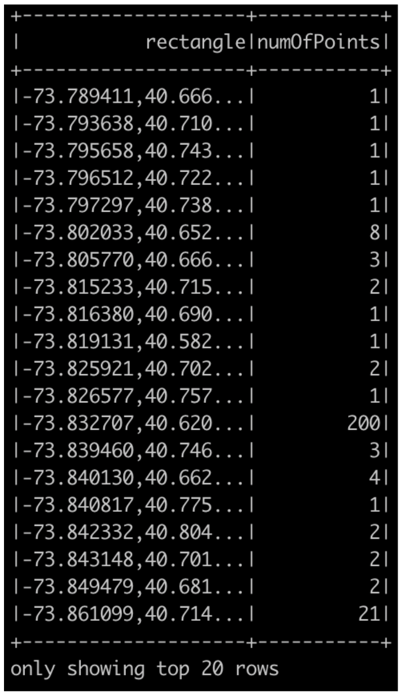


```
+-------------------+-------------------+
|                _c0|                _c0|
+-------------------+-------------------+
|-93.63173,33.0183...|-93.579565,33.205413|
|-93.63173,33.0183...|-93.417285,33.171084|
|-93.63173,33.0183...|-93.493952,33.194597|
|-93.63173,33.0183...|-93.436889,33.214568|
|-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 5: Distance Join query output**

(1) Hot Zone Analysis This analysis is done to perform a range join operation on the rectangle dataset and a point dataset. The result of all the points within each rectangle is obtained by running this query. The rectangle which has more points is considered to be hotter than the one with less number of points in it. The steps involved in implementing this task is given below:
  (a) Load the points and rectangles from pointPath and rectanglePath respectively.
  (b) Run the SparkSQL join query on the points with join condition from the ST-Contains method to obtain the join result.
  (c) On the join result, perform the group by query to get the grouping on rectangles and count of the points, which are sorted by ascending order of rectangle to obtain sorted dataframe of rectangles and their count

(2) Hot Cell Analysis This analysis is done to perform the task of creating a list of cells with highest G-Score(Getis - Ord statistic value) for the given.[6] The term cell here refers to a space-time cube with x and y axis for latitude and longitude of pickup location respectively and z axis represents the date of the pickup location. The steps involved in implementing this task is given below:
  (a) Load the points from the pointPath.
  (b) Run the SparkSQL query on the points which will give a dataframe as (x,y,z) where x is latitude, y is longitude and z is date of pickup.
  (c) Find the G-Score for each of the row in the dataframe and return the top 50 cells in descending order of G-Score[7]

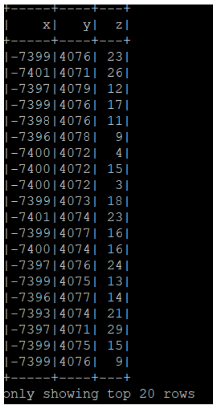


```
+--------------------+-----------+
|           rectangle|numOfPoints|
+--------------------+-----------+
|-73.789411,40.666...|          1|
|-73.793638,40.710...|          1|
|-73.795658,40.743...|          1|
|-73.796512,40.722...|          1|
|-73.797297,40.738...|          1|
|-73.802033,40.652...|          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 6: Hot Zone Analysis Output**

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

**Figure 7: Getis-Ord statistical formula**



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

**Figure 8: Hot Cell Analysis output**

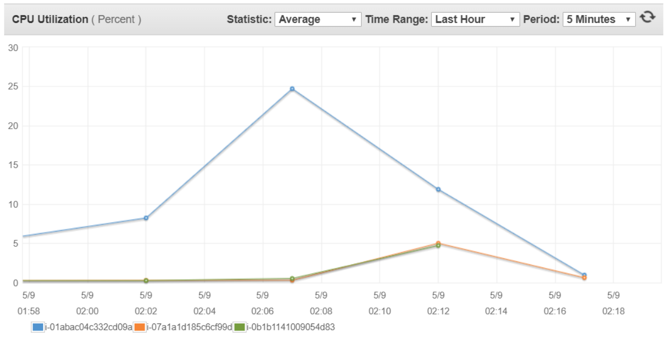
## 2.4 AWS Cloud Watch

AWS Cloud Watch is a monitoring tool and observation tool used to monitor various metrics along various AWS services such as EC2. It collects monitoring and observational data in form of logs, graphs, metrics, etc.[8]. The AWS cloud watch is used here to calculate the CPU utilization of the cluster i.e. when the spark script runs on a single node or more than one node. It also monitors the network packet transmitted between the cluster.

## 3 Experimental Evaluation

### 3.1 Performance Measures

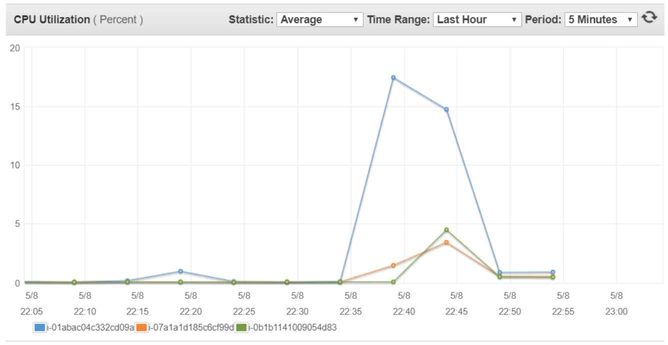
In this section we conducted experiments with the given dataset of Hot Zone point and zone as well as pickup location datset used for Hot Zone and Hot Cell analysis. We calculated the performance and machine utilization on the basis of CPU Utilization, Network Out and In in Bytes. The experiments were conducted on two types of dataset small dataset consisting on 10,000 rows and large dataset which consisted of 50,000 rows. We have used the program for Hot Cell and Hot Zone Analysis to obtain the below graphs with the help of AWS EC2 CloudWatch monitoring system.



CPU Utilization ( Percent ) Statistic: Average Time Range: Last Hour Period: 5 Minutes

**Figure 9: CPU Utilization of Worker 1, 2 and 3 for Large dataset**

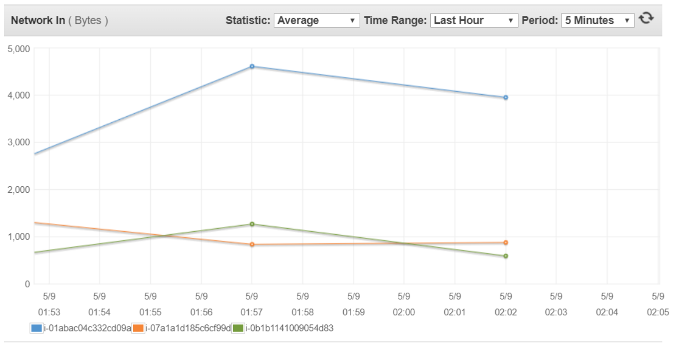
Figure 9 shows the CPU utilization of running spark submit script on a small data set. It shows how the spark distributes load on the slave nodes once the CPU utilization of master machine reaches its threshold. The above figure depicts CPU utilization at 5 min intervals for three workers with each worker shown in different color. Here the master node starts early processing with the slaves getting little workload afterwards.



CPU Utilization ( Percent ) Statistic: Average Time Range: Last Hour Period: 5 Minutes

**Figure 10: CPU Utilization of Worker 1, 2 and 3 for Large dataset**

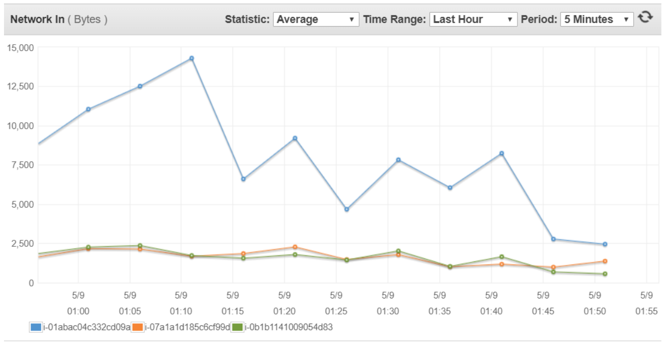
Similarly, Figure 10 shows the CPU utilization of running spark submit script on a large data set. This graph shows that most of work is done on worker 1 and slaves have CPU usage around 10 percentage,



Network In ( Bytes ) Statistic: Average Time Range: Last Hour Period: 5 Minutes

**Figure 11: Network In in Bytes of Worker 1, 2 and 3 for Large dataset**

Figure 11 shows the network packets in while running spark submit script on a small data set. The network packets indicates the bytes of data that gets transferred in the cluster to execute the spark script. It helps in identifying the number of packets received by each worker node in the cluster. The above figure depicts network packets received by each worker node at 5 min intervals with each worker shown in different color. The number of bytes going to master node is more than slaves which is ismilar to CPU utilization



Network In ( Bytes ) Statistic: Average Time Range: Last Hour Period: 5 Minutes

**Figure 12: Network In in Bytes of Worker 1, 2 and 3 for Large dataset**

Similarly, Figure 12 shows the network packets in while running spark submit script on a large data set. For large dataset, there are a few highs and lows in incoming traffic this shows that data is coming in at different rate.

Figure 13 shows the network packets out while running spark submit script on a small data set. It helps in identifying the number of packets sent out by each worker node in the cluster. The above figure depicts network packets received by each worker node at 5 min intervals with each worker shown in different color.

Similarly, Figure 14 shows the network packets out while running spark submit script on a large data set.

### 3.2 Runtime and Scalability Measures

We conduceted tests by using 3 worker nodes(1 master and 2 slaves). We performed the Hot Cell and Hot Zone Analysis on varying sizes

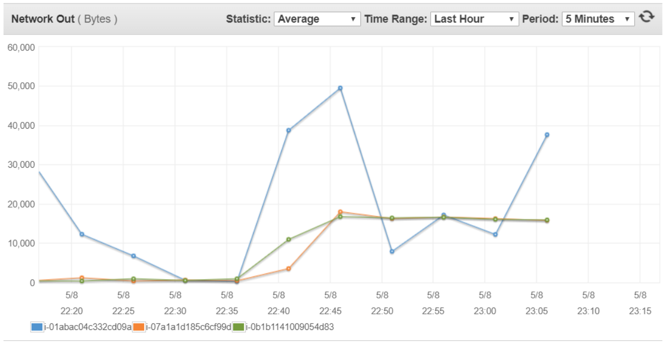


**Figure 13: Network Out in Bytes of Worker 1, 2 and 3 for Large dataset**

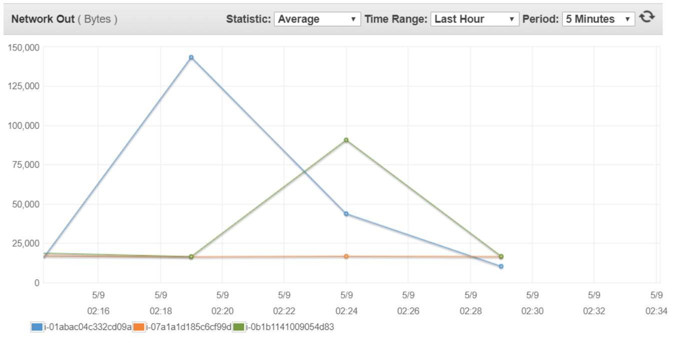


**Figure 14: Network Out in Bytes of Worker 1, 2 and 3 for Large dataset**

of datasets i.e. we started with 10,000 rows and with incrementation of 10,000 reached 50,000 rows. The graph depicting the runtime for each execution is given below.

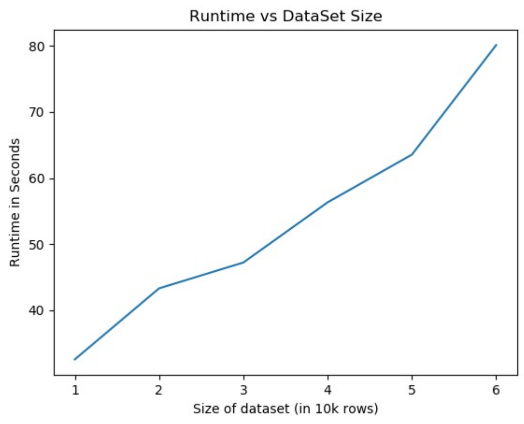


**Figure 15: Run-time for increasing sizes of data-sets**

We also conducted experiments to check scalability measures of this project by using different number of worker nodes. The graph depicting the run-time for each case is described in Figure 15. With increasing the number of worker nodes the run-time of the application decreases gradually.

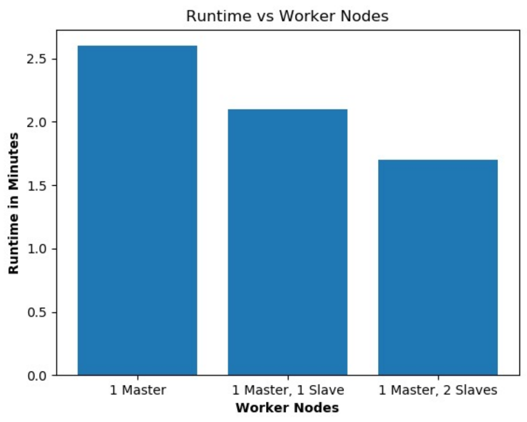


**Figure 16: Runtime for increasing number of worker nodes**

## 4 Conclusion

The main objective of this project was to perform two types of spatial data analysis, namely "Hot zone analysis" and "hot cell/hotspot analysis". This task was taken from ACM SIGSPATIAL GISCUP 2016 topic which was focused on implementing spatial statistics to Spatio-temporal big data to identify statistically notable spatial hot spots using Apache Spark cluster. Spark executes faster by caching data in memory across multiple parallel operations and provides better parallelism, and better CPU utilization. Thus, using Spark we were able to achieve high throughput and low latency. Running Spark on top of the Hadoop distributed file system has made us analyze that, as Spark is not tied to the two-stage MapReduce paradigm, it performs up to 100 times faster than Hadoop Map-Reduce. The last phase of this project has helped us examine the scalability and efficiency our of project, by running several experiments that included feeding different sized input data-sets, changing the number of cores and slave nodes.

## References

[1] J ia Yu, Zongsi Zhang, Mohamed Sarwat: Spatial Data Management in Apache Spark: The GeoSpark Perspective and Beyond. GeoInformatica Journal, 2018
[2] http://sigspatial2016.sigspatial.org/giscup2016/home
[3] https://hadoop.apache.org/.
[4] https://spark.apache.org/
[5] Spark: Cluster Computing with Working Sets. Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. HotCloud 2010. June 2010. http://people.csail.mit.edu/matei/papers/2010/hotcloudspark.pdf
[6] Panagiotis Nikitopoulos, Aris-Iakovos Paraskevopoulos, Christos Doulkeridis, Nikos Pelekis and Yannis Theodoridis. Hot Spot Analysis over Big Trajectory Data. pp 2. https://www.ds.unipi.gr/prof/cdoulk/papers/bigdata18.pdf
[7] Songchitruksa, Praprut Zeng, Xiaosi. (2010). Getis–Ord Spatial Statistics to Identify Hot Spots by Using Incident Management Data. Transportation Research Record Journal of the Transportation Research. https://www.researchgate.net/publication/267097975
[8] AWS - Cloud Watch and its usage, https://aws.amazon.com/cloudwatch/